Supplement

PostScript® Language Reference Manual

LanguageLevel 3 Specification
and

Adobe® PostScript® 3™
Version 3010
Product Supplement



10 October 1997

Adobe Systems Incorporated

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000

Eastern Regional Office
24 New England
Executive Park
Burlington, MA 01803
(617) 273-2120

Adobe Systems Europe Limited
Adobe House, Mid New Cultins
Edinburgh EH11 4DU
Scotland, United Kingdom
+44-131-453-2211

Adobe Systems Japan
Yebisu Garden Place Tower
4-20-3 Ebisu, Shibuya-ku
Tokyo 150 Japan
+81-3-5423-8100

970028-006

© 1997 Adobe Systems Incorporated. All rights reserved.

All Rights Reserved. Patents Pending. 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 the prior written permission of Adobe Systems Incorporated.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript and Adobe PostScript 3 also are used as product trademarks for Adobe Systems' implementations of the PostScript language interpreter.

Any references to a “PostScript printer,” a “PostScript file,” or a “PostScript driver” refer to printers, files, and driver programs (respectively) which are written in or support the PostScript language. The sentences in this document that use “PostScript language” as an adjective phrase are so constructed to reinforce that the name refers to the standard language definition as set forth by Adobe Systems Incorporated.

Adobe and the Adobe logo, the PostScript logo, Chameleon, ColorBurst, Display PostScript, Pixelburst, and TranScript are trademarks of Adobe Systems Incorporated. AppleTalk and LocalTalk are registered trademarks of Apple Computer, Inc. IPX and SunOS are trademarks of Sun Microsystems, Inc. TrueType is a trademark of Apple Computer, Inc. NetWare is a registered trademark of Novell, Inc. NeXTSTEP is a trademark of NeXT, Inc. PANTONE is a registered trademark and Hexachrome is a trademark of Pantone, Inc. IBM is a registered trademark of International Business Machines Corporation. ITC Stone is a registered trademark of International Typeface Corporation. Helvetica, Palatino, and Times Roman are registered trademarks of Linotype-Hell AG and/or its subsidiaries. Microsoft, MS-DOS, and Windows are registered trademarks of Microsoft Corporation. Times New Roman is a registered trademark of The Monotype Corporation PLC. Other brand or product names are the trademarks or registered trademarks of their respective holders.

Statement of Liability

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranties of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringment of third-party rights.

Preface

In the 1980s, Adobe devised a powerful graphics imaging model that over time has formed the basis for the Adobe PostScript technologies. These technologies—a combination of the PostScript language and PostScript language-based graphics and text-formatting applications, drivers, and imaging systems—have forever changed the printing and publishing world by sparking the desktop and digital publishing revolutions. Since their inception, PostScript technologies have enabled unprecedented control in the look and feel of printed documents and have changed the overall process for designing and printing them as well. The capabilities PostScript makes possible have established it as the industry page-description language standard.

Today, as never before, applications developers and imaging systems vendors support the PostScript language as the industry standard. And we at Adobe accept our responsibility as stewards of this standard to continually advance the standard in response to the creative needs of the industry.

With this third advance of the language, which we call LanguageLevel 3, Adobe has greatly expanded the boundaries of imaging capabilities made possible through the PostScript language. This most recent advance has yielded significant improvements in the efficiency and performance of the language as well as improvements in the quality of final output.

To complement the strengths of LanguageLevel 3, Adobe PostScript 3 imaging system technologies have been engineered to fully exploit the new LanguageLevel 3 constructs, fulfilling the Adobe commitment to provide printing solutions for the broad spectrum of users.

No significant change comes without the concerted effort of many individuals. The work to advance the PostScript language and to create Adobe PostScript 3 imaging system technologies is no exception. Our goal since the introduction of the first Adobe imaging model has been nothing less than to provide the most innovative, meaningful imaging solutions in the

industry. Dedicated Adobe employees and many industry partners have strived to make that goal a reality. We take this opportunity to thank all those who contributed to this effort.

John Warnock and Chuck Geschke

10 October 1997

Contents

Chapter 1: Introduction 17

1.1 About This Document 17

1.2 Intended Audience 17

1.3 Using This Document 17
- The LanguageLevel 3 Icon 17
- The Summary of Changes Listing 18
- Organization of Chapters 18
- Finding Information by Feature 19

1.4 Terminology Used in This Document 21

1.5 Related Publications 21

1.6 Documentation Problems 22

1.7 Statement of Liability 22

Chapter 2: Summary of Changes to the PostScript Language 23

Chapter 3: Additions and Modifications to the Language 33

3.1 Named Resources 34
- Regular Resource Categories 34
- Implicit Resource Categories 42

3.2 Early Name Binding 45
- bind Operator 45

3.3 Filtered File Details 47
- Optional Encode Filters 47
- CloseSource and CloseTarget 47
- CCITTFaxDecode Filter 48
- SubFileDecode Filter 48
- LZWEncode and LZWDecode Filters 48
- FlateEncode and FlateDecode Filters 50
- ReusableStreamDecode Filter 53

3.4 Functions 56
- Function Dictionaries 57

Chapter 4: Additions and Modifications to the Graphics Model 63

4.1 CIE-Based Color Spaces 65
Two New Color Spaces: CIEBasedDEF and CIEBasedDEFG 65

4.2 HiFi and Multitone Color 68
Indexed Color Space 69
DeviceN Color Space 70

4.3 Masked Images 71
Image Dictionaries 72

4.4 Smooth Shading 76
Pattern Dictionaries 77
Shading Dictionaries 77
Painting with a Pattern Dictionary 98

4.5 In-RIP Trapping 99
Setting Trapping Zones for In-RIP Trapping 100
Trapping Parameters 100
Trapping Parameter Dictionary 101
ColorantZoneDetails Dictionary 103

4.6 Device Setup 104
Page Device Parameters 104
Details Dictionaries 116
TrappingDetails and ColorantDetails Dictionaries 119
DeviceRenderingInfo Dictionaries 120
Errors Generated by Page Device Parameters 121
Input Media Selection: Envelope Orientation in User Space 123
New Entries in the Policies Dictionary 124

Chapter 5: Additions and Modifications to Fonts 127

5.1 Multiple Master Font Dictionary 128

5.2 Additional Base Font Types 129
Bitmap Fonts: Font Type 32 129
TrueType Fonts: Font Type 42 131
CFF and Chameleon Fonts: FontType 2 and FontType 14 132

5.3 CID-Keyed Fonts 135
CID Font Resources 136
CIDSystemInfo Dictionary Entries 145
Subsets and Incremental Definition of Glyph Procedures 145
CMap Resources 148
CIDInit ProcSet 149
CMap Example 150

5.4 Composite Font Extension 153
Type 0 Dictionary Entries 153
Character Mapping 153
Undefined (notdef) Character Handling 154
show String Parsing 155
Nesting 155

Chapter 6: Additions and Modifications to the Rendering Model 157

6.1 UseCIEColor 158

6.2 Color Rendering 159
CRD Selection Based on Rendering Intent 159
Synchronizing CRDs and ICC Profiles 163

6.3 Halftone Dictionaries 166
Changes to the Type 1 through Type 5 Halftone Dictionaries 167
New Halftone Dictionaries 167
Type 6 Halftone Dictionary 168
Type 9 Halftone Dictionary 169
Type 10 Halftone Dictionary 170
Type 16 Halftone Dictionary 172
Type 100 Halftone Dictionary 174

6.4 Extensions to Process Color Models 175
Device-Dependent Color Conversions for ProcessColorModel DeviceN 176

6.5 Transfer Functions 176

Chapter 7: Additions and Modifications to Display PostScript® 177

7.1 Type 2 Image Dictionary 178
Uses of Type 2 Images 178
Entries in a Type 2 Image Dictionary 179
Type 2 Image Dictionary Extensions 180

7.2 The Device Pixel Color Space 185
Drawing with a Device Pixel Color Model 185
Drawing with the Display PostScript Color Model 186
Specifying Custom Colors 187
Performing Color Map Animation 188
Coloring Overlapping Areas 189
Device Pixel Color Space and Layered Windows 192

Chapter 8: Additions and Modifications to the Operators 193

Chapter 9: LanguageLevel 1/ LanguageLevel 2 Compatibility 209

9.1 Compatibility Operators and Keys Listed by Dictionary and by Function 211
By Dictionary 211
By Function 212

9.2 Operator and Key Details 214

9.3 Compatibility Operations 233
Error Behavior 234
Changing Persistent Values with Password 234
SCC Operations 235
Paper Size Operations 238
Paper Tray Operations 239

Chapter 10: Additions and Modifications to the Interpreter Parameters 241

10.1 Overview 243

10.2 User Parameters 243

10.3 System Parameters 247

10.4 Administrator Jobs and Passwords 258
Unencapsulated Jobs 258
Passwords for System and Device Parameters 259

10.5 Device Parameters 259
Communications Parameter Sets (Type = /Communications) 262
Parameters Parameter Sets (Type = /Parameters) 303
File System Parameter Sets (Type = /FileSystem) 328

Appendix A: Acronyms 341

Appendix B: Implementation Limits 345

Appendix C: Emulators 347

C.1 Emulator Parameter Sets (Type = /Emulator) 347

Appendix D: Updates to the PostScript Language Reference Manual, Second Edition 355

D.1 Introduction 355

D.2 Errata for the Fifth and Subsequent Printings 355
Chapter 3 Errata 356
Chapter 4 Errata 359
Chapter 5 Errata 361
Chapter 6 Errata 361
Chapter 8 Errata 362
Appendix B Errata 367
Appendix E Errata 368
Appendix G Errata 368
Appendix H Errata 368
Appendix I Errata 369
Index Errata 369

D.3 Changes Made in the Third and Fifth Printings 370
Chapter 3 Changes 370
Chapter 4 Changes 371
Chapter 5 Changes 371
Chapter 6 Changes 372
Chapter 7 Changes 372
Chapter 8 Changes 373
Appendix C Changes 375
Appendix D Changes 375
Appendix G Changes 376

Appendix E: Standard Fonts, Character Sets, and Encodings 377

E.1 Typical Font Set Available on Adobe PostScript 3 Products 377

E.2 New Symbol Font Character 382

E.3 CE Character Encoding Values 382

E.4 CE Characters Not Encoded 390

Appendix F: System Name Encodings 393

Appendix G: Bibliography 395

G.1 *Books 395*

G.2 Technical Notes from Adobe Systems Incorporated 395

G.3 Other 396

Index 397

Figures

Figure 3.1 Mapping with the Decode array 60

Figure 4.1 Original CIEBasedABC structure 66
Figure 4.2 CIEBasedDEF(G) pre-extension to the CIEBasedABC color spaces 66
Figure 4.3 Drawing a new triangle (f = 0): Free-form Gouraud-shaded triangle meshes 87
Figure 4.4 How the value of the edge flag determines which edge is used for the next triangle 87
Figure 4.5 Varying the value of the edge flag to create different shapes 88
Figure 4.6 Sample lattice forms 89
Figure 4.7 Coons patch meshes: Coordinate mapping from a unit square to a four-sided patch 91
Figure 4.8 Coons patch meshes: Appearance, painted area, and boundary 92
Figure 4.9 Color values and edge flags in Coons patch meshes 94
Figure 4.10 Coons patch meshes: How the value of the edge flag determines the edge for the next patch 96
Figure 4.11 An example of trapping 99
Figure 4.12 Default user space orientation for an envelope with a PageSize value of portrait 124

Figure 6.1 Halftone cell graphically represented in device space 170
Figure 6.2 Halftone cell divided into two squares 170
Figure 6.3 Halftone cell and two squares tiled across device space 171
Figure 6.4 Tiling the device space in a type 16 halftone dictionary 173

Figure 7.1 Imaging source sample data into a destination device 182
Figure 7.2 Transforming from image space to current user space 183
Figure 7.3 Drawing with a typical display system’s color mechanism 186
Figure 7.4 Drawing with the Display PostScript color mechanism 187
Figure 7.5 Frame buffer with overlapping areas rendered on screen 189
Figure 7.6 Two irregular overlapping areas 191

Figure 10.1 The OSI (Open Systems Interconnection) layers 261
Figure 10.2 Relationship between network communications protocols and physical communications media 263
Figure 10.3 Relationship among the communication parameter sets 265
Figure 10.4 Communications parameter sets using NV values 267
Figure 10.5 Communications parameter sets using “hard-wired” values 268

Tables

Table 1.1 Finding information on features 19

Table 3.1 Regular resources 34
Table 3.2 Entries in an instance of the ControlLanguage resource category 36
Table 3.3 Instances of the HWOptions resource category 37
Table 3.4 Entries in an instance of the OutputDevice resource category 39
Table 3.5 Entries in an instance of the PDL resource category 41
Table 3.6 Entries in a Trapping ProcSet dictionary 42
Table 3.7 Resources whose instances are implicit 43
Table 3.8 Entries that may appear in any Filter dictionary 47
Table 3.9 New entries in an LZWDecode dictionary 48
Table 3.10 Optional entries in a FlateEncode dictionary 51
Table 3.11 Predictor-related entries in Flate and LZW encode and decode dictionaries 52
Table 3.12 Entries in a ReusableStreamDecode dictionary 55
Table 3.13 Entries common to all Function dictionaries 57
Table 3.14 Entries specific to a FunctionType 0 Function dictionary (sampled functions) 58
Table 3.15 Entries specific to a FunctionType 2 Function dictionary (exponential interpolation) 61
Table 3.16 Entries specific to a FunctionType 3 Function dictionary (1-input stitching) 61

Table 4.1 Entries in a CIEBasedDEFG color space dictionary 67
Table 4.2 Entries in a type 3 image dictionary 72
Table 4.3 Entries in a MaskDict dictionary 73
Table 4.4 Entries in a DataDict dictionary 74
Table 4.5 Entries in a type 4 image dictionary 75
Table 4.6 Entries in a type 2 pattern dictionary 77
Table 4.7 Entries common in all Shading dictionaries 78
Table 4.8 Entries in a ShadingType 1 Shading dictionary (function-based shadings) 81
Table 4.9 Entries in a ShadingType 2 Shading dictionary (axial shadings) 81
Table 4.10 Entries in a ShadingType 3 Shading dictionary (radial shadings) 83
Table 4.11 Entries in a ShadingType 4 Shading dictionary (free-form Gouraud-shaded

triangle meshes) 85
Table 4.12 Entries in a ShadingType 5 Shading dictionary (lattice-form Gouraud-shaded triangle meshes) 89
Table 4.13 Entries in a ShadingType 6 Shading dictionary (Coons patch meshes) 93
Table 4.14 Coons patch meshes: Coordinates for adjacent patches 95
Table 4.15 Entries in a trapping parameter dictionary 101
Table 4.16 Entries in a colorant dictionary 103
Table 4.17 Entries in the page device dictionary 105
Table 4.18 Typical entries in a CollateDetails dictionary 118
Table 4.19 Typical entries in a PostRenderingEnhanceDetails dictionary 118
Table 4.20 Entries in a TrappingDetails dictionary 119
Table 4.21 Entries in a ColorantDetails dictionary 119
Table 4.22 Entries in a device colorant name dictionary 120
Table 4.23 Typical entries in a DeviceRenderingInfo dictionary 121
Table 4.24 Entries in a Policies dictionary 124

Table 5.1 Entries specific to a multiple master font dictionary 129
Table 5.2 Entries in a Type 32 font dictionary 130
Table 5.3 Additional entries in a Type 42 font dictionary 132
Table 5.4 CIDFontType and FontType values 137
Table 5.5 Entries in all types of CIDFont dictionaries 138
Table 5.6 Additional entries in a CIDFontType 0 CIDFont dictionary 140
Table 5.7 Entries in a dictionary in the FDArray 142
Table 5.8 Additional entries in a Private dictionary of a CIDFontType 0 CIDFont 142
Table 5.9 Additional entries in a CIDFontType 1 CIDFont dictionary 143
Table 5.10 Additional entries in a CIDFontType 2 CID fonts 144
Table 5.11 Entries in a CIDSystemInfo dictionary 145
Table 5.12 Entries in a CMap dictionary 148
Table 5.13 Entry specific to a Type 0 (composite) font dictionary 153
Table 5.14 New FMapType mapping algorithm 154

Table 6.1 Rendering intents 161
Table 6.2 Format of crdInfoTag 164
Table 6.3 New entry for halftone dictionaries of type 1 through type 5 167
Table 6.4 Halftone dictionaries 167
Table 6.5 Entries in a type 6 halftone dictionary 168
Table 6.6 Entries in a type 9 halftone dictionary 169
Table 6.7 Entries in a type 10 halftone dictionary 172
Table 6.8 Entries in a type 16 halftone dictionary 173
Table 6.9 Entries in a type 100 halftone dictionary 174

Table 7.1 Entries in a type 2 image dictionary 179
Table 7.2 X window system drawing functions and their logical operations 190
Table 7.3 Example colors and index assignments for OR drawing function 192

Table 9.1 Stop bits 235
Table 9.2 Data bits 236
Table 9.3 Flow control 236
Table 9.4 Parity 236

Table 9.5 Options byte to device parameters conversion 237
Table 9.6 Positions 1 and 0 237
Table 9.7 Paper size compatibility operators (in userdict) 238
Table 9.8 Paper tray compatibility operators 239

Table 10.1 User parameters 244
Table 10.2 System parameters 248
Table 10.3 Entries in all communications parameter sets (Type = /Communications) 269
Table 10.4 Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets 274
Table 10.5 Entries in the %Parallel%, %ParallelB%, %ParallelC%, … parameter sets 279
Table 10.6 Entries in the %ScsiComm%, %ScsiCommB%, %ScsiCommC%, … parameter sets 281
Table 10.7 Entries in the %LPR%, %LPRB%, %LPRC%, … parameter sets 286
Table 10.8 Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets 288
Table 10.9 Entries in the %Telnet%, %TelnetB%, %TelnetC%, … parameter sets 291
Table 10.10 Entries in the %RemotePrinter%, %RemotePrinterB%, %RemotePrinterC%, … parameter sets 292
Table 10.11 Entries in the %PrintServer%, %PrintServerB%, %PrintServerC%, … parameter set 293
Table 10.12 Entries in the %LAT%, %LATB%, %LATC%, … parameter sets 295
Table 10.13 Entries in the %LocalTalk%, %LocalTalkB%, %LocalTalkC%, … parameter sets 297
Table 10.14 Entries in the %EtherTalk%, %EtherTalkB%, %EtherTalkC%, … parameter sets 299
Table 10.15 Entries in the %TokenTalk%, %TokenTalkB%, %TokenTalkC%, … parameter sets 301
Table 10.16 Form of the node address 308
Table 10.17 Entries in the %SNMP%, %SNMPB%, %SNMPC%, … parameter sets 308
Table 10.18 Entries in the %Syslog%, %SyslogB%, %SyslogC%, … parameter sets 310
Table 10.19 Entries in the %TCP%, %TCPB%, %TCPC%, … parameter sets 311
Table 10.20 Entries in the %UDP%, %UDPB%, %UDPC%, … parameter sets 311
Table 10.21 Entries in the %IP%, %IPB%, %IPC%, … parameter sets 312
Table 10.22 Entries in the %SPX%, %SPXB%, %SPXC%, … parameter sets 316
Table 10.23 Entries in the %IPX%, %IPXB%, %IPXC%, … parameter sets 316
Table 10.24 Entries in the %EthernetPhysical%, %EthernetPhysicalB%, %EthernetPhysicalC%, … parameter sets 318
Table 10.25 Entries in the %TokenRingPhysical%, %TokenRingPhysicalB%, %TokenRingPhysicalC%, … parameter sets 319

Table 10.26 Entries in the %Scsi%, %ScsiB%, %ScsiC%, … parameter sets 320
Table 10.27 Entries in the %Ide%, %IdeB%, %IdeC%, … parameter sets 322
Table 10.28 Entries in the %Engine%, %EngineB%, %EngineC%, … parameter sets 322
Table 10.29 Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets 324
Table 10.30 Entries in the %Calendar%, %CalendarB%, %CalendarC%, … parameter sets 327
Table 10.31 Entries in the %disk0%, %disk1%, %disk2%, … parameter sets 330
Table 10.32 Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets 334
Table 10.33 Entries in the %ram%, %ram1%, %ram2%, … parameter sets 337
Table 10.34 Entries in the %os%, %osB%, %osC%, … parameter sets 339

Table C.1 Entries in the %PCL% parameter set (when used as LaserJet 4 emulator) 349
Table C.2 Entries in the %LaserJetIII% parameter set (LaserJet III emulator) 349
Table C.3 Entries in the %LaserJetIIP% parameter set (LaserJet IIP emulator) 353
Table C.4 Entries in the %HP7475A% (color version) parameter set (HP7475A plotter emulator) 354
Table C.5 Entries in the %Diablo630% parameter set (Diablo630 emulator) 354

Table E.1 Adobe PostScript 3 font set 377
Table E.2 New Symbol character 382
Table E.3 CE character encodings 382
Table E.4 CE characters that are not encoded 390

CHAPTER 1

Introduction

1.1 About This Document

This *Supplement* describes the formal extensions made to the PostScript language since the publication of the *PostScript Language Reference Manual, Second Edition*. It serves three purposes:

- It provides a timely specification of LanguageLevel 3 operators and features that will be documented in the upcoming *PostScript Language Reference Manual, Third Edition*.
- It describes the extensions to the language as instantiated in Adobe PostScript 3 software.
- It describes all other extensions that were made to the language between the publication of the *PostScript Language Reference Manual, Second Edition*, and the advent of LanguageLevel 3 and Adobe PostScript 3 software.

1.2 Intended Audience

This document is intended for anyone who writes programs that use the PostScript language, writes drivers that generate PostScript language commands as their output, or writes software that interprets programs written in the PostScript language.

1.3 Using This Document

This document has been designed to accommodate readers trying to find information about LanguageLevel 3 extensions and the Adobe PostScript 3 instantiation.

1.3.1 The LanguageLevel 3 Icon

All new entries in this *Supplement* that describe LanguageLevel 3 extensions are flagged with the icon: 

1.3.2 The Summary of Changes Listing

A new release of this *Supplement* is made following the release of each new version of Adobe PostScript software. Additions and modifications to the text reflect additions and modifications to the definition of the PostScript language or to the Adobe PostScript software instantiation.

To locate those additions and modifications, use the summary of changes listing (Chapter *2*, “Summary of Changes to the PostScript Language,” on page 23). This listing shows the PostScript language extensions associated with the various versions of Adobe PostScript software released since the publication of the *PostScript Language Reference Manual, Second Edition*, and gives the pages where those extensions are described in this *Supplement*.

1.3.3 Organization of Chapters

The organization of this *Supplement* has been changed to mirror that of the *PostScript Language Reference Manual, Second Edition*, with the following exceptions:

- Chapter 2, rather than describing the basic ideas behind the PostScript technology, lists the changes made to the language since the publication of the *PostScript Language Reference Manual, Second Edition*.
- Chapter 9 has been added to describe how LanguageLevel 2 and LanguageLevel 3 can be made compatible with LanguageLevel 1 constructs.
- Chapter 10 has been added to describe interpreter parameters.
- Appendix A defines the acronyms appearing in this *Supplement*.
- Appendix C describes emulator-related issues.
- Appendix D summarizes all the updates made to the *PostScript Language Reference Manual, Second Edition*.
- Appendix G is a bibliography.

1.3.4 Finding Information by Feature

The following table maps feature names used commonly in public announcements about LanguageLevel 3 and Adobe PostScript 3 to the implementation particulars documented in this *Supplement.* For a complete listing of extensions to the PostScript language, see Chapter *2,* “Summary of Changes to the PostScript Language,” on page 23.

Table 1.1 *Finding information on features*

Feature	*Location*
Enhanced Image Technology	
Graphics features that benefit image quality and color	
Smooth Shading	*Renders gradient fills at the resolution of the printing system, reducing print time and improving output quality.* • Section 4.4, “Smooth Shading,” on page 76. Smooth shading also uses the **ReusableStreamDecode** filter and **Function** objects: • Section 3.3.7, “ReusableStreamDecode Filter,” on page 53. • Section 3.4, “Functions,” on page 56.
Masked Images	*Replaces complex clipping paths with a raster mask.* • Section 4.3, “Masked Images,” on page 71.
Selectable Separations	*Enables separations to be printed, even from low-end monochrome printing systems.* • **MaxSeparations** in Table 4.17 on page 105. • **DeviceN** in . • Section 4.2.1, “Indexed Color Space,” on page 69.
More Gray Levels	*Provides monochrome desktop printing systems with the ability to print photoquality grayscales (up to 256 levels).* • Section 6.3, “Halftone Dictionaries,” on page 166. • **MaxSuperScreen** in Table 10.1 on page 244. *For high resolution printing systems, delivers photoquality grayscales and 4,096 levels of each colorant.* • Section 6.3.6, “Type 16 Halftone Dictionary,” on page 172.

Table 1.1 *Finding information on features (continued)*

Feature	*Location*
In-RIP Trapping	*Allows files sent to print to be trapped in the RIP.* • Section 4.5, “In-RIP Trapping,” on page 99. • Section 4.6.3, “TrappingDetails and ColorantDetails Dictionaries,” on page 119. • **Trapping** and **TrappingDetails** in Table 4.17 on page 105. • **InkParams** on page 37, **Trapping** on page 42, and **TrappingParams** on page 42.
HiFi Color	*Provides more vibrant hues and richer colors.* • Section 4.2, “HiFi and Multitone Color,” on page 68.
Improved Color Control	*Provides greater control with respect to overprint between color components.* • Section 4.2.2, “DeviceN Color Space,” on page 70.
Advanced Page Processing	
Enhancements for improved performance	
Idiom Recognition	*Converts less efficient legacy constructs into higher quality, faster LanguageLevel 3 constructs.* • “IdiomSet” on page 37. • Section 3.2.1, “bind Operator,” on page 45. • **IdiomRecognition** in Table 10.1 on page 244.
Fast Image Printing	*Enables fast draft mode and near-final quality raster-image printing.* • **HalftoneMode** and Table 10.1 on page 244.
The Adobe PostScript 3 Improved Font Technology and Extended Font Set	
	• Section 5.2.3, “CFF and Chameleon Fonts: FontType 2 and FontType 14,” on page 132. • Appendix E, “Standard Fonts, Character Sets, and Encodings,” on page 377.

1.4 Terminology Used in This Document

Throughout this document, the following terms are used:

- *LanguageLevel*. A means for classifying a set of PostScript language capabilities. For instance, a feature identified as “LanguageLevel 3” is guaranteed to be present in a PostScript interpreter whose **systemdict languagelevel** operator reports 3 or greater, or whose PPD **LanguageLevel** entry indicates 3 or greater.

- *Device*. A device is defined as a piece of hardware, or a piece of software emulating hardware, under the control of a PostScript interpreter. There are several categories of devices, including:

  Communications device. A communications device can be serial, parallel, or LocalTalk® communications hardware and software, for example.

  File system device. A file system device can be a disk or cartridge system, for example.

- *Page device.* For example, a laser print engine producing paper output.

- *Host*. A host is defined as a computer system (for example, a personal computer or workstation) connected to a PostScript imaging system via one of its communications devices. The host sends PostScript language programs over the communications channel to the printer. The printer executes the programs.

- *PostScript interpreter.* A PostScript interpreter is defined as a body of software that executes programs written in the PostScript language to produce effects such as generating printed output on a page device.

- *PostScript printer product.* A PostScript printer product is defined as an imaging system consisting of a PostScript interpreter producing printer output.

- *Adobe PostScript 3 software*. Software that is licensed by Adobe to OEMs and that is used by OEMs to create imaging systems.

1.5 Related Publications

For a list of related publications, see Appendix G, “Bibliography,” on page 395.

1.6 Documentation Problems



If you discover any errors in or have any problems with this documentation, please e-mail us at:

doc_problems@adobe.com

Please describe the error or problem as completely as possible and give us the document id number (located at the foot of the cover page), the document title, and the page number or page range.

1.7 Statement of Liability

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranties of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringment of third-party rights.

CHAPTER 2

Summary of Changes to the PostScript Language

The following table highlights additions and modifications to the PostScript language since the publication of the *PostScript Language Reference Manual, Second Edition*. For each PostScript system version number, additions and modifications are listed by type. The "*Location in this Supplement*" entry gives the primary reference within this *Supplement*. If you are viewing this document on-line, click on the *Location* reference to go directly to the referenced material.

		Location in this Supplement
Changes to 2017 for 3010		
Page device parameters	**MaxSeparations**	Table 4.17 on page 105
	ProcessColorModel	Table 4.17 on page 105
	SeparationColorNames	Table 4.17 on page 105
	SeparationOrder	Table 4.17 on page 105
	Separations	Table 4.17 on page 105
	Trapping	Table 4.17 on page 105
	TrappingDetails	Table 4.17 on page 105 Table 4.20 on page 119
	UseCIEColor	Table 4.17 on page 105
User parameters	**HalftoneMode**	Table 10.1 on page 244
	IdiomRecognition	Table 10.1 on page 244
	MaxSuperScreen	Table 10.1 on page 244
Dictionaries	Colorant	Table 4.16 on page 103
	ColorantDetails	Table 4.21 on page 119
	ColorantZoneDetails	Section 4.5.4 on page 103
	DataDict	Table 4.4 on page 74
	Device colorant name	Table 4.22 on page 120

		Location in this Supplement
	Function	Table 3.13 on page 57 to Table 3.16 on page 61
	MaskDict	Table 4.3 on page 73
	ReusableStreamDecode	Table 3.12 on page 55
	Shading	Table 4.7 on page 78 to Table 4.13 on page 93
	TrappingDetails	Table 4.20 on page 119
	Trapping parameter dictionary	Table 4.15 on page 101
	Type 2 pattern dictionary	Table 4.6 on page 77
	Type 3 image dictionary	Table 4.2 on page 72
	Type 4 image dictionary	Table 4.5 on page 75
	Type 16 halftone	Table 6.8 on page 173
System parameters	**MaxDisplayAndSourceList**	Table 10.2 on page 248
	SourceListBypass	Table 10.2 on page 248
Device parameters	%ram%	Table 10.33 on page 337
Resources	**FontSet**	Page 133
	FunctionType	Section 3.4 on page 56
	IdiomSet	Page 37
	InkParams	Page 37
	OutputDevice **DeviceN** **TrappingDetailsType** **MediaClass**	Table 3.4 on page 39
	ProcSet **Trapping**	Page 41
	ShadingType	Section 4.4.2 on page 77
	TrapParams	Page 42
Fonts	CFF and Chameleon fonts	Section 5.2.3 on page 132
Filters	**CCITTFaxDecode**	Section 3.3.3 on page 48
	Encoding filters	Section 3.3 on page 47
	FlateEncode, FlateDecode	Section 3.3.6 on page 50
	LZWEncode, LZWDecode	Section 3.3.5 on page 48

		Location in this Supplement
	ReusableStreamDecode	Section 3.3.7 on page 53
	SubFileDecode	Section 3.3.4 on page 48
Color space	**DeviceN**	Section 4.2.2 on page 70
	Indexed	Section 4.2.1 on page 69
Operators	**addglyph**	Chapter 8 on page 194
	bind	Chapter 8 on page 196
	cliprestore	Chapter 8 on page 196
	clipsave	Chapter 8 on page 196
	copypage	Chapter 8 on page 197
	currentcolor	Chapter 8 on page 199
	currentsmoothness	Chapter 8 on page 199
	currenttrapparams	Chapter 8 on page 199
	setcolor	Chapter 8 on page 203
	setcolorspace	Chapter 8 on page 203
	setsmoothness	Chapter 8 on page 204
	settrapparams	Chapter 8 on page 205
	settrapzone	Chapter 8 on page 206
	shfill	Chapter 8 on page 206
	widthshow	Chapter 8 on page 208
Compatibility operators	**setpageparams**	Chapter 9 on page 230
	setpage	Chapter 9 on page 229
	Changes to 2016 for 2017	
Page device parameters	**CollateDetails**	Table 4.17 on page 105
	LeadingEdge	Table 4.17 on page 105
	MediaClass	Table 4.17 on page 105
	OutputDevice **OutputDeviceDict** (removed)	Table 4.17 on page 105
	PageDeviceName **MediaClass**	Table 4.17 on page 105
	RollFedMedia	Table 4.17 on page 105
Dictionaries	**CollateDetails**	Table 4.18 on page 118

		Location in this Supplement
	Color rendering dictionaries **DeviceRenderingInfo**	Table 4.17 on page 105
	Halftone (all) **HalftoneName**	Section 6.3 on page 166
	PostRenderingEnhanceDetails	Table 4.19 on page 118
	Type 10 halftone **Thresholds**	Table 6.7 on page 172
System parameters	**CompressImageSource**	Table 10.2 on page 248
	EnvironmentSave	Table 10.2 on page 248
	InstalledRam	Table 10.2 on page 248
	MaxHWRenderingBuffer ColorBurst	Table 10.2 on page 248
	MaxPermanentVM	Table 10.2 on page 248
Device parameters	%cartridge% **InitializeAction** **Mounted**	Table 10.32 on page 334
	%disk% **Bus** **InitializeAction** **Mounted** **PrepareAction** **ioerror**	Table 10.31 on page 330
	%Engine% **DarknessBlack** **DarknessCyan** **DarknessMagenta** **DarknessYellow**	Table 10.28 on page 322
	%Ide% (new section)	Table 10.27 on page 322
	%IP% **TransmitEncapsulation**	Table 10.21 on page 312
	%IPX% **HopCount**	Table 10.23 on page 316
	%LaserJetIIP% **WaitTimeout**	Table C.3 on page 353
	%LAT%	Table 10.12 on page 295
	%PrintServer% **PreferredServer**	Table 10.11 on page 293
	%ram%	Table 10.33 on page 337

		Location in this Supplement
Resources	**HalfToneType** Types 9 and 100	Table 6.6 on page 169 and Table 6.9 on page 174
	HWOptions ColorBurst	Page 37
	OutputDevice	Page 38
CIE-based color spaces	**CIEBasedDEF** **CIEBasedDEFG**	Section 4.1 on page 65
	Changes to 2015 for 2016	
Page device parameters	**Jog**	Table 4.17 on page 105
Dictionaries	**HalftoneName**	Section 6.3.1 on page 167
	Policies Type 7 **PageSize**	Table 4.24 on page 124
	Type 32 font dictionary	Section 5.2.1 on page 129
System parameters	**DoPrintErrors**	Table 10.2 on page 248
Device parameters	%CommName_Pending%	Figure 10.3 on page 265
	%ScsiComm% **Filtering**	Table 10.6 on page 281
	%SNMP% **PrivateHost** **TrapHost**	Table 10.17 on page 308
Resources	Type 32 font dictionary	Section 5.2.1 on page 129
Operators	Type 32 fonts **addglyph** **removeall** **removeglyphs**	Chapter 8 on page 194 on page 202 on page 202
Compatibility operators	**doprinterrors**	Section 9.2 on page 219
	processcolors	Section 9.2 on page 225
	setdoprinterrors	Section 9.2 on page 227
CIE-based color spaces	(New section)	Section 4.1 on page 65
Fonts	CID fonts	Section 5.3.1 on page 136
	Changes to 2014 for 2015	
Page device parameters	**PageDeviceName** CRD selection	Table 4.17 on page 105
	PageSize	Table 4.17 on page 105
Dictionaries	Color rendering dictionaries	Section 6.2.1 on page 159

		Location in this Supplement
	Type 10 Halftone	Section 6.3.5 on page 170
Device parameters	/Communications **PrinterControl** **Interpreter** /PCL	Table 10.3 on page 269
	%Console%	Table 10.29 on page 324
	%EtherTalk% **NodeID**	Table 10.14 on page 299
	%Parallel% **Handshake** **nAckPulseWidth** **nStrobeExpectedPulseWidth**	Table 10.5 on page 279
	%PCL%	Table C.1 on page 349
	%TokenRingPhysical%	Table 10.25 on page 319
	%TokenTalk%	Table 10.15 on page 301
Resources	CID font support	Section 5.3.1 on page 136
	ControlLanguage	Page 36
	Localization	Page 37
	PDL	Page 40
	ProcSet **ColorRendering**	Section 6.2.1 on page 159
Operators	**findcolorrendering**	Chapter 8 on page 200
	Changes to 2013 for 2014	
Page device parameters	**DeferredMediaSelection** **ImageShift** **InsertSheet** **MediaPosition** **PageOffset**	Table 4.17 on page 105
	setpagedevice **undefined**	Section 4.6.5 on page 121
Details dictionaries	**Type**	Section 4.6.2 on page 116
System parameters	**CurBufferType** **MinBandBuffers**	Table 10.2 on page 248
	PageCount	Table 10.2 on page 248
Device parameters	%AppSocket% **ControlPortNumber**	Table 10.8 on page 288

		Location in this Supplement
	/AutoSelect	Table 10.3 on page 269
	/Communications **DelayedOutputClose**	Table 10.3 on page 269
	%EthernetPhysical% **Name**	Table 10.24 on page 318
	/FileSystem **Removable**	Section 10.5.3 on page 328
	%IP% **Gateway Address**	Table 10.21 on page 312
	%IPX%	Table 10.23 on page 316
	Node address Novell SPX/IPX	Table 10.16 on page 308
	%PrintServer%	Table 10.11 on page 293
	%RemotePrinter%	Table 10.10 on page 292
	%ScsiComm% **Bus**	Table 10.6 on page 281
	%SNMP% **TrapHost**	Table 10.17 on page 308
	%SPX%	Table 10.22 on page 316
	%UDP%	Table 10.20 on page 311
Resources	**HWOptions** **WorldModem**	Page 37
	IODevice %IPX% %PrintServer% %RemotePrinter% %SPX% %UDP%	Section 3.1.2 on page 42
	OutputDevice **ProcessColorModel**	Page 38
	Changes to 2012 for 2013	
Page device parameters	**PageOffset**	Table 4.17 on page 105
Device parameters	%AppSocket%	Table 10.8 on page 288
	%Diablo630% **Pitch** **Type**	Table C.5 on page 354
	%EthernetPhysical%	Table 10.24 on page 318
	%HP745A% **Type**	Table C.4 on page 354

		Location in this Supplement
	%IP%	Table 10.21 on page 312
	%LaserJetIII%	Table C.2 on page 349
	%LPR%	Table 10.7 on page 286
	%rom%	Table 10.32 on page 334
	%SNMP%	Table 10.17 on page 308
	%Syslog%	Table 10.18 on page 310
	%TCP%	Table 10.19 on page 311
	%Telnet%	Table 10.9 on page 291
Resources	**HWOptions**	Page 37
	IODevice %AppSocket% %EthernetPhysical% %IP% %LPR% %SNMP% %SysLog% %TCP% %Telnet%	Table 3.7 on page 43
	Type 42 font dictionary	Section 5.2.2 on page 131
Compatibility operators	For imagesetters and roll-fed media devices	Section 9.2 on page 214
	Changes to 2011 for 2012	
Page device parameters	**DeviceRenderingInfo**	Table 4.17 on page 105
	FaxOptions (removed from this *Supplement*)	Table 4.17 on page 105
	ProcessColorModel	Table 4.17 on page 105
	SeparationColorNames	Table 4.17 on page 105
	SeparationOrder	Table 4.17 on page 105
User parameters	**AccurateScreens**	Table 10.1 on page 244
Dictionaries	Type 6 halftone	Section 6.3.3 on page 168
System parameters	**CurStoredFontCache**	Table 10.2 on page 248
	CurStoredScreenCache	Table 10.2 on page 248
	MaxHWRenderingBuffer	Table 10.2 on page 248
	MaxImageBuffer	Table 10.2 on page 248
	MaxStoredFontCache	Table 10.2 on page 248

		Location in this Supplement
	MaxStoredScreenCache	Table 10.2 on page 248
Device parameters	%Calendar%	Table 10.30 on page 327
	/Communications **Interpreter** /AutoSelect /EpsonFX850 /LaserJetIII	Table 10.3 on page 269
	%disk% **Bus** **Interleave**	Table 10.31 on page 330
	%Engine%	Table 10.28 on page 322
	%LaserJetIII%	Table C.2 on page 349
	%LocalTalk% **Filtering**	Table 10.13 on page 297
	%Scsi%	Table 10.26 on page 320
	%os%	Table 10.34 on page 339
	%Parallel% **Protocol** /TBCP **Handshake** **OutputDevice**	Table 10.5 on page 279
	%ScsiComm%	Table 10.6 on page 281
	%Serial% **FlowControl** /XonXoff2 **Protocol** /TBCP	Table 10.4 on page 274
	/FileSystem **BlockSize**	Table 10.31 on page 330 Table 10.32 on page 334 Table 10.33 on page 337
	HWOptions	Page 37
	HalftoneType, type 6	Table 3.7 on page 43
Compatibility operators	**processcolors**	Section 9.2 on page 225

CHAPTER 3

Additions and Modifications to the Language

This chapter supplements Chapter 3 in the *PostScript Language Reference Manual, Second Edition.* It provides information about new and modified resource categories, implicit resources, early binding operations, filters, and **Function** dictionaries.

Overview of This Chapter

3.1 Named Resources 35

3.1.1 Regular Resource Categories 35

Regular resources (Table 3.1) 35

CIDFont 35

CMap 36

ControlLanguage 36

Entries in an instance of the ControlLanguage resource category (Table 3.2) 36

FontSet 36

HWOptions 37

Instances of the HWOptions resource category (Table 3.3) 37

IdiomSet 37

InkParams 37

Localization 37

OutputDevice 38

Entries in an instance of the OutputDevice resource category (Table 3.4) 39

PDL 40

Entries in an instance of the PDL resource category (Table 3.5) 41

ProcSet 41

BitmapFontInit 41

CIDInit 41

ColorRendering 42

FontSetInit 42

Trapping 42

Entries in a Trapping ProcSet dictionary (Table 3.6) 42

TrapParams 42

3.1.2 Implicit Resource Categories 42

Resources whose instances are implicit (Table 3.7) 43

3.2 Early Name Binding 45
3.2.1 bind Operator 45

3.3 Filtered File Details 47
3.3.1 Optional Encode Filters 47
3.3.2 CloseSource and CloseTarget 47
Entries that may appear in any Filter dictionary (Table 3.8) 47
3.3.3 CCITTFaxDecode Filter 48
3.3.4 SubFileDecode Filter 48
3.3.5 LZWEncode and LZWDecode Filters 48
New entries in an LZWDecode dictionary (Table 3.9) 48
3.3.6 FlateEncode and FlateDecode Filters 50
FlateEncode Filter 50
Optional entries in a FlateEncode dictionary (Table 3.10) 51
FlateDecode Filter 51
Comparison of LZW and Flate Encoding 51
Predictor Functions 51
Predictor-related entries in Flate and LZW encode and decode dictionaries (Table 3.11) 52
3.3.7 ReusableStreamDecode Filter 53
Entries in a ReusableStreamDecode dictionary (Table 3.12) 55

3.4 Functions 56
3.4.1 Function Dictionaries 57
Entries common to all Function dictionaries (Table 3.13) 57
Sampled Functions (FunctionType 0) 57
Entries specific to a FunctionType 0 Function dictionary (sampled functions) (Table 3.14) 58
Mapping with the Decode array (Figure 3.1) 60
Exponential Interpolation Functions (FunctionType 2) 61
Entries specific to a FunctionType 2 Function dictionary (exponential interpolation) (Table 3.15) 61
1-Input Stitching Functions (FunctionType 3) 61
Entries specific to a FunctionType 3 Function dictionary (1-input stitching) (Table 3.16) 61

3.1 Named Resources

3.1.1 Regular Resource Categories

Regular resources are resources whose instances are ordinary, useful objects, such as font or halftone dictionaries. Table 3.1 lists regular resource categories that have been added since the publication of the *PostScript Language Reference Manual, Second Edition.* Brief descriptions of these regular resources are provided in the subsequent sections.

Table 3.1 *Regular resources*

	Category name	*Object type*	*Description*
	CIDFont	dictionary	Font dictionary
	CMap	dictionary	Font dictionary
	ControlLanguage	dictionary	Control language dictionary
	**FontSet**	dictionary	Font dictionary
	HWOptions	dictionary	Hardware options dictionary
	**IdiomSet**	dictionary	Procedure-substitution dictionaries
	**InkParams**	dictionary	Ink parameter dictionary
	Localization	dictionary	Natural language support
	OutputDevice	dictionary	Page device capabilities
	PDL	dictionary	PDL dictionary
	ProcSet	dictionary	Procedure set
	**TrapParams**	dictionary	Trapping parameter dictionaries

CIDFont

Instances of the **CIDFont** resource category are dictionaries that represent sets of glyph procedures. The glyph procedures may be defined as Type 1 **CharStrings**, PostScript **BuildGlyph** procedures, or TrueType glyph procedures. An integer character identifier (CID) is used to access glyph outlines in CID fonts.

For further information on the **CIDFont** resource category, see Section 5.3.1, “CID Font Resources,” on page 136.

CMap

Instances of the **CMap** resource category are dictionaries that define mappings from character codes (single or multiple byte) to CIDs, glyph names or character codes, and a font number.

For further information on the **CMap** resource category, see Section 5.3.4, “CMap Resources,” on page 148 and the *Adobe Developer Support Technical Note #5014, Adobe CMAP and CID Font Files Specification, Version 1.0.*

ControlLanguage

The **ControlLanguage** resource category has been added to enable an application or printer driver to determine which control languages are available on a given PostScript product. A control language specifies how the environment and parameter sets and values are configured and determines how jobs are identified. The control language also determines the format of printer-generated messages on the back channel.

The value of each instance of the **ControlLanguage** resource category is a dictionary that contains key-value pairs describing one of the languages supported on some channel on the PostScript product, as shown in Table 3.2.

Table 3.2 *Entries in an instance of the ControlLanguage resource category*

Key	*Type*	*Value*
Selector	name	(*Required*) Used to select a particular PJL or PSPrinter. This same name is used as the name of any parameter set associated with the interpreter.
LanguageFamily	string	(*Required*) Specifies a family of language levels and versions.
LanguageLevel	string	(*Optional)* Specifies the level of the language family. This can be an identifier, such as 2 (Level 2) or PJL. This key may be omitted, or the associated string may be empty.
LanguageVersion	string	(*Optional)* Describes the version for which the above level of the language family has been implemented. This value is typically a microcode version identifier, such as 2015.101. It can also be a product name, such as LaserJet 4si. This key may be omitted, or the associated string may be empty.

The value of the **Selector** key in an instance of the **ControlLanguage** resource category can be used as the value of the **PrinterControl** key in a parameter set of **Type** /Communications. The value of the **PrinterControl** key is set using the **setdevparams** operator.

FontSet



See the section “FontSet Resource” on page 133 for a discussion of the **FontSet** resource category.

HWOptions

The **HWOptions** resource category has been added to enumerate the special hardware options that are present on a product. The value of each instance of the **HWOptions** resource category is a string. Special hardware options do not have any PostScript language facility, other than this resource category, for indicating that they are present. For example, suppose a given product has the ability to support the Adobe PixelBurst® coprocessor. If the coprocessor is not installed on the product, the **HWOptions** resource category would not list **PixelBurst**. If the coprocessor is installed on the product, **PixelBurst** will appear in the **HWOptions** resource category. This resource category is optional.

Table 3.3 lists some possible instances of the **HWOptions** resource category.

Table 3.3 *Instances of the HWOptions resource category*

Key	*Type*	*Instance*
Clock	string	(TODClock)
PixelBurst	string	Version information
ColorBurst	string	Version information
Type1Coprocessor	string	Version information

IdiomSet



For further information, see Section 3.2.1, “bind Operator,” on page 45 and Chapter 8, “Additions and Modifications to the Operators,” on page 193.

InkParams



The **InkParams** resource category contains ink parameter dictionaries that define densities for process inks and predefined spot inks. See Section 4.5, “In-RIP Trapping,” on page 99.

Localization

The **Localization** resource category has been added to enable an application or printer driver to determine which natural languages (for example, English, Japanese, or German) are supported by a product. An instance of the **Localization** resource category is a dictionary that has at a minimum the keys **Language**, **Country**, and **CharSet**—that is, each dictionary has the **Localization** keys in the %Console% parameter set. See Table 10.29 on page 324. Each dictionary represents a legal combination of the values for those

keys. In general, only a sparse subset of the set of all possible combinations will be supported on any given product. For example, the entire category may consist of the following combinations:

<</Language /EN /Country /US /CharSet /ISO-646-ISV>>
<</Language /JA /Country null /CharSet /JIS-...>>

Here /EN is the code for English, /JA is the code for Japanese, /US is the code for the United States, and the *null* value is used to indicate that no dialect is identified for Japanese. For a complete list of values for **Language**, **Country**, and **CharSet**, see Table 10.29 on page 324.

Each instance in the **Localization** resource category has a unique name. Although no naming scheme is enforced, we recommend that you use the following guidelines for naming the instances that make up the **Localization** resource category.

Instances can be named by constructing a composite name for the instance, based on concatenating the three names in the dictionary, separated by hyphens. Thus, the two above dictionaries would be named as follows:

/EN-US-ISO-646-ISV
/JA--JIS...

This approach provides a name that is indicative of the information in the dictionary.

An alternative is to provide descriptive names for the dictionaries. In this case, the above dictionaries might be named as follows:

/AmericanEnglish
/Japanese

Either naming scheme provides a simple way to change the language. Using the second naming scheme, for example, the following PostScript code changes the language to Japanese:

```
%set the console to the Japanese localization
(%Console%) /Japanese /Localization findresource setdevparams
```

Note *If the **SystemParamsPassword** system parameter is set, you will also need to put the **Password** key in the dictionary or run this code as a system administration unencapsulated job. See Section 10.4.1, "Unencapsulated Jobs," on page 258 for details.*

OutputDevice

The **OutputDevice** resource category has been added to perform the following tasks:

- Enable applications to query product capabilities directly.
- Maintain functional equivalence with LanguageLevel 1 (where page size capability information was given by keys such as **letter, legal,** and **a4** in the **userdict** dictionary).

The resource category **OutputDevice** contains one instance for each **OutputDevice** value that the **setpagedevice** operator can accept for that product. The name of each instance should be a valid value for the page device parameter **OutputDevice**. Products that do not contain the **OutputDevice** page device key—that is, products that support only one possible output device—have a single instance for the **OutputDevice** resource category. This single instance may be named **Default** or any product-specific name.

Each instance of the **OutputDevice** resource category is a dictionary that contains key-value pairs that describe certain capabilities of that particular output device, such as the possible page sizes or the possible resolutions. This dictionary does not represent the current state of the PostScript product; it simply provides a static list of some of the capabilities of the product. Over time, Adobe is likely to define new entries in this dictionary to reflect added capabilities. Table 3.4 lists instances that may be required.

Table 3.4 *Entries in an instance of the OutputDevice resource category*

Key	*Type*	*Semantics*
DeviceN 	array	(*Optional*) Specifies colorants of a **DeviceN** color model supported by a device (when producing composite outputs). It is an array of arrays in which each subarray identifies the colorants of a **DeviceN** instance. A subarray can contain names or strings—for example, [/Cyan /Pink] or [(Cyan) (Pink)] or [/Cyan (Pink)]. The size of a subarray need not correspond to the number of toner stations of the device when printing. Because the **DeviceN** color model has no implied colorants, all colorants must be named explicitly. This entry is not required for devices that offer only standard process color model support, where the model implies colorants.
HWResolution	array	(*Optional*) Defines values that can be supported by the product. Each element within the array can be either an array of two numbers indicating a discrete **HWResolution** support or an array of four numbers [*x1 y1 x2 y2*] indicating that the range of hardware resolutions between [*x1 y1*] and [*x2 y2*] is supported. Redundant values may be present.

Table 3.4 *Entries in an instance of the OutputDevice resource category (continued)*

Key	*Type*	*Semantics*
ManualSize	array	(*Optional*) Defines page sizes that can be fed manually for the product. Each element can be either an array of two numbers indicating a discrete page size supported or an array of four numbers [*x1 y1 x2 y2*] indicating that the range of page sizes between [*x1 y1*] and [*x2 y2*] is supported. Redundant values may be present. If a product does not support the **ManualFeed** page device parameter, the array of page sizes should have no entries.
MediaClass	array	(*Optional*) An array of strings or names. Enumerates the allowed values of media class.
PageSize	array	(*Required*) Defines page sizes that can be fed automatically for the product, assuming appropriate media are installed. Each element can be either an array of two numbers indicating a discrete page size supported or an array of four numbers [*x1 y1 x2 y2*] indicating that the range of page sizes between [*x1 y1*] and [*x2 y2*] is supported. Redundant values may be present. This entry is required.
ProcessColorModel	array	(*Optional*) Defines names or strings that indicate the possible color models that can be chosen on the product for composite output. An element in the array can be /DeviceGray, /DeviceRGB, /DeviceCMYK, /DeviceCMY, /DeviceRGBK, or /DeviceN. This entry is required if the device supports more than one value for the **ProcessColorModel** page device parameter.
TrappingDetailsType 	array	(*Optional*) Defines the allowed values of the **Type** key in the **TrappingDetails** dictionary. This entry is not required when trapping is not supported. The **TrappingDetails** dictionary is described in Table 4.20 on page 119.

PDL

The **PDL** resource category has been added to allow an application or printer driver to determine which page description language (PDL) interpreters are available on a given PostScript product. Each category contains an instance for each language selector available on the PostScript product.

The value of each instance of the **PDL** resource category is a dictionary that contains key-value pairs describing one of the languages supported on some channel on the PostScript product. Adobe may add new entries to these dictionaries to further identify the language. Table 3.5 lists entries that are typically present

.

Table 3.5 *Entries in an instance of the PDL resource category*

Key	*Type*	*Value*
Selector	name	(*Required*) Used to select a particular value of the parameter **Interpreter** in a parameter set of **Type** /Communications. This same name is used as the name of any parameter set associated with the interpreter. See the section "ControlLanguage" on page 36.
LanguageFamily	string	(*Required*) Specifies a family of language levels and versions.
LanguageLevel	string	(*Optional)* Specifies the level of the language family. This can be a level identifier, such as 2 (LanguageLevel 2) or 5e (PCL 5e), or it can be a product name if no independent level naming system exists. This key may be omitted or the associated string may be empty.
LanguageVersion	string	(*Optional)* Describes the version for which the above level of the language family has been implemented. This value is typically a microcode version identifier, such as 2015.101. It could also be a product name. This key may be omitted or the associated string may be empty.

The value of a **Selector** key in a **PDL** category instance corresponds to one of the legal values of the **Interpreter** key in a parameter set of **Type** /Communications. The value of the **Interpreter** key is set using the **setdevparams** operator.

ProcSet

Several instances of the **ProcSet** resource category have been added: **BitmapFontInit**, **CIDInit**, **ColorRendering**, **FontSetInit**, and **Trapping**. For further information on **ProcSet**, see Section 3.9.2, in the *PostScript Language Reference Manual, Second Edition*.

BitmapFontInit

The **BitmapFontInit** instance of the **ProcSet** resource category contains operators for incremental downloading and management of glyph bitmaps in a Type 32 font. For further information, see Section 5.2.1, "Bitmap Fonts: Font Type 32," on page 129.

CIDInit

The **CIDInit** instance of the **ProcSet** resource category contains operators for building **CIDFont** and **CMap** resource instances. For further information, see Section 5.3.5, "CIDInit ProcSet," on page 149.

ColorRendering

The **ColorRendering** instance of the **ProcSet** resource category contains operators used in selecting a CRD. For further information, see Section 6.2.1, “CRD Selection Based on Rendering Intent,” on page 159.

FontSetInit

The **FontSetInit** instance of the **ProcSet** resource category contains operators for building **FontSet** resource instances. For further information, see Section 5.2.3, “CFF and Chameleon Fonts: FontType 2 and FontType 14,” on page 132.

Trapping



Trapping is a new dictionary in the **ProcSet** resource category that defines the operators used for in-RIP trapping. Table 3.6 lists the entries in the **Trapping ProcSet** dictionary. For further information on these three operators, see Chapter 8, “Additions and Modifications to the Operators.”

Table 3.6 *Entries in a Trapping ProcSet dictionary*

Operator	*Semantics*
settrapparams	Sets or changes the trapping parameter set.
currenttrapparams	Retrieves the trapping parameter set.
settrapzone	Sets a trapping zone.

TrapParams



The **TrapParams** resource category contains trapping parameter dictionaries that define trapping parameter sets for various press conditions. Each instance of the resource is formatted as a trapping parameter dictionary for the **settrapparams** operator.

3.1.2 Implicit Resource Categories

Implicit resources are resources whose instances are not objects but which represent some built-in capability of the PostScript interpreter. Table 3.7 lists the implicit resources.

Table 3.7 *Resources whose instances are implicit*

Category name	*Instances*	
ColorSpaceFamily	**CIEBasedA**	**DeviceGray**
	CIEBasedABC	**DeviceN**
	CIEBasedDEF	**DeviceRGB**
	CIEBasedDEFG	**Indexed**
	DeviceCMY	**Pattern**
	DeviceCMYK	**Separation**
Emulator[a]	**LaserJetIII**	**LaserJetIIP**
	HP7475A	**Diablo630**
	EpsonFX850	**ProprinterXL**
	PCL	
Filter	**ASCII85Encode**[b]	**ASCII85Decode**
	ASCIIHexEncode[b]	**ASCIIHexDecode**
	CCITTFaxEncode[b]	**CCITTFaxDecode**
	DCTEncode[b]	**DCTDecode**
	FlateEncode[b]	**FlateDecode**
	LZWEncode[b]	**LZWDecode**
	RunLengthEncode[b]	**RunLengthDecode**
	NullEncode	
		GIFDecode[c]
		PNGDecode[c]
		ReusableStreamDecode
		SubFileDecode
FunctionType	0, 2, 3	
IODevice	%Serial%	%Serial_NV%
		%Serial_Pending%
	%Parallel%	%Parallel_NV%
		%Parallel_Pending%
	%ScsiComm%	%ScsiComm_NV%
		%ScsiComm_Pending%
	%LocalTalk%	%LocalTalk_NV%
		%LocalTalk_Pending%
	%EtherTalk%	%EtherTalk_NV%
		%EtherTalk_Pending%
	%TokenTalk%	%TokenTalk_NV%
		%TokenTalk_Pending%
	%LPR%	%LPR_NV%
		%LPR_Pending%
	%AppSocket%	%AppSocket_NV%
		%AppSocket_Pending%
	%Telnet%	%Telnet_NV%
		%Telnet_Pending%
	%RemotePrinter%	%RemotePrinter_NV%
		%RemotePrinter_Pending%
	%PrintServer%	%PrintServer_NV%
		%PrintServer_Pending%

Table 3.7 *Resources whose instances are implicit (continued)*

Category name	*Instances*
	%LAT% %SNMP% %SysLog% %TCP% %UDP% %IP% %SPX% %IPX% %EthernetPhysical% %TokenRingPhysical% %disk0%…%diskn% %cartridge% %rom% %ram% %os% %Engine% %Console% %Scsi% %Ide% %Calendar% %PCL% %LaserJetIII% %LaserJetIIP% %Diablo630% %HP7475A%
ColorRenderingType	1
FMapType	2, 3, 4, 5, 6, 7, 8, 9
FontType	0, 1, 2, 3, 4, 5, 6, 9, 10, 11, 14, 32, 42
FormType	1
HalftoneType	1, 2, 3, 4, 5, 6, 9, 10, 16, 100
ImageType	1, 2, 3, 4
PatternType	1, 2
ShadingType	1, 2, 3, 4, 5, 6, 7

a. The **Emulator** resource category has been displaced by the **PDL** resource and may not be present. The emulator device parameter sets are described in Appendix C, “Emulators,” on page 347.
b. Optional filters. To ensure portability, PostScript language programs that are page descriptions should not invoke these filters.
c. Optional filter. Used only as part of the implementation of web printing. PostScript language programs should not invoke this filter.

3.2 Early Name Binding

In the PostScript language, early binding of names is done with the **bind** operator and the immediately evaluated name, as described in Section 3.11 in the *PostScript Language Reference Manual, Second Edition*, and Chapter 8, “Additions and Modifications to the Operators.” With the introduction of idiom recognition, the description of the **bind** operator has been modified, as described below.

3.2.1 bind Operator



As a final step in the execution of the **bind** operator, the value of the user parameter **IdiomRecognition** is tested. If the value is *true*, then the bound procedure is compared to all of the template procedures in all the dictionaries that are instances in the resource category **IdiomSet**. (The cost of this comparison is small because it is done efficiently.) If a match is found, then the **bind** operator replaces the procedure with the substitute procedure associated with the matching template.

Instances of the **IdiomSet** resource category are procedure substitution dictionaries. The keys of the procedure substitution dictionary are arbitrary names; their values are arrays. The first element of the array is the template procedure, which defines an old or inefficient PostScript procedure. This template procedure is used for matching by the **bind** operator. The second element of the array is the substitute procedure, which is the more efficient PostScript code. Each template procedure represents one idiom. Each key in this dictionary has only one template procedure and one substitute procedure.

Two arrays or procedures are considered comparable if each element that is itself not an array is equal (as returned by the **eq** operator) and each element that is an array is in turn comparable. Nested arrays or procedures are compared only to a level of ten. Replacement is suppressed if the proposed replacement procedure is in local VM when the candidate procedure is in global VM.

If a substitution may have an undesirable effect, substitution may be prevented by setting the value of the user parameter **IdiomRecognition** to *false* before executing the **bind** operator. For example, the value of **IdiomRecognition** should be set to *false* while you are creating new **IdiomSet** resource instance dictionaries.

The following example demonstrates how to construct an instance of the **IdiomSet** resource category:

```
% Temporarily turn off idiom recognition so ‘bind’ does not
% change our template
currentuserparams /IdiomRecognition get
<</IdiomRecognition false>> setuserparams

% Define an IdiomSet resource instance named AdobeWinDriver
% containing a single substitution
/AdobeWinDriver
<< /snap % name of this particular idiom (any name)
     [ % The template procedure
        {  transform 0.25 sub round 0.25 add exch
                     0.25 sub round 0.25 add exch itransform
        } bind


        % The substitute procedure
        { } % Nothing — assure ‘setstrokeadjust’ is “on”
     ]
>> /IdiomSet defineresource pop

% Return idiom recognition to its previous state so ‘bind’
% will replace occurrences of the template procedure with
% the substitute procedure
<</IdiomRecognition 3 -1 roll>> setuserparams
```

Idiom recognition should be disabled during the construction of these instances so that the template or replacement procedures are not recognized as idioms.

As shown in the example, the template procedure should be bound explicitly. The comparison that occurs during execution of the **bind** operator occurs after the candidate procedure is bound. Thus, a bound template is usually what is desired. No binding occurs on either the substitute or template procedures during execution of the **bind** operator on the candidate procedure. Generally, the substitute procedure should be bound at resource instantiation, unless the effects of late binding on the substitute code are what is desired during execution.

Instances of the **IdiomSet** resource category reside in VM, either local or global, and if local, are subject to the **save** and **restore** operators.

Multiple instances of the resource category may contain identical template procedures, but only one will be in effect when idiom recognition is enabled. The instance that takes precedence is not defined.

An idiom set can be deleted or replaced using an explicit **undefineresource** operation.

3.3 Filtered File Details

For information on filtered files, see Section 3.13 in the *PostScript Language Reference Manual, Second Edition*.

All filters accept an optional parameter dictionary. This is a LanguageLevel 2 feature that was added after the *PostScript Language Reference Manual, Second Edition,* was published.

3.3.1 Optional Encode Filters



In Adobe PostScript 3 software beginning with version 3010, all encode filters, with the exception of the **NullEncode** filter, become optional—that is, they may or may not be present in a PostScript interpreter product. Use the **resourceforall** or **resourcestatus** operator to examine the list of filters present to determine whether a particular encode filter is included in a given product. Filters available in a given product are contained in the implicit resource category **Filter**.

3.3.2 CloseSource and CloseTarget



The **CloseSource** and **CloseTarget** parameters are optional Boolean parameters in a decode or encode filter's dictionary. **CloseSource** is used with decode filters, and **CloseTarget** is used with encode filters. These parameters govern the disposition of the filter's data source or target when the **closefile** operator is applied to the filter explicitly or implicitly, by the **restore** operator, garbage collection, or reaching EOD. If the value of **CloseSource** or **CloseTarget** is *false* (the default), no additional action is taken on the data source or target; this is the behavior of LanguageLevel 2 products. However, if the value is *true*, after the **closefile** operator has been applied to the filter, it is also applied to the filter's data source or target (iteratively, as necessary).

Table 3.8 *Entries that may appear in any Filter dictionary*

Key	*Type*	*Semantics*
CloseSource	boolean	(*Optional*) If the value is *true*, closes the input stream when the data source is closed. If the value is *false*, no additional action is taken on the data source. Default value: *false*
CloseTarget	boolean	(*Optional*) If the value is *true*, closes the output stream when the data target is closed. If the value is *false*, no additional action is taken on the data target. Default value: *false*

3.3.3 CCITTFaxDecode Filter

The implementation-defined limit to the **Columns** entry in the **CCITTFaxDecode** dictionary has been increased from 25,000 to 62,000.

3.3.4 SubFileDecode Filter



The **SubFileDecode** filter can be invoked with the following syntax:

source count string /SubFileDecode filter

This syntax has been extended to also allow:

source <</EODCount *count* /EODString *string* >> /SubFileDecode filter

This second format is the only format that can be used with the **ReusableStreamDecode** filter.

3.3.5 LZWEncode and LZWDecode Filters

Changes to the LanguageLevel 2 specification of the **LZWEncode** and **LZWDecode** filters are documented in Appendix D, “Updates to the PostScript Language Reference Manual, Second Edition,” on page 355 of this *Supplement*.

Table 3.9 lists the new entries in the **LZWDecode** dictionary associated with LanguageLevel 3.

Table 3.9 *New entries in an LZWDecode dictionary*

Key	*Type*	*Semantics*
UnitSize 	integer	(*Optional*) Specifies the size of the units encoded by LZW. Legal values are 3 to 8, inclusive. Default value: 8
LowBitFirst 	boolean	(*Optional*) Specifies the endianness of the encoded byte stream. If the value is *true*, the encoded data are treated as little-endian. If the value is *false*, the encoded data are treated as big-endian. Default value: *false*

Both the **UnitSize** and **LowBitFirst** entries in the **LZWDecode** dictionary refer to the form of the encoded data. Data encoded in the data stream are unsigned integers in the range 0 to ($2^{UnitSize} - 1$); that is, the encoded data are unsigned integers of **UnitSize** bits. The two codes `EOD` and `CLEAR` are defined in terms of **UnitSize** as follows:

$\texttt{CLEAR} = 2^{UnitSize}$

$$EOD = 2^{UnitSize} + 1$$

Code lengths can be from (**UnitSize** + 1) to a maximum of 12 bits. The **LZWDecode** filter always returns 8-bit values, even if the value of the **UnitSize** key is less than 8. If the value of **UnitSize** is less than 8, values are returned in the least significant bits of the byte.

LZW is a bit stream protocol, and the units of compression do not necessarily fall on byte boundaries. How these compression codes get packed into a byte stream is not specified, and some extant data streams do not pack the data in a way that is consistent with LanguageLevel 2 LZW filters. With LanguageLevel 2 filters, the bytes can be thought of as flowing through a shift register, from low order (least significant) to high order (most significant), and the appropriate number of bits is masked off the high-order end of the unused bit stream. Alternatively, new bytes can be added to the high-order (most significant) side of the shift register, and codes can be masked off from the low-order side. The Boolean **LowBitFirst** key controls this decision: if the value is *false*, operation is consistent with LanguageLevel 2 LZW filters; if the value is *true*, new bytes will be shifted into the most significant place in the shift register.

These concepts are relatively simple mathematically and yet difficult to describe. The following examples may help. Although the **UnitSize** and **LowBitFirst** entries are implemented only for the decode filter, it is easier to discuss them in terms of encoding. Extending the example on page 132 of the *PostScript Language Reference Manual, Second Edition*, let us encode the input sequence of decimal values:

```
45 45 45 45 45 65 45 45 45 66 …
```

Input sequence	*Output code*		*Code added to table*		*Sequence represented by new code*
	UnitSize				
	7	8	7	8	
—	128[a]	256[a]			
45	45	45	130	258	45 45
45 45	130	258	131	259	45 45 45
45 45	130	258	132	259	45 45 65
65	65	65	133	260	65 45
45 45 45	132	259	134	261	45 45 45 66

a. Clear code

If the value of **UnitSize** is 7, then the codes in the above example are 8 bit and would pack into bytes (in hexadecimal) as follows:

```
80 2D 82 82 41 84 …
```

In this example, because the codes are 8 bits long, the value of **LowBitFirst** is irrelevant. However, the value of **LowBitFirst** will affect encodings when the code size is extended to 9 bits.

If the value of **UnitSize** is 8, then the 9-bit codes would be packed differently depending on the value of **LowBitFirst**. If the value of **LowBitFirst** is *false*, then the encoding would be as follows:

```
80 0B 60 50 22 0C 0 …
```

If the value of **LowBitFirst** is *true*, then the encoding would be as follows:

```
00 5B 08 14 18 64 8 …
```

The **LZWEncode** filter will accept these new entries in the parameter dictionary only if **UnitSize** is 8 and **LowBitFirst** is *false*. Other values will result in a **rangecheck** error at filter creation time.

See also the section “Predictor Functions” on page 51.

3.3.6 FlateEncode and FlateDecode Filters

FlateEncode Filter

The syntax for invoking the **FlateEncode** filter is as follows:

target dictionary /FlateEncode filter

The **FlateEncode** filter encodes binary or ASCII data, optionally after pretransformation by a predictor function. See also the section “Predictor Functions” on page 51. Encoding is based on the public-domain zlib/deflate compression method, which is a variable-length Lempel-Ziv adaptive compression method cascaded with adaptive Huffman coding. The zlib/deflate compression method is fully defined in Internet Engineering Task Force Requests for Comments (IETF RFCs) 1950 and 1951. See *DEFLATE Compressed Data Format Specification,* listed in Appendix G, “Bibliography,” on page 395. The optional predictor functions are discussed in the section “Predictor Functions” on page 51.

The output produced by the **FlateEncode** filter is always binary, even if the input is ASCII text

.

Table 3.10 *Optional entries in a FlateEncode dictionary*

Key	*Type*	*Semantics*
Effort	integer	(*Optional*) Controls the memory used for Flate compression and the execution speed. Legal values range from –1 to 9. A value of 0 compresses rapidly but not tightly, using little auxiliary memory. A value of 9 compresses slowly but as tightly as possible, using plenty of auxiliary memory. Default value: –1 (The default value is mapped to a value within the range 0 to 9 that is a “reasonable” default for the implementation.)

FlateDecode Filter



The syntax for invoking the **FlateDecode** filter is as follows:

source dictionary /FlateDecode filter

The **FlateDecode** filter decodes data encoded in zlib/deflate compressed format. The zlib/deflate format has an EOD. See the description of the **FlateEncode** filter for details of the compressed format.

Comparison of LZW and Flate Encoding

Flate encoding, like LZW encoding, discovers and exploits many patterns in its input data, whether text or images. Thanks to its cascaded adaptive Huffman coding, Flate-encoded output is usually substantially more compact than LZW-encoded output for the same input. Flate and LZW decoding speeds are comparable, but the Flate encoding speed is considerably slower than LZW encoding. Usually, the **FlateEncode** and **LZWEncode** filters compress their inputs substantially. In the worst case, however, the **FlateEncode** filter expands its input by no more than a factor of 1.003, plus the effects of algorithm tags added by PNG predictors and the effects of any explicit **flushfile** operations. LZW compression has a worst-case expansion of at least a factor of 1.125, which can increase to a factor of nearly 1.5 in some implementations (plus PNG tags effects, as with the **FlateEncode** filter).

Predictor Functions

LZWEncode and **FlateEncode** filters compress more compactly if their input data are highly predictable. One way of increasing the predictability of many continuous-tone sampled images is to replace each pixel with the difference between that pixel and some predictor function applied to earlier neighboring pixels. If the predictor function works well, the postprediction data will cluster toward zero.

Two predictor function groups are supported. The first, the TIFF group, consists of the single function that is Predictor 2 in the TIFF standard. (In TIFF 6.0, Predictor 2 applies only to LZW compression, but here it applies to Flate compression also.) TIFF Predictor 2 predicts that each color component of a pixel will be the same as the corresponding color component of the pixel immediately to the left.

The second supported predictor function group, the PNG group, consists of the "filters" of the W3C PNG recommendation. The term "predictors" is used here instead of "filters" to avoid confusion. There are five basic PNG predictor algorithms, and a sixth one that invites an optimum hybrid of the first five. The first five are "None," "Sub" (predicts the same as the pixel to the left), "Up" (predicts the same as the pixel above), "Average" (predicts the average of the pixel to the left and the pixel above), and "Paeth" (a nonlinear function of the pixel above, the pixel to the left, and the pixel to the upper left).

Both predictor function groups have some commonalities. Both assume that data are presented in order, from the top row to the bottom row, and within a row, from left to right. Both assume that a row occupies a whole number of bytes, rounded upward if necessary. Both assume that pixels and their components are packed into bytes from high- to low-order bits. Both assume that all color components of pixels outside the image (which are necessary for predictions near the boundaries) are zero.

Table 3.11 *Predictor-related entries in Flate and LZW encode and decode dictionaries*

Key	*Type*	*Semantics*
Predictor	integer	(*Optional*) Selects predictor function: 1 — No prediction, default value 2 — TIFF Predictor 2 10 — PNG prediction (on encode, PNG None on all rows) 11 — PNG prediction (on encode, PNG Sub on all rows) 12 — PNG prediction (on encode, PNG Up on all rows) 13 — PNG prediction (on encode, PNG Average on all rows) 14 — PNG prediction (on encode, PNG Paeth on all rows) 15 — PNG prediction (on encode, PNG optimum)
Columns	integer	(*Optional*) The number of samples in a sampled row. Has an effect only if **Predictor** > 1. Default value: 1
Colors	integer	(*Optional*) The number of interleaved color components in a sample. Has an effect only if **Predictor** > 1. Default value: 1

Table 3.11 *Predictor-related entries in Flate and LZW encode and decode dictionaries (continued)*

Key	*Type*	*Semantics*
BitsPerComponent	integer	(*Optional*) The number of bits used to represent each color component. Must be 1, 2, 4, 8, or 16. Default value: 8
	Note	*Images with 16 bits per component may not be used directly as input to the* ***image*** *operator.*

The two predictor function groups also differ in significant ways. First, the postprediction data for each PNG-predicted row begins with an explicit algorithm tag, so different rows can be predicted with different algorithms to improve compression. TIFF 2 prediction has no such identifier; the same algorithm applies to all rows. Second, the TIFF function group predicts each color component from the prior instance of that color component, without regard to the width of the color component or the number of colors. In contrast, the PNG function group predicts each byte from the corresponding byte of the prior pixel (and/or the same pixel on the prior line and/or the prior pixel on the prior line), regardless of whether there are multiple color components in a byte, or whether a single color component spans multiple bytes. This can yield significantly better speed but with somewhat worse compression.

3.3.7 ReusableStreamDecode Filter



It is sometimes necessary to store blocks of data that may be used as image data for a form, or for function or mesh data used by a smooth shading operator. Such data can be stored in a string, but strings are limited to 64 kilobytes.

A reusable stream is a simple way of storing blocks of data longer than 64 kilobytes in a printer. The printer may store the data internally in VM or on a disk. The amount of data that can be stored is limited only by the amount of storage available in the printer.

A reusable stream is a file object derived from a **ReusableStreamDecode** filter. The **ReusableStreamDecode** filter provides a way of creating a data block that is randomly accessible via a repositionable file object and that can exceed the 64K limit associated with string objects. By definition, when this file object hits EOF, it is not automatically closed, and it can be repositioned. The creation of this file object uses a **ReusableStreamDecode** dictionary and an extension of the **filter** operator. The data source may be in-line data delimited with a **SubFileDecode** filter, or an external file object.

The **ReusableStreamDecode** filter takes an optional dictionary argument on the stack. The **filter** operator is invoked as follows:

source *dict* /ReusableStreamDecode **filter** *fileobj*

This use of the **filter** operator has some unusual side effects, and the resulting *fileobj* has some unusual attributes. When **filter** is executed, the data from *source* may or may not be immediately buffered in VM or written to disk, depending on a variety of factors, including the following:

- The nature of *source*; whether it is **currentfile**, disk file, string, procedure, or filtered file
- The availability of system disk storage
- The availability of VM, constrained by the %ram% **LogicalSize** parameter (Table 10.33 on page 337)
- The set of **Filters** specified in *dict*
- Implementation and system memory management details

If *source* is derived from **currentfile** or from a PostScript procedure, the data will always be read at the time the **filter** operator is executed. **currentfile** data should typically be filtered through a **SubFileDecode** filter. If *source* is a string or if *source* is derived from a disk-based file, that string or file should thereafter be dealt with as read only; writing into such a string or file will have unpredictable consequences for the data read from *fileobj*. However, such strings may be undefined, although they will not be garbage collected until they are no longer needed. Also, in many file systems, such files may be deleted, although their disk space will not be freed until it is no longer needed.

Unlike other filtered files, the filtered file object resulting from the **ReusableStreamDecode** filter is repositionable. It has a *length* value. When it reaches EOF or EOD, the file object is not closed automatically, and the file pointer can be repositioned. The following **file** operators may act on the filtered file object:

- **closefile** Closes the file object. The file object is also closed when it is destroyed by the **restore** operator or garbage collection. Any associated temporary file created on a file system will be deleted when the file object is closed.
- **bytesavailable** Returns file object size minus current file position. If file position is at EOF, 0 is returned.

- **flushfile** Sets file object position to EOF.
- **resetfile** Equivalent to 0 **setfileposition**.
- **setfileposition** Sets file object position with value in the range 0 to *length*.
- **fileposition** Returns current file object position. The result is always in the range 0 to *length*. If file object position is at EOF, *length* is returned.

Table 3.12 gives the entries in the **ReusableStreamDecode** dictionary.

Table 3.12 *Entries in a ReusableStreamDecode dictionary*

Key	*Type*	*Semantics*
Filter	name or array	(*Optional*) Filters to be applied prior to delivering data to the reader. The value of the **Filter** key can be either the name of a single decode filter or an array of filter names. Specify multiple filters in the order they should be applied to decode the data. For example, data compressed using LZW and then ASCII85 encode filters can be decoded by providing the following value: [/ASCII85Decode /LZWDecode]
DecodeParms	dictionary or array	(*Optional*) Parameters used by the decode filters specified with the **Filter** key. They must have a one-to-one relationship with the **Filter** parameters. (If **Filter** is a name, then **DecodeParms** must be a dictionary. If **Filter** is an array, then **DecodeParms** must be an array of dictionaries.) Each parameter in the array can be only a dictionary object or a *null* object. If there is only one parameter—a dictionary, it does not have to be listed in an array. Note that the **SubFileDecode** filter takes a mandatory dictionary object.
Intent	integer	(*Optional*) "Hints" at the intended purpose of the reusable data. If the value is recognized, it may help optimize the data storage or caching strategy. If the value is absent or not a recognized value, it gets a default value of zero (0). Currently recognized values are: 0 Image data 1 Image mask data 2 Sequentially accessed look-up table data (for example, threshold arrays) 3 Randomly accessed look-up table data (for example, functions, CSAs, CRDs) Default value: 0
AsyncRead	boolean	(*Optional*) If the value is *false*, file position of *<source>* is advanced to EOF or EOD. This key affects only the disk files. Default value: *false*

Table 3.12 *Entries in a ReusableStreamDecode dictionary* *(continued)*

Key	*Type*	*Semantics*
CloseSource	boolean	(*Optional*) If the value is *true*, closes the input stream when the reusable stream is closed. If the value is *false*, no additional action is taken on the data source. Default value: *false*

How and when filters are applied to the source data and how caching is applied are implementation dependent. An implementation might typically choose to apply filters at data-acquisition time, with initial filters that cause the size of the data to decrease—for example, **ASCII85Decode** or **ASCIIHexDecode**. Filters that cause the data size to increase (for example, decompression filters) might be left until data-read time. However, the amount of system resources available, the caching strategy, and whether the data is being randomly or sequentially accessed might reasonably affect such choices. Image data will typically be accessed in "slices" of constant size. Look-up table data may be accessed sequentially or randomly, depending on the application.

3.4 Functions



Each class of a **Function** dictionary has a **FunctionType** value that specifies the representation of the function, the set of attributes which parameterize that representation, and the additional data needed by that representation. **FunctionType** is an implicit resource category, as defined in Table 3.7.

Functions may be thought of as "*m*-in, *n*-out" numerical transformations. Each **Function** dictionary implicitly declares the sizes of *m* and *n*, and explicitly declares a domain of input values for which the function is defined and a range outside of which no result value will fall. Domain and range intervals must be bounded and rectangular in the input or output space of the function. They are assumed to be "closed" in the mathematical sense; that is, the edges of the interval are included in the interval. The function must be defined (but not necessarily continuous or smooth) across its entire domain. If any input in the declared domain of the function would cause the function to output a value outside the declared range, then that output value will be clipped to the declared range. If a function is called with input values outside the declared domain, the inputs will be clipped to the declared domain.

Each client of function objects must specify how it uses the function and how it maps the client's domain into the function's domain. If the output of the function is modified by the client before use, that must also be specified by each client. Clients of functions must note that the function's declared domain may be smaller than the actual domain of the function, and the

declared range may be larger than the actual range of the function. Because of this, it is usually necessary to selectively specify the function so that its domain and range are appropriate for the client's use.

3.4.1 Function Dictionaries



All **Function** dictionaries share the attributes given in Table 3.13.

Table 3.13 *Entries common to all Function dictionaries*

Key	*Type*	*Semantics*
FunctionType	integer	(*Required)* Must be one of the defined function type values.
Domain	array	(*Required)* An array of numbers, interpreted in pairs. Each pair of numbers declares the domain of one input variable. The smaller bound must precede the larger bound in the pair. The size of the **Domain** array implicitly declares the input dimensionality *m* of an *m*-input, *n*-output (*m*-in, *n*-out) function, because *m* is one half the number of elements in **Domain**. Inputs outside of the declared domain will be clipped to the nearest boundary value.
Range	array	(*Optional for some **FunctionTypes**)* An array of numbers, interpreted in pairs. Each pair of numbers declares the range of one output value. The smaller bound must precede the larger bound in the pair. The size of the **Range** array implicitly declares the output dimensionality *n* of an *m*-input, *n*-output (*m*-in, *n*-out) function, as *n* is one half the number of elements in **Range**. Output of the function will be clipped to the declared range. If the range is undeclared, no clipping will be done.

In addition, each type of **Function** dictionary must include attributes appropriate to the **FunctionType** key. The output dimensionality of a function can usually be deduced from other attributes of the function; if not, the **Range** attribute is required. The dimensionality of the function inferred from the **Domain** and **Range** declarations must be consistent with the dimensionality inferred from other attributes of the function.

Sampled Functions (FunctionType 0)



Function dictionaries with a **FunctionType** value of 0 use a sequence of sample values to provide an approximation for functions whose domains and ranges are bounded. The samples are organized in a table or array. The dimensionality of the sample table is equal to the dimensionality of the input domain. Samples may have more than one component. The number of components in each sample is equal to the dimensionality of the output range.

Sampled functions are highly general and offer reasonably accurate representations of arbitrary analytic functions at low expense. For example, a single-input sinusoidal function can be represented over the range [0 180] with an average error of only 1%, using just ten samples and linear

interpolation. Two-input functions will require significantly more samples, but usually not a prohibitive number, as long as the function does not have high frequency variations.

The dimensionality of a sampled function is restricted only by implementation limits. However, the number of samples required to represent high-dimensionality functions multiplies very rapidly unless the sampling resolution is very low. Also, the process of multilinear interpolation becomes computationally intensive if the input dimensionality is greater than two. (The multidimensional spline interpolation is even more computationally intensive.)

Because functions are presumed to be reusable, the internal representation of a sampled function must fit entirely within system memory, otherwise a **VMerror** will occur. This limit is dependent on the amount of system memory available.

In addition to the entries listed in Table 3.13, a **Function** dictionary with a **FunctionType** value of 0 has the entries given in Table 3.14.

Table 3.14 *Entries specific to a FunctionType 0 Function dictionary (sampled functions)*

Key	*Type*	*Semantics*
FunctionType	integer	(*Required*) Must be 0. Specifies a sampled function.
Order	integer	(*Optional*) Specifies the order of interpolation between samples. Allowed values are 1 or 3, specifying linear or cubic spline interpolation, respectively. Default value: 1
DataSource	various	(*Required)* May be a string or a reusable stream. (A *reusable stream* is a file object derived from a **ReusableStreamDecode** filter.) Provides the sequence of sample values that specifies the function. If the amount of sampled data is greater than 64 kilobytes, a reusable stream must be used.
BitsPerSample	integer	(*Required*) Specifies the number of bits used to represent each sample value. Limited to 1, 2, 4, 8, 12, 16, 24, or 32.
Encode	array	(*Optional*) Specifies the linear mapping of input values into the domain of the function's sample table. Default value: [0 ($Size_0 - 1$) …]
Decode	array	(*Optional*) Specifies the linear mapping of sample values into the range of values appropriate for the function's output variables, as with **Image** dictionaries. Default value: Same as for **Range**
Size	array	(*Required*) The number of samples in each input dimension of the sample table.
Domain	array	(*Required*) As in Table 3.13.
Range	array	(*Optional*) See **Range** in Table 3.13 on page 57.

The **Domain**, **Encode**, and **Size** attributes determine how the function's input variable values will be mapped into the sample table. For example, if the **Domain** is [–1 1 –1 1] and the **Size** is [21 31], the default **Encode** is [0 20 0 30], which maps the entire **Domain** into the full set of sample table entries. Other values of **Encode** may be used. In general, for the i^{th} input variable d_i, the corresponding encoded value e_i, is

$$e_i = (d_i - D_{2i}) \cdot \frac{(E_{2i+1} - E_{2i})}{(D_{2i+1} - D_{2i})} + E_{2i}$$

where D_i and E_i are elements of the **Domain** and **Encode** arrays, respectively. If a resultant encoded value e_i falls outside the domain [0, $Size_n$ – 1], the value is clipped to the nearest allowed value. The encoded input values are real numbers, not restricted to integers, and multivariable interpolation is used to determine an output value from the surrounding nearest-match sample table values.

Similarly, the **Range**, **Decode**, and **BitsPerSample** attributes determine how the function's sample values are mapped into output values. This is essentially identical to the way image sample values are decoded. The value of **BitsPerSample** implies that all sample values must be in the range [0 ($2^{BitsPerSample}$ – 1)]. This range will be linearly transformed by the **Decode** array to an output range. The default **Decode** array is equal to the **Range** array, indicating a mapping of the entire possible sample range into the entire possible output range. However, other values of **Decode** may be used, as will be seen in the example at the end of this section. In general, for the i^{th} sample component s_i, the corresponding output value r_i, is

$$r_i = s_i \times \frac{(D_{2i+1} - D_{2i})}{(2^{BitPerSample} - 1)} + D_{2i}$$

where D_i are elements of the **Decode** array.

Samples are encoded and interpreted exactly as are image samples, except that function sample data for a new row must continue to be packed with the previous row and need not necessarily start on a byte boundary. No row padding is done with sampled function data. As with image data, a sequence of samples is considered to represent an array in which the first dimension of the array varies fastest; that is, in a two-dimensional array of data, the *x* component varies fastest and the *y* component next fastest.

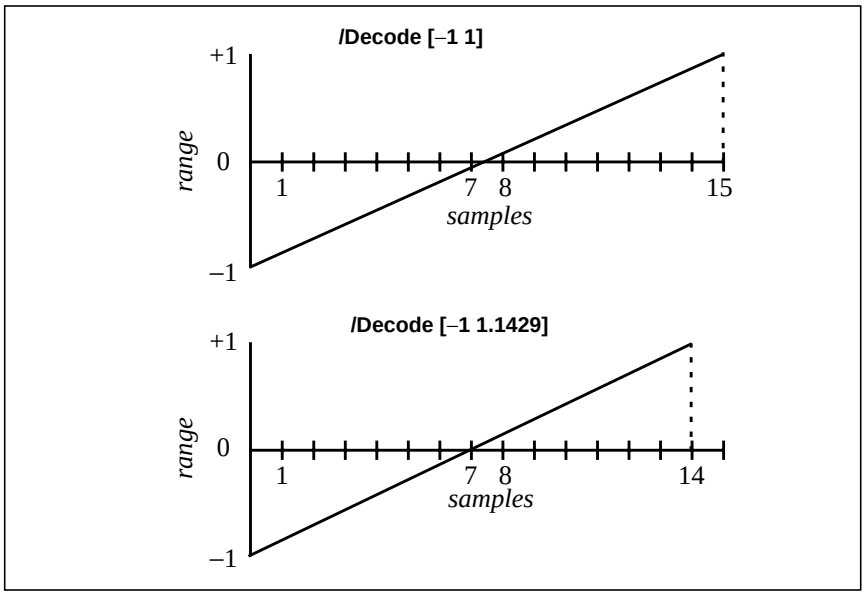
As an example, consider a sampled function with 4-bit samples in an array containing 21 columns and 31 rows, and consider using this function to represent a halftone spot function. A spot function takes two arguments, *x* and *y*, in the domain [–1 1], and returns one value, *z*, in that same range. In the **Function** dictionary, the value of **Domain** would be [–1 1 –1 1], the value of **Size** would be [21 31], and the value of **Encode** would be [0 20 0 30]. The value of **BitsPerSample** would be 4, the value of **Range** would be [–1 1], and the value of **Decode** would be [–1 1]. The *x* argument

would be linearly transformed by the encoding to the domain [0 20] and the *y* argument to the domain [0 30]. Using bilinear interpolation between sample points, the function computes a value for *z*, which will be in the range [0 15], and the decoding transforms *z* to a number in the range [–1 1] for the result. The sample array is stored in a stream of 326 bytes = [31 rows × 21 samples/row × 4 bits/sample ÷ 8 bits/byte]. The first byte contains the sample for the point (–1, –1) in the high-order 4 bits and the sample for the point (–0.9, –1) in the low-order 4 bits.

The key **Encode** gives the linear mapping between the keys **Decode** and **Size**. The default value of **Encode** is [0 ($s_0 - 1$) 0 ($s_1 - 1$)], where s_i is the i^{th} value in the **Size** array. To specify a nondefault encoding, the beginning and ending points for the encoding must be contained between 0 and ($s_i - 1$).

The key **Decode** may be used creatively to increase the accuracy of encoded samples corresponding to certain values in the range. For example, if the desired range of the function is [–1 1] and the value of **BitsPerSample** is 4, the usual value of **Decode** would be [–1 1], and the sample values would be integers in the interval [0 15]. But if these values are used, the midpoint of the range of the function (0) would not be represented exactly by any sample value, since it would fall halfway between 7 and 8. Instead, one could use a **Decode** array of [–1 +1.1428571] and sample values in the interval [0 14]. The desired effective range of [–1 1] would be achieved, and the range value 0 would be precisely represented by the sample value 7 (Figure 3.1)

Figure 3.1 *Mapping with the Decode array*



The value of the **Size** of an input dimension can be 1, in which case all input values in that dimension will be mapped to the single allowed value. If the **Size** of an input dimension is less than 4, then cubic spline interpolation is not possible, and if **Order** 3 is specified, it will be ignored.

Exponential Interpolation Functions (FunctionType 2)



Function dictionaries of this type include a set of parameters that define an exponential **interpolation** in one variable (a 1-input function).

Table 3.15 *Entries specific to a FunctionType 2 Function dictionary (exponential interpolation)*

Key	*Type*	*Semantics*
FunctionType	integer	(*Required)* Must be 2. Specifies an exponential interpolation function.
C0	array	(*Optional)* Defines the function result for input value 0. **C0** must be the same size as **C1**, and the function is *n*-out, where *n* is the size of the array. Default value: [0]
C1	array	(*Optional)* Defines the function result for input value 1. **C1** must be the same size as **C0.** Default value: [1]
N	number	(*Required*) Defines the interpolation exponent. Each input value *t* will return the value given by $C_0 + t^N \times (C_1 - C_0)$.

Values of **Domain** must constrain *t* such that, if **N** is not an integer, all values of *t* must be greater than or equal to zero, and if **N** is negative, no value of *t* may be zero.

For typical use as an interpolation function, the value of **Domain** will be declared as [0 1], and the value of **N** will be a number greater than 0. The **Range** key may be used to clip the output to a desired range.

1-Input Stitching Functions (FunctionType 3)



Function dictionaries of this type define a “stitching” of the subdomains of several 1-input functions to produce a single new 1-input function.

Table 3.16 *Entries specific to a FunctionType 3 Function dictionary (1-input stitching)*

Key	*Type*	*Semantics*
FunctionType	integer	(*Required*) Must be 3. Specifies a 1-input stitching function.
Domain	array	(*Required*) As in Table 3.13.
Range	array	(*Optional*) As in Table 3.13.
Functions	array	(*Required)* Array of 1-input functions making up the stitched function. Output dimensionality of all functions must be compatible with the value of **Range**. The array is an array of dictionaries.

Table 3.16 *Entries specific to a FunctionType 3 Function dictionary (1-input stitching) (continued)*

Key	*Type*	*Semantics*
Bounds	array	(*Required)* Size of the **Bounds** array must be one less than the size of the **Functions** array. **Bounds** elements must be in order of increasing magnitude, and each value must be within the value of **Domain**. The entry **Bounds**, in combination with the entry **Domain**, defines the intervals for which each function from the **Functions** array is used to determine the value of the stitched function. Each interval is mapped through the **Encode** array into the domain of the corresponding function. The array must be an array of numbers.
Encode	array	(*Required)* Size of the **Encode** array must be twice the size of the **Functions** array. A pair of **Encode** array values is associated with each function. The **Encode** values map each subset of the domain defined by **Domain** and the **Bounds** array to the domain of the corresponding function. The array must be an array of numbers.

The stitching function is designed to make it easy to combine several functions to be used within one shading, over different parts of the shading's domain. The same effect can be achieved by creating separate shading dictionaries for each of the functions, which have adjacent domains. However, since each shading would have similar parameters, and because the overall effect is one shading, it is more convenient to have a single shading with a multiple function definition. Also, **FunctionType** 3 provides a general mechanism to invert the domains of 1-input functions.

Note *For further information, see Section 4.4, "Smooth Shading," on page 76.*

An input *d* to the stitching function in the subdomain $B_{2i-1} \le d \le B_{2i}$ will be encoded as follows:

$$e = (d - B_{2i-1}) \times \frac{E_{2i+1} - E_{2i}}{B_{2i} - B_{2i-1}} + E_{2i}$$

where B_i and E_i are elements of the **Bounds** array and **Encode** array, respectively, and the resulting value *e* is routed as input to the i^{th} function in the **Function** array. This is similar to the **Encode** definition in the sampled function description. For these purposes, B_{-1} is considered to be the first element of the **Domain** array, and B_n (where *n* is the number of subdomains) is considered to be the second element of the **Domain** array. The subdomain mappings may be inverted by allowing E_{2i+1} to be less than E_{2i}.

CHAPTER 4

Additions and Modifications to the Graphics Model

This chapter supplements Chapter 4 in the *PostScript Language Reference Manual, Second Edition.* It provides information about color space names, HiFi and multitone color, masked images, smooth shading, in-RIP trapping, patterns, new entries in the page device dictionary, input media selection, and new entries in the **Policies** dictionary.

Overview of This Chapter

4.1 CIE-Based Color Spaces 65

4.1.1 Two New Color Spaces: CIEBasedDEF and CIEBasedDEFG 65

Original CIEBasedABC structure (Figure 4.1) 66

CIEBasedDEF(G) pre-extension to the CIEBasedABC color spaces (Figure 4.2) 66

Entries in a CIEBasedDEFG color space dictionary (Table 4.1) 67

4.2 HiFi and Multitone Color 68

A Note About Terminology 69

A Note About Color Specification 69

4.2.1 Indexed Color Space 69

4.2.2 DeviceN Color Space 70

4.3 Masked Images 71

4.3.1 Image Dictionaries 72

Entries in a type 3 image dictionary (Table 4.2) 72

Entries in a MaskDict dictionary (Table 4.3) 73

Entries in a DataDict dictionary (Table 4.4) 74

Entries in a type 4 image dictionary (Table 4.5) 75

4.4 Smooth Shading 76

4.4.1 Pattern Dictionaries 77

Entries in a type 2 pattern dictionary (Table 4.6) 77

4.4.2 Shading Dictionaries 77

Entries common in all Shading dictionaries (Table 4.7) 78

ColorSpace: Special Considerations 79

Shading Dictionary Entries Associated with Each Defined ShadingType 80

ShadingType 1: Function-Based Shading 80

Entries in a ShadingType 1 Shading dictionary (function-based shadings) (Table 4.8) 81
ShadingType 2: Axial Shading 81
Entries in a ShadingType 2 Shading dictionary (axial shadings) (Table 4.9) 81
ShadingType 3: Radial Shading 83
Entries in a ShadingType 3 Shading dictionary (radial shadings) (Table 4.10) 83
ShadingType 4: Free-Form Gouraud-Shaded Triangle Meshes 85
Entries in a ShadingType 4 Shading dictionary (free-form Gouraud-shaded triangle meshes) (Table 4.11) 85
Drawing a new triangle (f = 0): Free-form Gouraud-shaded triangle meshes (Figure 4.3) 87
How the value of the edge flag determines which edge is used for the next triangle (Figure 4.4) 87
Varying the value of the edge flag to create different shapes (Figure 4.5) 88
ShadingType 5: Lattice-Form Gouraud-Shaded Triangle Meshes 89
Sample lattice forms (Figure 4.6) 89
Entries in a ShadingType 5 Shading dictionary (lattice-form Gouraud-shaded triangle meshes) (Table 4.12) 89
ShadingType 6: Coons Patch Meshes 90
Coons patch meshes: Coordinate mapping from a unit square to a four-sided patch (Figure 4.7) 91
Coons patch meshes: Appearance, painted area, and boundary (Figure 4.8) 92
Entries in a ShadingType 6 Shading dictionary (Coons patch meshes) (Table 4.13) 93
Color values and edge flags in Coons patch meshes (Figure 4.9) 94
Coons patch meshes: Coordinates for adjacent patches (Table 4.14) 95
Coons patch meshes: How the value of the edge flag determines the edge for the next patch (Figure 4.10) 96
ShadingType 7: Tensor Product Patch Meshes 96
4.4.3 Painting with a Pattern Dictionary 98

4.5 In-RIP Trapping 99
An example of trapping (Figure 4.11) 99
4.5.1 Setting Trapping Zones for In-RIP Trapping 100
4.5.2 Trapping Parameters 100
4.5.3 Trapping Parameter Dictionary 101
Entries in a trapping parameter dictionary (Table 4.15) 101
4.5.4 ColorantZoneDetails Dictionary 103
Entries in a colorant dictionary (Table 4.16) 103

4.6 Device Setup 104
4.6.1 Page Device Parameters 104
Page Device Dictionary 104
Entries in the page device dictionary (Table 4.17) 105
4.6.2 Details Dictionaries 116
Type 117
Typical entries in a CollateDetails dictionary (Table 4.18) 118

Typical entries in a PostRenderingEnhanceDetails dictionary (Table 4.19) 118
4.6.3 TrappingDetails and ColorantDetails Dictionaries 119
Entries in a TrappingDetails dictionary (Table 4.20) 119
Entries in a ColorantDetails dictionary (Table 4.21) 119
Entries in a device colorant name dictionary (Table 4.22) 120
4.6.4 DeviceRenderingInfo Dictionaries 120
Type 121
Typical entries in a DeviceRenderingInfo dictionary (Table 4.23) 121
4.6.5 Errors Generated by Page Device Parameters 121
typecheck Errors 121
rangecheck Errors 122
undefined Errors 122
invalidaccess Errors 123
limitcheck Errors 123
4.6.6 Input Media Selection: Envelope Orientation in User Space 123
Default user space orientation for an envelope with a PageSize value of portrait (Figure 4.12) 124
4.6.7 New Entries in the Policies Dictionary 124
Entries in a Policies dictionary (Table 4.24) 124

4.1 CIE-Based Color Spaces

Whereas Adobe PostScript software, version 2015 and earlier, supported only device-dependent CMYK colors, Adobe PostScript software now supports calibrated CMYK colors as well. This new feature is an extension to the CIE-based color spaces described in Section 4.8.3 in the *PostScript Language Reference Manual, Second Edition*.

4.1.1 Two New Color Spaces: CIEBasedDEF and CIEBasedDEFG

Two CIE-based names have been added for use with the **setcolorspace** operator. They are **CIEBasedDEF** and **CIEBasedDEFG.**

CIEBasedDEF and **CIEBasedDEFG** are three- and four-component color spaces that extend the Adobe CIE-based color spaces to include additional color spaces—in particular, calibrated CMYK color spaces. The additional color spaces supported as of Adobe PostScript software, version 2016, include the following:

- CIE 1976 L*u*v and calibrated RGB from scanners, both of which are three-component color spaces
- Calibrated CMYK, which is a four-component color space

CIEBasedDEF and **CIEBasedDEFG** are simple pre-extensions to the **CIEBasedABC** color space. Figure 4.1 shows how the **CIEBasedABC** color space looked prior to the addition of the **CIEBasedDEF** and **CIEBasedDEFG** pre-extensions.

Figure 4.1 *Original CIEBasedABC structure*

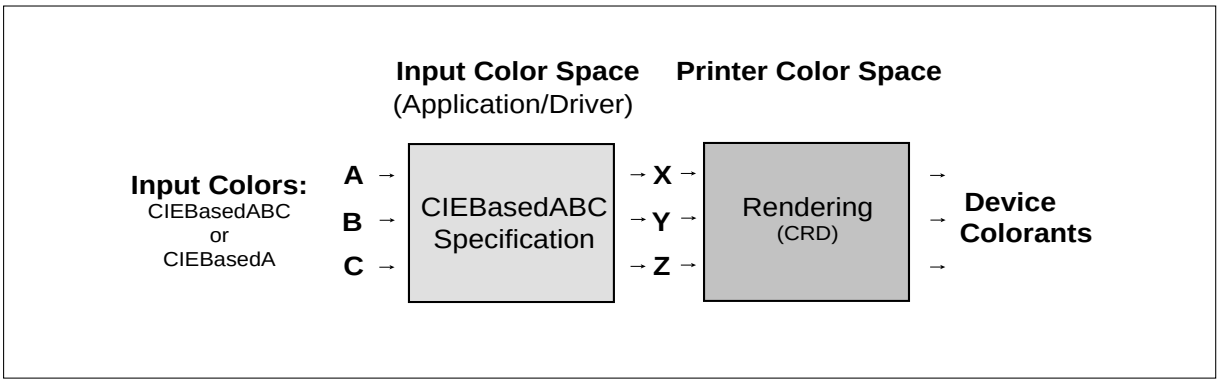


Figure 4.2 shows how the new pre-extension fits with the pre-existing **CIEBasedABC** color spaces.

Figure 4.2 *CIEBasedDEF(G) pre-extension to the CIEBasedABC color spaces*

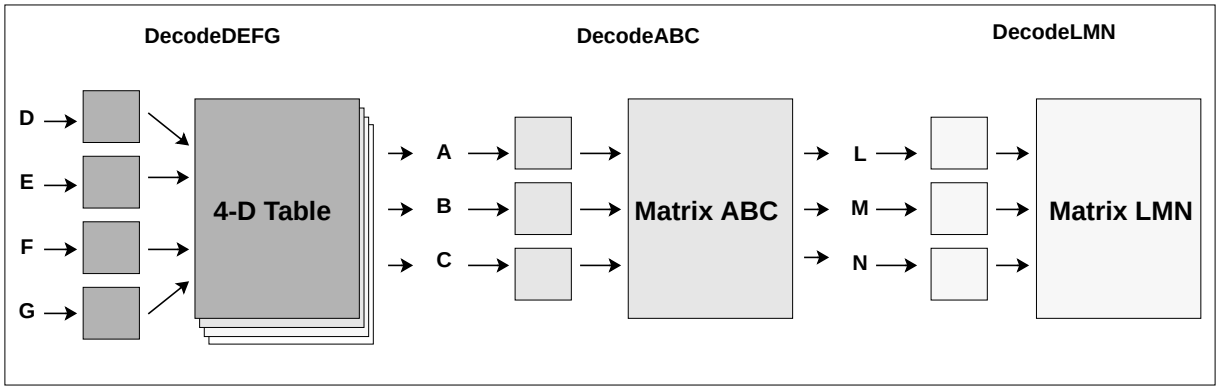


Figure 4.2 represents a four-component color space transformation. A three-component color space transformation would be structured identically, except that a 3-input/3-output table would function where the 4-input/3-output (four-dimensional) table appears in Figure 4.2, and the procedure **DecodeDEF** would be applied instead of **DecodeDEFG**.

As Figure 4.2 shows, the components of the input colors (*D*, *E*, *F*, *G*) are first transformed component-wise in the color space pre-extension (**CIEBasedDEFG** specification). This transformation is performed by applying the PostScript procedure **DecodeDEFG**, and the values that procedure yields are used to look up and interpolate in the four-dimensional table. It is worth noting that a four-dimensional table is mechanistically styled after the multidimensional table of the rendering table of the CRD. The processes generated by the **DecodeDEFG** procedure yield the (*A*, *B*, *C*)

values. Afterwards, the (*A*, *B*, *C*) values are processed as **CIEBasedABC** color space values by the CSA and passed to the CRD in the normal way, as illustrated in Figure 4.1 above and as described in Section 4.8.3 in the *PostScript Language Reference Manual, Second Edition*.

Table 4.1 gives the entries in a **CIEBasedDEFG** color space dictionary. The dictionary for the **CIEBasedDEF** color space is the same as below except that the inputs are three components. That is, the **RangeDEF** and **RangeHIJ** keys have six array values that give the ranges for three components; the **DecodeDEF** key is a three-value array; and finally, the **Table** key, which is a look-up table, is a 3-input/3-output mapping table with corresponding adjustments in its parameters.

After execution of the **setcolorspace** operator, the initial values of *D*, *E*, *F*, and *G* are zero (0), unless the range of valid values for a color component does not include 0, in which case the nearest valid value is substituted.

Table 4.1 *Entries in a CIEBasedDEFG color space dictionary*

Key	*Type*	*Semantics*
RangeDEFG	array	(*Optional*) Array of eight numbers [D_0 D_1 E_0 E_1 F_0 F_1 G_0 G_1] that specify the range of valid values for the *D*, *E*, *F*, and *G* components of the color space—that is, $D_0 <= D <= D_1$, $E_0 <= E <= E_1$, and so forth. Default value: [0 1 0 1 0 1 0 1]
DecodeDEFG	array	(*Optional*) Array of four PostScript procedures [D_D D_E D_F D_G] that decode the *D*, *E*, *F*, and *G* components of the color space into values *H*, *I*, *J*, and *K*, respectively, which are more suitable for performing a table look-up. Default value: Array of identity procedures [{} {} {} {}]
RangeHIJK	array	(*Optional*) Array of eight numbers [H_0 H_1 I_0 I_1 J_0 J_1 K_0 K_1] that specify the range of the valid values for the *H*, *I*, *J*, and *K* components of the color space—that is, $H_0 <= H <= H_1$, $I_0 <= I <= I_1$, and so forth. Default value: [0 1 0 1 0 1 0 1]
Table	array	(*Required*) Array of the form [N_H N_I N_J N_K *table*] that describes a four-dimensional look-up table that maps colors in the four-dimensional color space with coordinates *H*, *I*, *J*, and *K* into a three-dimensional color space with coordinates *A*, *B*, and *C* via table look-up and interpolation. The *ABC* space subsequently maps into the CIE 1931 (XYZ) space using the same process and guided by the same dictionary entries as in the **CIEBasedABC** color space. Those dictionary entries are in this dictionary. The table contains $N_H \times N_I \times N_J \times N_K$ entries, each of which consists of three values making up an *ABC* color value. N_H, N_I, N_J, and N_K must be integers greater than 1. The entry in the table at coordinates (*h*, *i*, *j*, *k*), where $0 <= h < N_H$, $0 <= i < N_I$, and so forth, contains the color value in the *ABC* space that corresponds to the value in the *HIJK* space, where:

Table 4.1 *Entries in a CIEBasedDEFG color space dictionary (continued)*

Key	*Type*	*Semantics*
		$H = H_0 + h[(H_1 - H_0) / (N_H - 1)]$ $I = I_0 + i[(I_1 - I_0) / (N_I - 1)]$ $J = J_0 + j[(J_1 - J_0) / (N_J - 1)]$ $K = K_0 + k[(K_1 - K_0) / (N_K - 1)]$ and where H_0, H_1, I_0, and I_1, and so forth are given in the **RangeHIJK** entry. The element *table* must be an array of N_H arrays. Each of those arrays must contain an array of N_1 strings. Each of those strings must contain $3 \times N_J \times N_K$ characters. The h^{th} array value of the mapping table contains an array whose i^{th} value is the string for which the three characters starting at position $3 \times (j \times N_1 + k)$ constitute the table entry location (h, i, j, k). These three characters are interpreted as color values in the *ABC* color space, each in the range 0 to 255.
Other entries	entry-specific	All of the entries required for a **CIEBasedABC** color space dictionary are also required entries in this type of dictionary. See the **CIEBasedABC** color space dictionary specification for their details. All of the entries specified as optional for a **CIEBasedABC** color space dictionary are also optional entries in this type of dictionary. Again, see the **CIEBasedABC** color space dictionary specification for their details.

4.2 HiFi and Multitone Color



This section describes extensions to the PostScript language color models for support of general color specification and rendering mechanisms. These extensions address the needs of HiFi and multitone (duotones, tritones, and quadtones) class color systems in the PostScript language.

HiFi color is the use of more than the standard CMYK process colorants in order to produce an extended color gamut. A popular HiFi color system is PANTONE® Hexachrome™, which uses six inks—CMYK plus orange and green. Frequently, HiFi color systems use CMYK process inks with additional process inks to form a HiFi process color model. The CMYK inks used in these cases, however, are often not the traditional CMYK inks.

HiFi color spaces are supported for both color specification and color rendering. HiFi color specification is needed because applications, often with the help of color management systems, transform data into HiFi color spaces prior to printing. HiFi color rendering is required because there are advantages to performing the color transformations in the PostScript interpreter.

Multitone-class color systems represent the use of a single-component image to paint multiple color components. In a duotone, for example, a single-component image can be used to paint both the black component and a spot-

colored component. The tone reproduction is generally different for the different components. For example, the black component might be painted with the exact data from the single-component image, while the spot-colored component might be generated as a nonlinear function of the single-component image data in a manner that emphasizes the shadows.

A Note About Terminology

The term *separation* is often misused. In the context of this discussion, a printing system that produces separations generates *n* pieces of media (generally film). These pieces of media are known as separations.

The *colorants* of a device should not be confused with its separations. A specific device colorant should be referred to as a separation only if the device is generating separations, one of which corresponds to the colorant.

The **Separation** color space is not a separation per se; rather, it specifies a named color component that can mark directly on a device's colorant if the device has the appropriately named colorant. This behavior is independent of whether the device actually generates separations. For example, the **Separation** color space can be used to mark on the cyan colorant of a composite device.

The **SeparationColorNames** entry in the page device dictionary contains the names of the colorants. The **SeparationColorNames** entry need not include colorants implied by the process color model.

A Note About Color Specification



The color space names used with the **setcolorspace** operator are expanded to include the **DeviceN** color space name. The dictionary form of the **image** operator is required for use with the **DeviceN** color space. Both the **setcolor** and **currentcolor** operators are extended to support an arbitrary number of components.

Only the LZW, Flate, and DCT filters consider the number of color components. For LZW, the **Colors** key in the filter parameters dictionary has its upper limit removed. The JPEG standard is defined only for one to four color components of interleaved data. As a consequence, the DCT filters cannot be used for more than four color components.

4.2.1 Indexed Color Space



The **Indexed** color space has been modified to allow the base color space to be either the **Separation** or the **DeviceN** color space. This additional functionality allows the indexed mechanism to be used to apply a tone-scale adjustment to an individual color component. It also allows component data to be used for painting multiple color components.

Note *Using **Separation** or **DeviceN** color spaces as the base color space for an **Indexed** color space will generate an error on LanguageLevel 2 devices.*

4.2.2 DeviceN Color Space



The **DeviceN** color space provides for the composite specification of an arbitrary number of device-dependent color components. This color space provides greater control with respect to overprint between the components. Previously, for example, to paint only the cyan, magenta, and yellow device colorants and leave the black colorant unaltered required using the **Separation** color space once each for cyan, magenta, and yellow; a composite description of this color is not permitted using the **DeviceCMYK** color space without altering the black component. The **DeviceN** color space allows you to create a composite description for the cyan, magenta, and yellow color components, while leaving the black component unaltered.

It is also often desirable to use data for a single component to paint multiple color components. For example, a single gray image can be used to produce a duotone. The **DeviceN** color space, used in conjunction with the modified indexed mechanism, provides this functionality.

Note *Using **DeviceN** color spaces will generate an error on LanguageLevel 2 devices.*

A **DeviceN** color space is specified as follows:

```
[/DeviceN names alternativeSpace tintTransform] setcolorspace
```

Color values in the **DeviceN** color space are tint components in the range 0 to 1. The value 0 represents application of the minimum amount of colorant to the component; the value 1 represents application of the maximum amount of colorant. The **setcolor** operator sets the current color in the graphics state to a set of tint values; the initial value is 1 for each tint. A sampled image can also be treated as a source of tint values.

The number of color components is derived from the length of the *names* array. *names* is an array containing the unique names of the individual color components. If the *names* array contains repeated names of the same colorant, a PostScript error will be raised. Each *name* can be either a name or string object. When **DeviceN** is the current color space, the **setcolor** operator takes a number of arguments equal to the length of the *names* array.

It is decided at the moment the color space is set to **DeviceN**, if all named color components can be made—that is, it is decided if the device has colorants with those names. If all the named color components can be made, the PostScript parameters *alternativeSpace* and *tintTransform* are ignored. Subsequent painting operations paint the named device colorants. If all the named color components cannot be made, then subsequent painting

operations are performed using the *alternativeSpace* color space. The *alternativeSpace* color space enables the colors to be approximated by colors in some device-dependent or device-independent color space.

The *alternativeSpace* parameter must be an array or name object that identifies the alternative color space. This can be any device-dependent or device-independent color space, but not a special color space. The *alternativeSpace* parameter should be constructed as if it were to be used as an operand to the **setcolorspace** operator.

The *tintTransform* procedure is a PostScript procedure that provides an *n*-to-*m* transformation, where *m* is consistent with the number of color components needed by the *alternativeSpace* parameter. During subsequent painting operations, when the named color components are not available on the device, this procedure is used to transform *n* tint values into *m* color component values in the alternative color space.

4.3 Masked Images



In the graphic arts industry and page layout applications, it is common to take an image, “mask” out the background, and then place the cropped image on a different background. With LanguageLevel 2, this is commonly done by drawing a clipping path between the pixels of the source image. However, if the clipping path is very complex, or if there is more than one clipping path, a **limitcheck** error can result when an attempt is made to save or print the clipping path(s).

The masked image feature enables users to place images on the page without concern for the constraints of the clipping path. The implementation efficiently represents masked images by indicating which individual pixels in the image should be blocked out or masked. The definition of the **image** operator has two new image types, **ImageType** 3 and **ImageType** 4.

ImageType 3 combines a regular sampled image with an explicit mask. The mask is treated essentially the same as the source data of the **imagemask** operator. It indicates the places on the page that are painted and those that are “masked” (that is, left unchanged). The unmasked places are painted with the corresponding portions of the sampled image; the masked places are not painted.

The description of a regular image contains *N* color components per pixel (where *N* depends on the current color space); masked images contain *N* color components plus one mask component per pixel. The mask data may be included with the image data, or it may be provided separately. The mask and the image do not need to be the same resolution, but they do need to be in the same place on the page; that is, they must overlay each other.

ImageType 4 masked images are defined by specifying a color or range of colors. Pixels in the image that match these colors are not painted, allowing the background to show through. This technique is often called Chroma-key masking.

The mask data is specified by the **InterleaveType** key in the **ImageType** dictionary. The behavior of the mask is determined by the value of the mask and the entries in the **MaskDict** dictionary. See Table 4.3 on page 73.

4.3.1 Image Dictionaries



The image dictionaries have been expanded to include a type 3 dictionary for masked image processing (Table 4.2) and a type 4 dictionary for Chroma-key image processing (Table 4.5 on page 75). These new dictionaries are used only when the **image** operator is called; they are not permitted with the **imagemask** operator.

Note *Entries in a type 3* ***DataDict*** *dictionary and a type 4 image dictionary are interpreted as if they were entries for a type 1 image dictionary, except as indicated otherwise in the following tables. Also, entries in the* ***MaskDict*** *dictionary are interpreted as if the dictionary were being supplied to the* ***imagemask*** *operator, except as indicated in the following tables. Type 2 image dictionary entries are described in Section 7.1, "Type 2 Image Dictionary," on page 178. Type 1 image dictionary entries are described in the Postscript Language Reference Manual, Second Edition, in Table 4.8 on page 219.*

Table 4.2 *Entries in a type 3 image dictionary*

Key	*Type*	*Semantics*
ImageType	integer	(*Required*) Must be 3 for masked images.
InterleaveType	integer	(*Required*) Specifies how the mask and image samples are supplied. The values are defined as follows: 1 The mask data source is provided with the image data source. The mask data is interleaved on a per-sample basis, mask data first. The number of bits per component of the mask must equal the number of bits per component of the image source. The value of the mask component should either be zero or all ones. Other values will be treated as all ones.

Table 4.2 *Entries in a type 3 image dictionary (continued)*

Key	*Type*	*Semantics*
		2 The mask and image source samples are interleaved at the source scanline boundary. The mask and image sample scans must be padded to byte boundaries separately. The mask samples are one bit per pixel, regardless of the image sample depth. The mask data source lines precede the image data. The mask height may differ from the image height, with the following restrictions: one must be an integral multiple of the other; that is, there may be multiple lines of mask data per line of image data or multiple lines of image data per line of mask data. The smaller of the mask **Height** or image **Height** defines the number of interleave blocks. An *interleave block* consists of either one row of mask data followed by one or more rows of image source data, or one or more rows of mask data followed by one row of image source data. All interleave blocks must have the same number of scan lines of each data type. The relationship between the **Height** entries of the two dictionaries will determine the format. The width is arbitrary. 3 The mask data is provided through a separate data source, as defined in the **MaskDict** dictionary. The height and width of the mask data are independent of the height and width of the image source data, but the mask must have the same orientation and placement as the image.
MaskDict	dictionary	(*Required*) Modified **ImageType** 1 dictionary (Table 4.3).
DataDict	dictionary	(*Required*) Modified **ImageType** 1 dictionary (Table 4.4). The image data in **DataDict** can also be interleaved of separate data sources.

The **MaskDict** and **DataDict** dictionaries are modifications of **ImageType** 1 dictionaries, as described in Table 4.8 in the *Postscript Language Reference Manual, Second Edition,* page 219. The entries in these two dictionaries are described in Table 4.3 and Table 4.4, respectively.

Table 4.3 *Entries in a MaskDict dictionary*

Key	*Type*	*Semantics*
ImageType	integer	(*Required*) Must be 1.
Width	integer	(*Required*) Width of the mask in samples. If the value of **InterleaveType** is 1, **Width** must be equal to the value of **Width** in the **DataDict** dictionary.

Table 4.3 *Entries in a MaskDict dictionary (continued)*

Key	*Type*	*Semantics*
Height	integer	(*Required*) Height of the mask. If the value of **InterleaveType** is 1, **Height** must be equal to the value of **Height** in the **DataDict** dictionary. If the value of **InterleaveType** is 2, then **Height** in **MaskDict** may differ from **Height** in **DataDict**, with the following restrictions: one must be an integral multiple of the other; that is, there may be multiple lines of mask data per line of image data or multiple lines of image data per line of mask data.
ImageMatrix	array	(*Required*) An array of six numbers that defines a transformation from current user space to image source space. This matrix must be approximately equal to the value of **ImageMatrix** in the **DataDict** dictionary, scaled by the difference in size of the mask and the image data.
MultipleDataSources	boolean	(*Optional*) If present, must be *false*.
DataSource	(various)	(*Required if the value of **InterleaveType** is 3; must not be present otherwise*) Specifies the data source for mask data.
BitsPerComponent	integer	(*Required*) Must be 1 unless the value of **InterleaveType** is 1. If the value of **InterleaveType** is 1, it must be equal to the value of **BitsPerComponent** in the **DataDict** dictionary.
Decode	array	(*Required*) An array of two numbers that describe how to map mask sample values into the appropriate range of values. A decoded value of 0 designates a sample to paint. A decoded value of 1 designates a sample that is not to be painted.
Interpolate	boolean	(*Optional*) If present and *true*, requests that interpolation be performed on the mask. Default value: *false*

Table 4.4 *Entries in a DataDict dictionary*

Key	*Type*	*Semantics*
ImageType	integer	(*Required*) Must be 1.
Width	integer	(*Required*) Width of the source image in samples.
Height	integer	(*Required*) Height of the source image in samples.
ImageMatrix	array	(*Required*) An array of six numbers that defines a transformation from current user space to image source space.
MultipleDataSources	boolean	(*Optional*) If the value is *true*, the image data is provided through multiple data sources, one per color component and one for mask data. Also, the value of **InterleaveType** must be 3. Default value: *false*

Table 4.4 *Entries in a DataDict dictionary (continued)*

Key	*Type*	*Semantics*
DataSource	(various)	(*Required*) The values of **DataSource** will provide mask data interleaved with source image data if the value of **InterleaveType** is 1 or 2. The mask data will come before the image data. See the *PostScript Language Reference Manual, Second Edition*, page 220.
BitsPerComponent	integer	(*Required*) Specifies the number of bits used to represent each color component. The number must be 1, 2, 4, 8, or 12. The number of bits is the same for all color components.
Decode	array	(*Required*) An array of numbers. The length of the array must be twice the number of color components in the current color space. This describes how to map image sample values into the range of values appropriate for the current color space.
Interpolate	boolean	(*Optional*) If present and *true*, requests that image interpolation be performed. Default value: *false*

Table 4.5 *Entries in a type 4 image dictionary*

Key	*Type*	*Semantics*
ImageType	integer	(*Required*) Must be 4.
MaskColor	integer array	(*Required*) An array of values corresponding to the source representation of the color value that will be masked. One value is needed for each source color component. The **Indexed** color space and the **DeviceGray** color space will need only one value. Alternatively, a pair of values may be specified for each component. An image sample will be considered to match the mask color if each component lies within each pair of values. Because each component must be tested individually, this option is slower than the single color version described above. Values are checked against the incoming data samples as they are read. No conversion or decoding is done, except for decompressing any compressed data.
Width	integer	(*Required*) Width of the source image in samples.
Height	integer	(*Required*) Height of the source image in samples.
ImageMatrix	array	(*Required*) An array of six numbers that define a transformation from current user space to image source space.

Table 4.5 *Entries in a type 4 image dictionary (continued)*

Key	*Type*	*Semantics*
MultipleDataSources	boolean	(*Optional*) If the value is *true*, the image data is provided through multiple data sources, one per color component. If *false*, the image data for all color components are packed into one data stream, interleaved on a per-sample basis. Default value: *false*
DataSource	(various)	(*Required*) If the value of **MultipleDataSources** is *false* or not present, the value of **DataSource** must be a single data source (file, procedure, or string). If the value of **MultipleDataSources** is *true*, **DataSource** must be an array of data sources. The length of the array must be the same as the number of components in the current color space.
BitsPerComponent	integer	(*Required*) Specifies the number of bits used to represent each color component. The number must be 1, 2, 4, 8, or 12. Only one number may be specified. The number of bits is the same for all color components.
Decode	array	(*Required*) An array of numbers. The length of the array must be twice the number of color components in the current color space. This describes how to map image sample values into the range of values appropriate for the current color space.
Interpolate	boolean	(*Optional*) If present and *true*, requests that image interpolation be performed. Should always be *false* to prevent visual artifacts in the image. Default value: *false*

4.4 Smooth Shading



Just as the **setstrokeadjust** operator provides a resolution-independent method for requesting uniform line thickness, the smooth shading capability provides a way to request smooth transitions between paths and between colors, without specifying the number of steps in the transition.

Smooth shadings of objects and regions are defined in terms of *gradient fills*, where a gradient fill is a complex paint that may be used with the **fill**, **stroke**, **show**, or **imagemask** operator to paint a path or mask, using a smooth transition between colors across the areas painted. A new type of **Pattern** dictionary (**PatternType** 2) is used to define such paints, as described in Section 4.4.1, “Pattern Dictionaries.” Such **Pattern** dictionaries include a **Shading** subdictionary to define the color transition used.

Smooth shading allows the geometry of the area to be filled (the object) to be separate from the geometry of the color gradient fill or transition (the description of the colors to be used to create the gradient fill). When the object to be painted is a relatively simple shape, or when the geometry of the object to be painted with a gradient fill is the same as the geometry of the

gradient fill itself, the **shfill** operator can be used (**shfill** is described in Chapter 8 on page 206.) **shfill** accepts a single operand, which is a **Shading** dictionary. The **Shading** dictionary contains details of the type (method) of shading, the geometry of the area or object to be shaded, and the geometry of the color gradient fill. In addition, the **Shading** dictionary can contain a **Function** dictionary that defines how the colors vary across the area or object to be shaded. The **Function** dictionary is required for some types of shading and optional for others.

The shading methods are defined in the entry for **ShadingType** in Table 4.7 on page 78.

4.4.1 Pattern Dictionaries



Table 4.6 gives the entries in the pattern dictionary used to define the complex paints used for painting gradient fills.

Table 4.6 *Entries in a type 2 pattern dictionary*

Key	*Type*	*Semantics*
PatternType	integer	(*Required*) Must be 2.
Shading	dictionary	(*Required*) Defines the color transition. A **Shading** dictionary consists of the entries in Table 4.7 plus those in one of Tables 4.8 through 4.13.
XUID	array	(*Optional*) An extended unique ID that identifies the pattern. See Section 5.8.2 in the *PostScript Language Reference Manual, Second Edition*. Presence of an **XUID** key in a pattern dictionary enables the PostScript interpreter to save cached instances of the pattern cell for later use, even when the pattern dictionary is loaded into VM multiple times by different jobs, for example. To ensure correct behavior, the values of **XUID** must be assigned from a central registry. This is particularly appropriate for patterns treated as named resources. Patterns that are created dynamically by an application program should not contain **XUID** values.
Implementation	any	Defined by the **makepattern** operator. The type and value of this entry are implementation dependent.

Note ***PatternType*** *2 gradient fills do not tile. To create a tiling pattern containing a gradient fill, use the* ***shfill*** *operator in the* ***PaintProc*** *procedure of a* ***PatternType*** *1 pattern resource.*

4.4.2 Shading Dictionaries



All **Shading** dictionaries contain the entries listed in Table 4.7, plus the entries associated with the specified **ShadingType** value (Table 4.8 through Table 4.13).

Some types of **Shading** dictionaries also include a **Function** dictionary (**ShadingTypes** 1, 2, 3, 6, and 7). In such cases, the **Shading** dictionary usually defines the geometry of the shading, while the **Function** dictionary defines the color transitions across that geometry. Functions with high spatial frequency (or discontinuous) color transitions may display aliasing effects when imaged at low effective resolutions.

Table 4.7 *Entries common in all Shading dictionaries*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Must be a defined **ShadingType** value. The attributes associated with each value of **ShadingType** are defined in Table 4.8 to Table 4.13. The shading types are defined as follows: 1 (*Function-based shadings*) Defines the color of every point in the domain using a mathematical function (not necessarily smooth or continuous). (Table 4.8) 2 (*Axial shading*) Defines a color blend along a line between two points; optionally extended beyond the points by continuing the boundary colors. (Table 4.9) 3 (*Radial shading*) Defines a blend between two circles; commonly used to image three-dimensional spheres and cones. (Table 4.10) 4 (*Free-form Gouraud-shaded triangle meshes*) Defines a common construct used by many three-dimensional applications to image complex colored and shaded shapes. (Vertices are specified in free-form geometry.) (Table 4.11) 5 (*Lattice-form Gouraud-shaded triangle meshes*) Defines a common construct used by many three-dimensional applications to image complex colored and shaded shapes. (Vertices are specified in terms of a pseudo-rectangular lattice.) (Table 4.12) 6 (*Coons patch meshes*) Constructs shadings from one or more color patches, each bounded by four Bézier curves. (Table 4.13) 7 (*Tensor product patch meshes*) Similar to Coons patch meshes, except there are 16 control points in each patch instead of 12. See “ShadingType 7: Tensor Product Patch Meshes” on page 96.
ColorSpace	name or array	(*Required*) May be any device-dependent, device-independent, or special color space specification, except **Pattern**. (**Indexed** color space requires some special considerations, as defined below in the section “ColorSpace: Special Considerations.”) All color values in the shading are interpreted relative to this color space.
Background	array	(*Optional*) An array of color components appropriate to the **ColorSpace** key, specifying a single color value.

Table 4.7 *Entries common in all Shading dictionaries (continued)*

Key	*Type*	*Semantics*
BBox	array	(*Optional*) An array of four numbers, interpreted as lower-left and upper-right coordinates in the current coordinate space at the time the shading is imaged. If present, the shading is clipped to the intersection of this bounding box and the current clipping path. If omitted, the shading is clipped to the bounding box of the clipping region at the time the shading is imaged.
AntiAlias	boolean	(*Optional*) If *true*, the shading function is combined with a convolution function to average shading values across device pixels. This produces a more device-independent representation when the spatial frequency of the shading is more than about half the device resolution. It also makes shadings more resistant to variations in appearance due to changes in the CTM. This computation can have a performance impact. The implementation of **AntiAlias** is device specific, and some devices may ignore this key. Default value: *false*

ColorSpace: Special Considerations



The **ColorSpace** key common to all shading dictionaries defines not only the color space in which color values are specified in the shading, but also the color space in which the gradient fill calculations are performed. The gradient fills between colors defined by most shadings are implemented using a variety of interpolation algorithms, and these algorithms are sensitive to the characteristics of the color space. Linear interpolation, for example, may have observably different results if specified in CMYK color space as opposed to CIE L*a*b* color space, even if the starting and ending colors are perceptually identical. The difference arises because the two color spaces are not linear relative to each other. Shadings are rendered according to the following rules:

- If **ColorSpace** is a device dependent color space and different from the process color space of the device, then color values will be converted to device colors using standard conversion formulas. To maximize performance, these conversions may take place at any time. Thus, any shadings defined with device-dependent color spaces may have color gradient fills that are somewhat device-dependent. (This does not apply to any of the vignette shadings (**ShadingTypes** 2 and 3), because those shadings perform gradient fill calculations on a single variable and then convert to parametric colors.)

- If **ColorSpace** is a device independent color space, then all gradient calculations will occur in the device independent color space. Conversion to device colors will occur only after all interpolation calculations are performed. Thus, the color gradients will be device independent for the colors generated at each point.

- If **ColorSpace** is **Separation** or **DeviceN** and the specified colorant(s) are not supported so that the *alternativeSpace* parameter must be used, the gradient fill calculations will be performed in the special color space prior to conversion to the *alternativeSpace*. Thus, nonlinear *tintTransform* functions will be accommodated for the best possible representation of the shading. (If the specified colorants are supported, then no color conversion calculations are needed.)

- If **ColorSpace** is **Indexed**, then all color values specified in the shading will be immediately converted to the base color space. Depending on whether the base color space is device dependent or device independent, gradient fill calculations will be performed as stated above. Interpolation never occurs in the **Indexed** color space, which is quantized and inappropriate for calculations that assume a continuous range of colors. Also, as described for the available **ShadingType** entries, the **Indexed** color space may not be allowed in some shadings. (For example, the **Indexed** color space is not allowed for vignette shadings that perform interpolation calculations on a single variable and then convert to parametric colors, which are assumed to represent a continuous range of colors. Similarly, the **Indexed** color space is not allowed for function-based shadings, which interpolate between sampled color values.)

Shading Dictionary Entries Associated with Each Defined ShadingType



In addition to the entries listed in Table 4.7, a **Shading** dictionary must have entries specific to the value of the **ShadingType** key. This section defines the available shading types.

ShadingType 1: Function-Based Shading



In **ShadingType** 1 shadings, the color of every point in the domain is defined by a mathematical function that is not necessarily smooth or continuous. In addition to the common entries listed in Table 4.7, a **Shading** dictionary for **ShadingType** 1 includes the following entries:

Table 4.8 *Entries in a ShadingType 1 Shading dictionary (function-based shadings)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 1. Specifies function-based shading.
Domain	array	(*Optional*) Array of four numbers specifying the rectangular domain of arguments with which the color function(s) are called. Default value: [0 1 0 1]
Matrix	array	(*Optional*) A transformation matrix specifying the mapping from the **Domain** value defined in the shading dictionary into the coordinate space in which the shading is being imaged. For example, if the **Domain** value defined in the shading dictionary is [0 1 0 1] and it must be mapped to a one-inch square with lower-left corner at (100, 100) in user space, the **Matrix** value would be [72 0 0 72 100 100]. Default value: Identity matrix.
Function	dictionary or array	(*Required*) May be a single 2-in, *n*-out **Function** dictionary or an array of *n* 2-in, 1-out **Function** dictionaries (where *n* is the number of color components in the **ColorSpace** entry). The **Domain** value defined in the **Function** dictionary (described in Table 3.13 on page 57) must be a superset of the **Domain** value defined in the **Shading** dictionary. If the values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.

Any points within the **BBox** value defined in the **Shading** dictionary but outside the **Domain** value defined in the **Shading** dictionary will be left unpainted. However, in the case of gradient fill patterns with a **Background** color specified, such points will be painted with the background color.

If the function is undefined at any point within the declared **Domain** value defined in the **Function** dictionary, an **undefinedresult** error may occur, even if such points are not within the **BBox** value of the **Shading** dictionary.

ShadingType 2: Axial Shading



ShadingType 2 shadings define a color blend along a line between two endpoints. The color blend is optionally extended beyond the endpoints by continuing the boundary colors. In addition to the common attributes listed in Table 4.7, a **Shading** dictionary for **ShadingType** 2 includes the following entries:

Table 4.9 *Entries in a ShadingType 2 Shading dictionary (axial shadings)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 2. Specifies axial shading.
Coords	array	(*Required*) Array of four numbers [x_0 y_0 x_1 y_1] specifying the start and end coordinate pairs.

Table 4.9 *Entries in a ShadingType 2 Shading dictionary (axial shadings) (continued)*

Key	*Type*	*Semantics*
Domain	array	(*Optional*) Array of two numbers. A parametric variable *t* is considered to vary linearly between these two values, as the blend varies between the start and end points, respectively. The variable *t* becomes the argument with which the color function(s) are called. Default value: [0 1]
Function	dictionary or array	(*Required*) A single 1-in, *n*-out **Function** dictionary, or an array of *n* 1-in, 1-out **Function** dictionaries, where *n* is the number of components in the **ColorSpace** entry. The function(s) are called with values of *t* in the **Domain** value defined in the **Shading** dictionary (above). The **Domain** value defined in the **Function** dictionary must be a superset of the **Domain** value defined in the **Shading** dictionary. If the values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.
Extend	array	(*Optional*) Array of two Boolean values specifying whether to extend the start and end colors beyond the start and end points, respectively. Default value: [*false false*]

ShadingType 2 defines a field of color that varies along the line between the start and end coordinates and extends infinitely perpendicular to that line. If the **Extend** Boolean values are *true*, the field may also extend infinitely far along the line, past either or both endpoints, with the constant color of that endpoint.

The blend is accomplished by linearly mapping the range between the endpoints to the **Domain** value defined in the **Shading** dictionary, as follows. For every point (*x, y*), it is mapped to a coordinate space where (*0, 0*) corresponds to (x_0, y_0) and (*1, 0*) corresponds to (x_1, y_1). Since all points on a line perpendicular to the line from (0, 0) to (1, 0) in that space will have the same color, only the new value of *x* needs to be computed in that space, *x′*:

$$x' = ((x_1 - x_0)(x - x_0) + (y_1 - y_0)(y - y_0)) / ((x_1 - x_0)^2 + (y_1 - y_0)^2)$$

Once *x′* is calculated, the value of the parametric value *t* can be determined. This value is used as the input argument to the **Function** key, and the returned value(s) are used to paint the blend. This parametric gradient (“vignette”) may not be used with the value of **ColorSpace** set to **Indexed** color space. *t* is determined as follows:

- If $x' < 0$ and the first value in the **Extend** array is *true*, then the parameter *t* is set to the value of t_0. However, if the first value in the **Extend** array is *false*, then that point is not colored.

- If $x' > 1$ and the second value in the **Extend** array is *true*, then the parameter t is set to the value of t_1. However, if the second value in the **Extend** array is *false*, then that point is not colored.

- If $0 <= x' <= 1$, then $t = t_0 + (t_1 - t_0)x'$.

Note *This shading is called "axial" rather than "linear" because, while the mapping from (x, y) to t is linear, the color transition defined by the* ***Function*** *key need not be.*

ShadingType 3: Radial Shading



ShadingType 3 shadings define a color blend between two circles and are commonly used to image three-dimensional spheres and cones. In addition to the common attributes listed in Table 4.7, a **Shading** dictionary for **ShadingType** 3 includes the following entries:

Table 4.10 *Entries in a ShadingType 3 Shading dictionary (radial shadings)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 3. Specifies radial shading.
Coords	array	(*Required*) Array of six numbers [$x_0\ y_0\ r_0\ x_1\ y_1\ r_1$] specifying the center coordinates and radii of the start and end circles. The radii r_0 and r_1 must be greater than or equal to zero. If one radius is zero, that circle is treated as a point. If both radii are zero, nothing is rendered.
Domain	array	(*Optional*) Array of two numbers. A parametric variable t is considered to vary linearly between these two values, as the blend varies between the start and end circles, respectively. The variable t becomes the argument with which the color function(s) are called. Default value: [0 1]
Function	dictionary or array	(*Required*) A single 1 in, *n*-out **Function** dictionary, or an array of *n* 1-in, 1-out **Function** dictionaries, where *n* is the number of components in the **ColorSpace** key. The function(s) are called with values of t in the **Domain** value defined in the **Shading** dictionary (above). The value of **Domain** defined in the **Function** dictionary must be a superset of the **Domain** value defined in the **Shading** dictionary. If the values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.
Extend	array	(*Optional*) Array of two Boolean values specifying whether to extend the start and end colors past the start and end circles, respectively. Default value: [*false false*]

When using **ShadingType** 3, note that:

- If the circle with the smaller radius is extended by the **Extend** boolean value, the interior of that circle will be painted with the constant color of that circle. If the circle with the larger radius is extended by the **Extend** boolean value, the exterior of that circle will be painted with the constant color of that circle, out to the value of the **BBox** key in the **Shading** dictionary.

- If the start and end circles are not contained one within the other and the larger radius is given first, the object will depict a cone pointing out of the page. If under the same conditions the smaller radius is given first, the object will depict a cone pointing into the page.

- If a sphere is depicted, the larger circle will entirely contain the smaller circle.

The blend is accomplished by mapping the region between the start and end circles to a linear parametric variable in the **Shading** dictionary's **Domain** key. The resulting parametric value is used as the input argument to the **Function** key. The returned value(s) are used to paint the blend.

The mathematics of the mapping are not complex. Consider the parametric variable $s = (t - t_0)/(t_1 - t_0)$, which varies linearly between 0 and 1 as t varies across the **Domain** value defined in the **Shading** dictionary. The parametric equations for the center and radius of the "blend" circle moving between the start circle and the end circle as a function of s are:

$$x_c(s) = x_0 + s \times (x_1 - x_0)$$

$$y_c(s) = y_0 + s \times (y_1 - y_0)$$

$$r(s) = r_0 + s \times (r_1 - r_0)$$

Given a geometric coordinate position (x, y) in the blend, the corresponding value of s can be determined by solving the quadratic constraint equation:

$$[x - x_c(s)]^2 + [y - y_c(s)]^2 = [r(s)]^2$$

From s, one can determine the value of t, which is then passed to the **Function** key in the **Shading** dictionary to determine the color at the position (x, y). If both roots of the equation are in the domain [0 1], then the larger value of s defines the color because it comes "later" than the smaller value and thus overlays it. For values of s outside the domain [0 1], the **Extend** boolean values determine how the shading will be painted. The following rules hold true for pixel coordinates (x, y) satisfying the above equation. If the value for the start **Extend** boolean is *false*, then pixels corresponding to values of $s < 0$ are left unpainted or are painted with the **Background** color, if

specified. If the value for the end **Extend** boolean is *false*, then pixels corresponding to values of $s > 1$ are left unpainted or are painted with the **Background** color, if specified. If the value for the start **Extend** boolean is *true* and r[s] >= 0, then pixels corresponding to values of $s < 0$ are painted with the start color. Similarly, if the value for the end **Extend** boolean is *true* and r[s] >= 0, then pixels corresponding to values of $s > 1$ are painted with the end color. Note that for cases where the one circle is not completely contained within the other, **Extend** values of *true* can cause painting to extend beyond the areas defined by the two circles.

This parametric gradient (“vignette”) may not be used with the value of **ColorSpace** set to **Indexed** color space.

ShadingType 4: Free-Form Gouraud-Shaded Triangle Meshes



ShadingType 4 shadings define a common construct used by many three-dimensional applications to image complex colored and shaded shapes. Gouraud-shaded triangle meshes construct paths composed entirely of triangles. The color at each vertex of the triangles is specified, and Gouraud interpolation is used to color the interior points. A primary use of this shading type is to allow the specification of polygon vignettes as triangle meshes, optionally with nonlinear interpolation functions.

In addition to the common attributes listed in Table 4.7, a **Shading** dictionary for **ShadingType** 4 includes the following entries:

Table 4.11 *Entries in a ShadingType 4 Shading dictionary (free-form Gouraud-shaded triangle meshes)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 4. Specifies free-form Gouraud-shaded triangle mesh.
DataSource	(various)	(*Required*) Provides the sequence of vertex coordinate and color data that specifies the free-form triangle mesh. A value of **DataSource** must be provided; it may be an array of numbers, a string, or a stream.
BitsPerCoordinate	integer	(*Required, unless the value of **DataSource** is an array*) Specifies the number of bits used to represent each vertex coordinate. The data is decoded based on the **Decode** key, in much the same way as color component data is decoded in image dictionaries. Allowed values are 1, 2, 4, 8, 12, 16, 24, and 32.
BitsPerComponent	integer	(*Required, unless the value of **DataSource** is an array*) Specifies the number of bits used to represent each color component. The data is decoded based on the **Decode** key, as in image dictionaries. Allowed values are 1, 2, 4, 8, 12, and 16.

Table 4.11 *Entries in a ShadingType 4 Shading dictionary (free-form Gouraud-shaded triangle meshes) (continued)*

Key	*Type*	*Semantics*
BitsPerFlag	integer	(*Required, unless the value of* ***DataSource*** *is an array*) The number of bits used to represent the edge flag for each vertex. The allowed values of **BitsPerFlag** are 2, 4, and 8. Allowed values for the edge flag are 0, 1, and 2.
Decode	array	(*Required, unless the value of* ***DataSource*** *is an array*) Specifies how to decode coordinate and color component values into the ranges of numbers appropriate for these values. The ranges are specified as: $[x_{min}\ x_{max}\ y_{min}\ y_{max}\ c_{1,min}\ c_{1,max} \cdots c_{n,min}\ c_{n,max}]$
Function	dictionary or array	(*Optional*) A single 1-in, *n*-out **Function** dictionary, or an array of *n* 1-in, 1-out **Function** dictionaries, where *n* is the number of components in the **ColorSpace** key. If present, the mesh's vertex color data must be specified by single values rather than color tuples, and the **Function** key will be called with each interpolated color value to determine the actual color of each point. The **Domain** value defined in the **Function** dictionary must be a superset of the range specified in the **Decode** array for the color value defined in the **Shading** dictionary. If **DataSource** is an array, in which case **Decode** is not defined, the **Domain** value defined in the **Function** dictionary must be a superset of the domain [0 1]. In both cases input values will be clipped to the subset of the function domain defined above. The **Function** key may not be used with unencoded vertex data, nor with the **ColorSpace** key set to **Indexed** color space. If the values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.

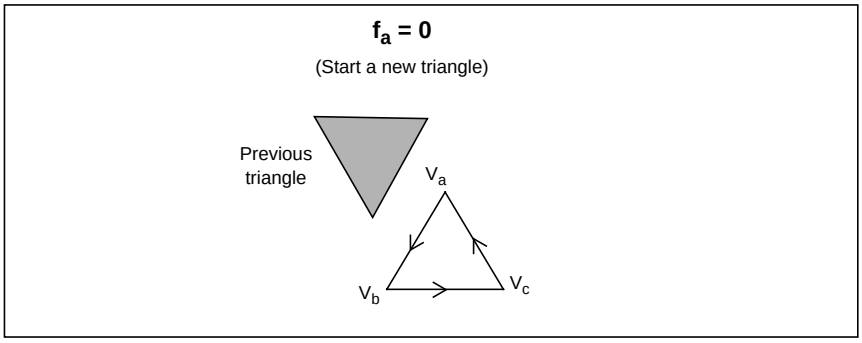
The **DataSource** key provides a sequence of vertex data. The data contain a sequence of vertex coordinates and color data that specify the triangle mesh. The i^{th} vertex, v_i, is represented by:

$$f_i\ x_i\ y_i\ c_{i,\,1} \ldots c_{i,\,n}$$

where f is the edge flag for each vertex, x and y are vertex coordinates, c is a tuple of color values, and n is the number of color components. If the dictionary specifies a **Function** key, then only one color component, $c_{(i,\,1)}$, is permitted in each sequence. Triangles are built up as described below.

To draw a new triangle, the first vertex v_a must have an edge flag value $f_a = 0$, which means "start a new triangle." At least two more vertices must be provided (v_b and v_c), and their edge flags will be ignored. These three vertices define the triangle (v_a, v_b, v_c), as shown in Figure 4.3.

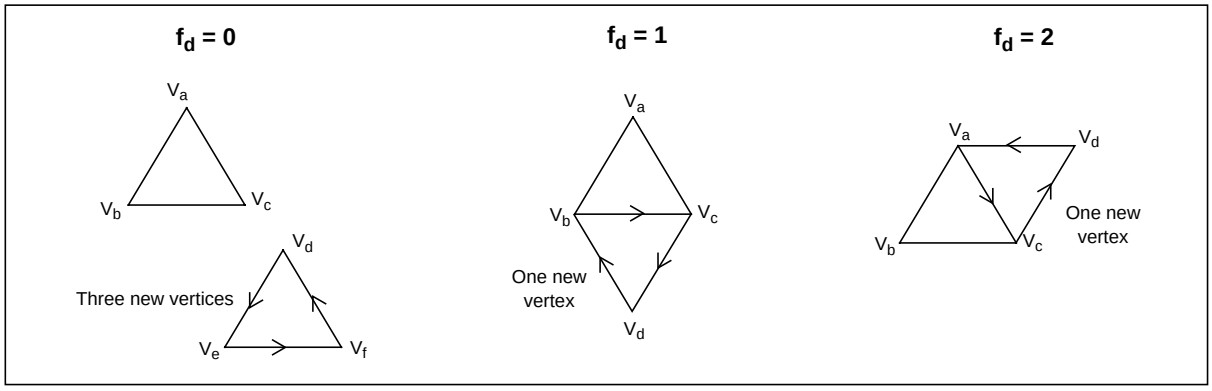
Figure 4.3 *Drawing a new triangle (f = 0): Free-form Gouraud-shaded triangle meshes*



Subsequent triangles are defined by a single new vertex and two vertices of the preceding triangle. Given triangle (v_a, v_b, v_c), where vertex *a* is older than vertex *b* and vertex *b* is older than vertex *c* (older means earlier in the data source), a new triangle can be formed on side v_{bc} or side v_{ac}, using a new vertex v_d, as shown in Figure 4.4. If the edge flag is $f_d = 1$ (side v_{bc}), the next vertex forms the triangle (v_b, v_c, v_d). If the edge flag is $f_d = 2$ (side v_{ac}), the next vertex forms the triangle (v_a, v_c, v_d). The edge on side v_{ab} is assumed to be shared with a preceding triangle and so is not appropriate for continuing the mesh tiling.

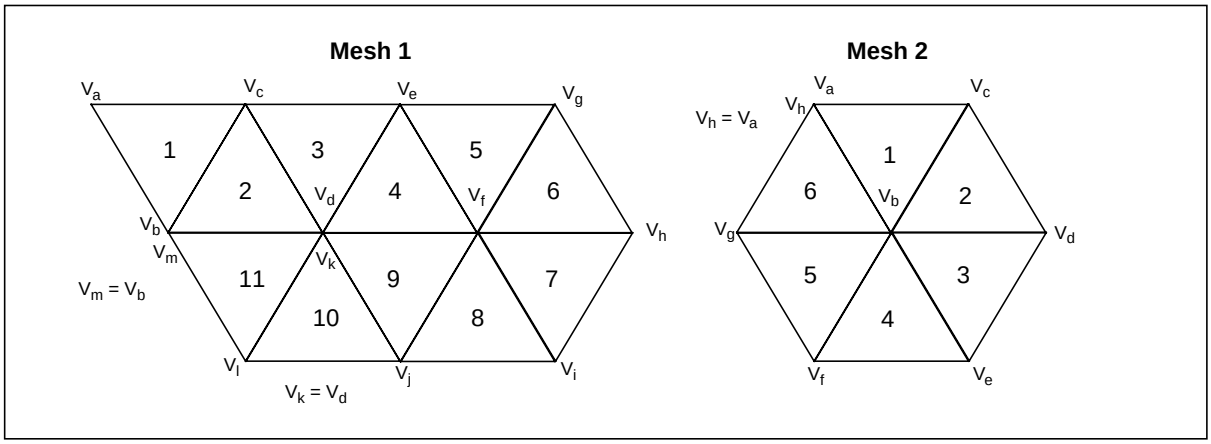
Whenever the edge flag is zero, a new triangle is started. At least two vertices must be provided, but their edge flags will be ignored. The value $f = 3$ is not allowed.

Figure 4.4 *How the value of the edge flag determines which edge is used for the next triangle*



Using triangle meshes, it is possible to create complex shapes simply by varying the edge on which subsequent triangles are formed. Figure 4.5 shows two very simple examples.

Figure 4.5 *Varying the value of the edge flag to create different shapes*



To create Mesh 1, start with triangle 1 and draw each triangle using the following edge flags: 1 ($f_a = f_b = f_c = 0$), 2 ($f_d = 1$), 3 ($f_e = 1$), 4 ($f_f = 1$), 5 ($f_g = 1$), 6 ($f_h = 1$), 7 ($f_i = 2$), 8 ($f_j = 2$), 9 ($f_k = 2$), 10 ($f_l = 1$), 11 ($f_m = 1$). To create Mesh 2, start with triangle 1 and draw each triangle using the following edge flags: 1 ($f_a = f_b = f_c = 0$), 2 ($f_d = 1$), 3 ($f_e = 2$), 4 ($f_f = 2$), 5 ($f_g = 2$), 6 ($f_h = 2$).

This representation optimizes useful tiling meshes, although it complicates the data representation. The value of **DataSource** must provide a whole number of triangles with appropriate vertex edge flags; otherwise, a **rangecheck** error will occur. If the mesh contains only a few vertices, the vertices may be represented by a simple array of numbers. In this case, only the **ShadingType** and **DataSource** attributes are relevant. If the mesh contains many vertices, the data should be encoded compactly and drawn from a stream. The encoding is specified by the **BitsPerFlag**, **BitsPerCoordinate**, **BitsPerComponent**, and **Decode** attributes. Each vertex edge flag is expressed in **BitsPerFlag** bits, each vertex coordinate pair is expressed in $2 \times$ **BitsPerCoordinate** bits, and each vertex color tuple is expressed in $n \times$ **BitsPerComponent** bits. Each set of vertex data takes an integer number of bytes; if the total number of bits in the vertex data is not divisible by eight, the vertex data is padded with ignored bits inserted between the color data and the next set of vertex data. The coordinate and color data is decoded based on the **Decode** array, as with image data.

If a value of the **Function** key is present, the mesh's vertex color data must be specified by single values (*t)*, rather than color tuples. All linear interpolation within the triangle mesh will be done using the values of *t*, and after interpolation, the **Function** key will be called with each value to determine the color of each point.

Note *Using free-form Gouraud-shaded triangle meshes differs from using an* ***Indexed*** *color space for the shading. If an* ***Indexed*** *color space is used, the vertex color value is converted to the base color space first, and linear interpolation occurs in that color space. Thus, there is no opportunity to effect a nonlinear interpolation using an* ***Indexed*** *color space.*

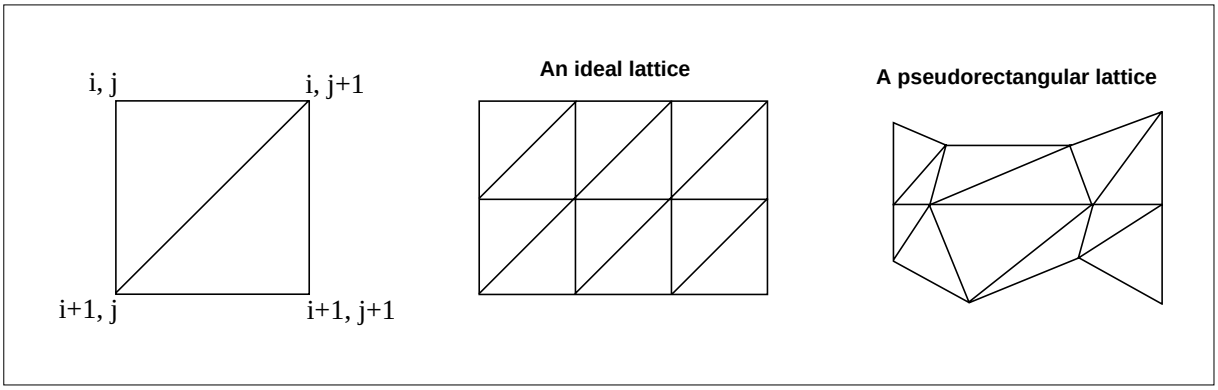
ShadingType 5: Lattice-Form Gouraud-Shaded Triangle Meshes

ShadingType 5 shadings are identical to **ShadingType** 4 shadings, except in the interpretation of the **DataSource** key. Also, edge flags are not used in **ShadingType** 5. In **ShadingType** 4, vertices are specified in free-form geometry; in **ShadingType** 5, vertices are arranged in a lattice, which is topologically equivalent to a rectangular grid (a pseudorectangular lattice, as shown in Figure 4.6). The lattice need not be rectangular, but the set of vertices must be organized into "rows." The rows do not need to be geometrically linear. Given *m* rows of *n* vertices, where the number of vertices is defined by **VerticesPerRow**, as described in Table 4.12, triangles are constructed using the following triplets of vertices:

$(V_{i,j}, V_{i,j+1}, V_{i+1,j})$ for 0 <= i <= m–2, 0 <= j <= n–2

$(V_{i,j+1}, V_{i+1,j}, V_{i+1,j+1})$ for 0 <= i <= m–2, 0 <= j <= n–2

Figure 4.6 *Sample lattice forms*



In addition to the common attributes listed in Table 4.7 on page 78, a **Shading** dictionary for **ShadingType** 5 includes the following entries:

Table 4.12 *Entries in a ShadingType 5 Shading dictionary (lattice-form Gouraud-shaded triangle meshes)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 5. Specifies lattice-form Gouraud-shaded triangle mesh.
DataSource	(various)	(*Required*) Provides the sequence of vertex coordinate and color data that specifies the lattice-form triangle mesh. A value for **DataSource** may be an array of numbers, a string, or a stream.

Table 4.12 *Entries in a ShadingType 5 Shading dictionary (lattice-form Gouraud-shaded triangle meshes) (continued)*

Key	*Type*	*Semantics*
BitsPerCoordinate	integer	(*Required, unless* ***DataSource*** *is an array*) Specifies the number of bits used to represent each vertex coordinate. The data is decoded according to the value of **Decode**, in much the same way as color component data is decoded in image dictionaries. Allowed values are 1, 2, 4, 8, 12, 16, 24, and 32.
BitsPerComponent	integer	(*Required, unless* ***DataSource*** *is an array*) Specifies the number of bits used to represent each color component. The data is decoded according to the value of **Decode**, as in image dictionaries. Allowed values are 1, 2, 4, 8, 12, and 16.
Decode	array	(*Required, unless* ***DataSource*** *is an array*) Specifies how to decode coordinate and color component values into the ranges of numbers appropriate for these values. The ranges are specified as: $[x_{min}\ x_{max}\ y_{min}\ y_{max}\ c_{1,min}\ c_{1,max} \ldots\ c_{n,min}\ c_{n,max}]$
VerticesPerRow	integer	(*Required*) Defines the number of vertices in each "row" of the lattice. The number of rows does not need to be specified. Must be greater than or equal to 2.
Function	dictionary or array	(*Optional*) A single 1-in, *n*-out **Function** dictionary, or an array of *n* 1-in, 1-out **Function** dictionaries, where *n* is the number of components in the **ColorSpace** entry. If present, then the mesh's vertex color data must be specified by single values rather than color tuples, and **Function** will be called with each interpolated color value to determine the actual color of each point. The **Domain** value defined in the **Function** dictionary must be a superset of the range specified in the **Decode** array for the color value defined in the **Shading** dictionary. If **DataSource** is an array, in which case **Decode** is not defined, the **Domain** value defined in the **Function** dictionary must be a superset of the domain [0 1]. In both cases input values will be clipped to the subset of the function domain defined above. **Function** may not be used with unencoded vertex data, nor with **ColorSpace** set to **Indexed** color space. If the values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.

ShadingType 6: Coons Patch Meshes



ShadingType 6 shadings are constructed from one or more color patches, each bounded by four Bézier curves. A primary use of this feature is to allow the specification of "conical" vignettes and other complex blends as patch meshes with nonlinear interpolation functions. The geometry of these patches is defined below, followed by the specification of the entries in the **Shading** dictionary associated with **ShadingType** 6.

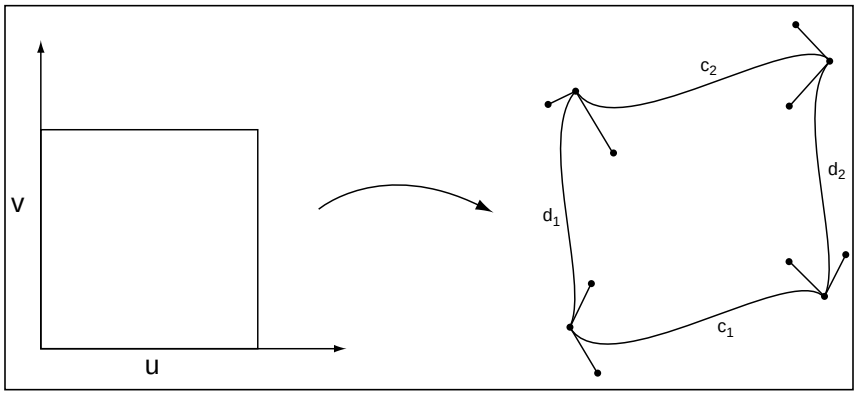
A Coons patch generally has two independent aspects, a color specification and a coordinate mapping. These are defined as follows:

- Colors are specified for each of the corners of the unit square, and bilinear interpolation is used to fill in colors over the entire unit square.
- Coordinates are mapped from the unit square into a four-sided patch whose sides are not necessarily linear. The mapping is continuous; the corners of the unit square map to corners of the patch, and the sides of the unit square map to sides of the patch, as shown in Figure 4.7.

A **ShadingType** 6 shading results from coloring the unit square and then mapping it.

A bicubic Coons patch maps the unit square to a region that is bounded by four Bézier curves, c_1, c_2, d_1, and d_2.

Figure 4.7 *Coons patch meshes: Coordinate mapping from a unit square to a four-sided patch*



Two surfaces can be described that are linear interpolations over a pair of boundary curves. Along the *u* axis, the surface, S_c, is

$$S_c(u, v) = (1–v) \times c_1(u) + (v) \times c_2(u)$$

Along the *v* axis, the surface, S_d, is

$$S_d(u, v) = (1–u) \times d_1(v) + (u) \times d_2(v)$$

The four corners are

$$c_1(0) = d_1(0),\ c_1(1) = d_2(0),\ c_2(0) = d_1(1),\ \text{and}\ c_2(1) = d_2(1)$$

A third surface is the bilinear interpolation of the four corners

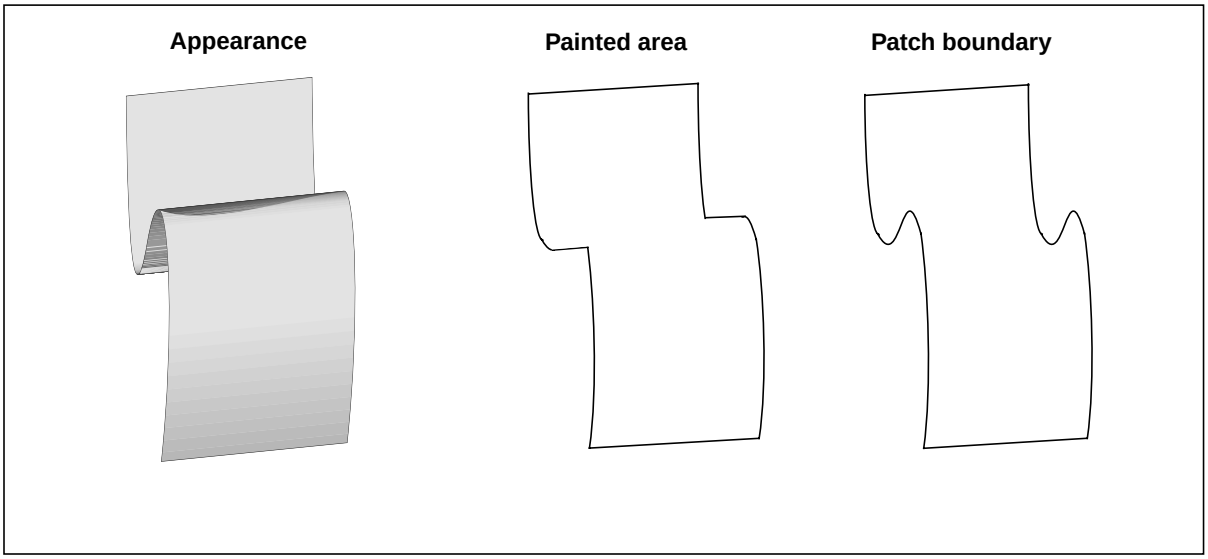
$$S_b(u,v) = (1-v) \times [(1-u) \times c_1(0) + (u) \times c_1(1)] + (v) \times [(1-u) \times c_2(0) + (u) \times c_2(1)]$$

The coordinate mapping for the shading is defined as the surface $S = S_c + S_d - S_b$. This defines the geometry of each patch. A patch mesh is constructed from a sequence of one or more such colored patches.

Patches can sometimes appear to fold over on themselves—for example, if a boundary curve is self-intersecting. Foldovers occur as follows. Consider mapping from (u,v) parameter space to the patch in device space. As the value of u or v increases in parameter space, the location of the pixels in device space may change direction so that pixels are mapped onto previously mapped pixels. If more than one parameter space location (u,v) is mapped to the same location in device space, the value of (u,v) selected will be that with the largest value of v, and if multiple (u,v) values have the same v, that with the largest value of u.

Note also that the patch is a control surface rather than a painting geometry. The outline of a projected square (i.e., the painted area) may not be the same as the patch boundary if, for example, the patch folds over on itself, as shown in Figure 4.8.

Figure 4.8 *Coons patch meshes: Appearance, painted area, and boundary*



If a “mesh” contains several patches, and if some portions of one patch overlap portions of another patch, later patches paint over earlier patches, where “earlier” and “later” refer to the order of appearance of the patches in the **DataSource** entry.

In addition to the common attributes listed in Table 4.7, a **Shading** dictionary associated with **ShadingType** 6 includes the following entries:

Table 4.13 *Entries in a ShadingType 6 Shading dictionary (Coons patch meshes)*

Key	*Type*	*Semantics*
ShadingType	integer	(*Required*) Always 6. Specifies a Coons patch mesh.
DataSource	(various)	(*Required*) Provides the sequence of coordinate and color data that specifies the patch mesh. A value of **DataSource** must be provided; it may be an array of numbers, a string, or a stream. If the total number of bits used to represent the patch data is not divisible by eight, then the patch data is padded with ignored bits inserted between the color data and the start of the next set of patch data.
BitsPerCoordinate	integer	(*Required, unless* ***DataSource*** *is an array*) Specifies the number of bits used to represent each geometric coordinate. The data is decoded based on the **Decode** entry, in the same way color component data is decoded in image dictionaries. Allowed values are 1, 2, 4, 8, 12, 16, 24, and 32.
BitsPerComponent	integer	(*Required, unless* ***DataSource*** *is an array*) Specifies the number of bits used to represent each color component. The data is decoded based on the value of the **Decode** entry, as in image dictionaries. Allowed values are 1, 2, 4, 8, 12, and 16.
BitsPerFlag	integer	(*Required, unless* ***DataSource*** *is an array*) The number of bits used to represent the edge flag for each patch. The allowed values of **BitsPerFlag** are 2, 4, and 8, but only the least significant two bits in each flag value are used. Allowed values for the edge flag are 0, 1, 2, and 3.
Decode	array	(*Required, unless* ***DataSource*** *is an array*) Specifies how to decode coordinate and color component values into the ranges of numbers appropriate for these values. The ranges are specified as: $[x_{min}\ x_{max}\ y_{min}\ y_{max}\ c_{1,min}\ c_{1,max} \ldots\ c_{n,min}\ c_{n,max}]$
Function	dictionary or array	(*Optional*) A single 1-in, *n*-out **Function** dictionary, or an array of *n* 1-in, 1-out **Function** dictionaries, where *n* is the number of components in the **ColorSpace** entry. If present, then the mesh's vertex color data must be specified by single values rather than color tuples, and the **Function** key will be called with each interpolated color value to determine the actual color of each point. The value of **Domain** defined in the **Function** dictionary must be a superset of the range specified in the value of the **Shading** dictionary's **Decode** array for the color value. If **DataSource** is an array, in which case **Decode** is not defined, the **Domain** value defined in the **Function** dictionary must be a superset of the domain [0 1]. In both cases input values will be clipped to the subset of the function domain defined above. The **Function** key may not be used with unencoded vertex data, nor with the value of **ColorSpace** set to **Indexed** color space. If values returned by the functions are out of range for a given color component, the values will be adjusted to the nearest allowed value.

The **DataSource** provides a sequence of coordinates and color component values that are interpreted in the same way as in a triangle mesh shading, and with similar constraints, except that each patch's coordinate pairs are provided first, followed by its color values. Color values are specified for the corners of the patch, in the same order as the control points corresponding to the corners. Thus, c_1 is the color at (x_1, y_1), c_2 is the color at (x_4, y_4), c_3 is the color at (x_7, y_7), and c_4 is the color at (x_{10}, y_{10}), as shown in Figure 4.9.

Figure 4.9 *Color values and edge flags in Coons patch meshes*

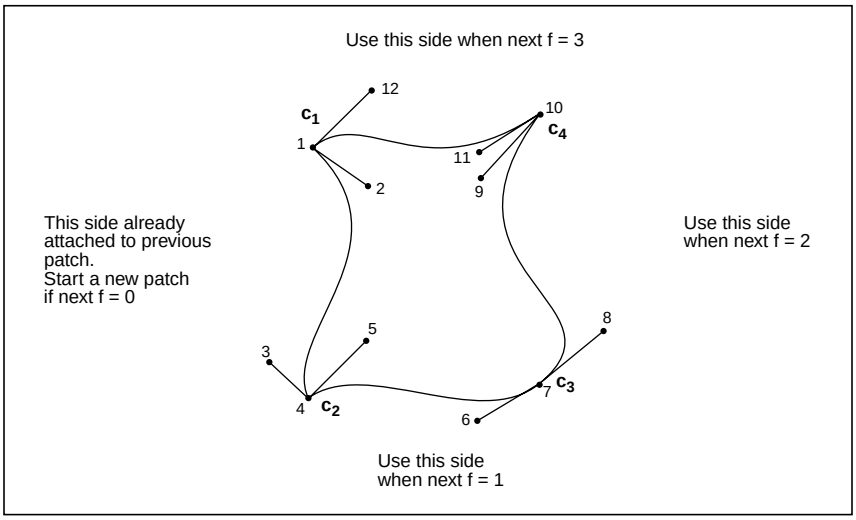


Figure 4.9 also shows how the edge values ($f = 0$, $f = 1$, $f = 2$, $f = 3$) correspond to the coordinates that describe the sides. As with triangle mesh shading, an edge flag for each patch allows specification of a shared edge with the previous patch. This arrangement improves the efficiency of the representation for meshes but complicates the data representation and stream compression. Table 4.14 below and Figure 4.10 on page 96 illustrate what this encoding would look like for each value of an edge flag. Figure 4.10 shows a new patch (Patch *A*) and the placement of the subsequent patch that is created (Patch *B*), depending on the edge flag value.

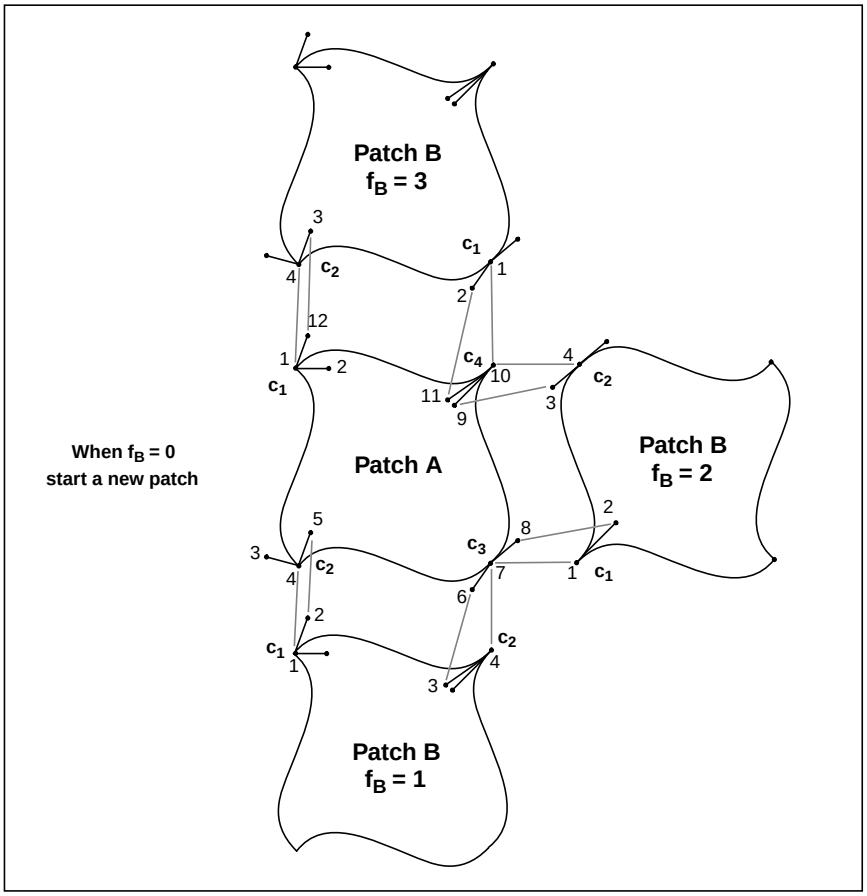
The edge flag of the first patch, f_A, must have the value zero, which means "start a new patch." The twelve control points, $x_1\, y_1\, x_2\, y_2 \ldots x_{12}\, y_{12}$, specify the Bézier curves that define the boundary curves. $c_1\, c_2\, c_3\, c_4$ is a sequence of $4 \times n$ color values, where n is the number of components per color in the **ColorSpace** attribute specifying the vertex colors. Degenerate Bézier curves

are allowed and are useful for certain graphical effects. At least one complete patch must be specified.

Table 4.14 *Coons patch meshes: Coordinates for adjacent patches*

Edge flag	*Next set of vertices*
f = 0	$x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4\ x_5\ y_5\ x_6\ y_6\ x_7\ y_7\ x_8\ y_8\ x_9\ y_9\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}$ $c_1\ c_2\ c_3\ c_4$ (new patch; no implicit values)
f = 1	$x_5\ y_5\ x_6\ y_6\ x_7\ y_7\ x_8\ y_8\ x_9\ y_9\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}$ $c_3\ c_4$ Implicit values: $(x_1\ y_1) = (x_4\ y_4)$ previous $(x_2\ y_2) = (x_5\ y_5)$ previous $(x_3\ y_3) = (x_6\ y_6)$ previous $(x_4\ y_4) = (x_7\ y_7)$ previous $(c_1) = (c_2)$ previous $(c_2) = (c_3)$ previous
f = 2	$x_5\ y_5\ x_6\ y_6\ x_7\ y_7\ x_8\ y_8\ x_9\ y_9\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}$ $c_3\ c_4$ Implicit values: $(x_1\ y_1) = (x_7\ y_7)$ previous $(x_2\ y_2) = (x_8\ y_8)$ previous $(x_3\ y_3) = (x_9\ y_9)$ previous $(x_4\ y_4) = (x_{10}\ y_{10})$ previous $(c_1) = (c_3)$ previous $(c_2) = (c_4)$ previous
f = 3	$x_5\ y_5\ x_6\ y_6\ x_7\ y_7\ x_8\ y_8\ x_9\ y_9\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}$ $c_3\ c_4$ Implicit values: $(x_1\ y_1) = (x_{10}\ y_{10})$ previous $(x_2\ y_2) = (x_{11}\ y_{11})$ previous $(x_3\ y_3) = (x_{12}\ y_{12})$ previous $(x_4\ y_4) = (x_1\ y_1)$ previous $(c_1) = (c_4)$ previous $(c_2) = (c_1)$ previous

Figure 4.10 *Coons patch meshes: How the value of the edge flag determines the edge for the next patch*



If a value of the **Function** key is present, the mesh's vertex color data must be specified by single values, *t*, rather than color tuples. All linear interpolation within the mesh will be done using the values *t*, and after interpolation, the **Function** key will be called with each value to determine the color of each point.

Note *Using **ShadingType** 6 differs from using an **Indexed** color space for the shading. If an **Indexed** color space is used, the vertex color value is converted to the base color space first, and interpolation occurs in that color space. Thus, there is no opportunity to effect a nonlinear interpolation using an **Indexed** color space.*

ShadingType 7: Tensor Product Patch Meshes

ShadingType 7 shadings are identical to **ShadingType** 6 shadings, except that instead of a bicubic Coons patch defined by twelve control points, a bicubic tensor product patch defined by sixteen control points is used. Each set of twelve coordinate pairs in the **DataSource** key is replaced by a set of

sixteen coordinate pairs. The two cases are distinguished only by the **ShadingType** key and the data in the **DataSource** key; there are no other differences, except in the way the color varies across the geometry.

As with the Coons patch surface, the tensor product surface is defined by a mathematical mapping from a square patch (*u*,*v*) to the patch coordinate system (*x*, *y*).

$$S(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} Pij \times Bi(u) \times Bj(v)$$

Pij is the control point for the *i,j* row and column of the tensor. Since each *Pij* = (*Xij*, *Yij*), one can also express this as

$$x(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} Xij \times Bi(u) \times Bj(v)$$

$$y(u,v) = \sum_{i=0}^{3} \sum_{j=0}^{3} Yij \times Bi(u) \times Bj(v)$$

Bi(u) and *Bj(v)* are Bernstein polynomials, where

$$B0(t) = (1-t)^3$$

$$B1(t) = 3t(1-t)^2$$

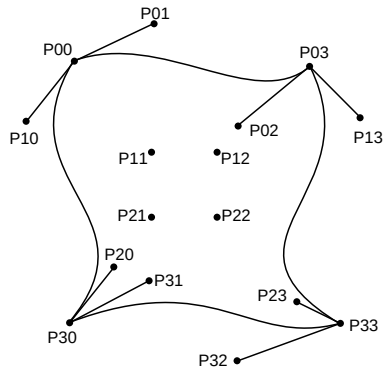
$$B2(t) = 3t^2 \times (1-t)$$

$$B3(t) = t^3$$

An illustration may help the mathematical description. *Pij* are defined as follows:

P00	P01	P02	P03
P10	P11	P12	P13
P20	P21	P22	P23
P30	P31	P32	P33

Graphically, this is represented as follows:



This is a convenient numbering for the mathematical description, but a better numbering for the language is:

0	11	10	9
1	12	15	8
2	13	14	7
3	4	5	6

This allows the Coons patch numbering to be a subset of the tensor product patch numbering.

The tensor product patch mapping, like the Coons patch mapping, is controlled by the location and shape of the four boundary curves. Unlike the Coons patch, however, the tensor product patch has four more “internal” control points to adjust the mapping. For example, imagine a cubic Bézier curve moving from the top boundary of the patch to the bottom. The control points of this curve start as the top curve’s control points and end as the bottom curve’s control points. The curve changes shape as it moves down the patch because the control points change position. Each control point follows a trajectory defined by the four column control points. In particular, each column defines its own cubic Bézier curve, and this is the trajectory each of the control points of the moving curve takes. The same can be said for a curve moving from left to right, except the trajectory taken by the control points is defined by the curves specified by four rows of control points.

The tensor product patch gives more control over mapping than does the Coons patch. However, the Coons patch is easier to use and more concise because the internal control points are implicitly specified by the boundary control points.

4.4.3 Painting with a Pattern Dictionary



When a painting operation is performed with a gradient pattern dictionary as the current color in the graphics state, the pattern coordinate space is obtained in the same way as with **PatternType** 1 patterns. However, instead of executing a **PatternType** 1 **PaintProc** procedure, the shape and color transitions described by the **Shading** dictionary are interpreted relative to this coordinate space to create a logical “paint” with which graphical objects can be rendered. The **Shading** dictionary’s **BBox** attribute, if present, will clip the logical painting region, which may also be affected by the geometry of the shading. All color values are interpreted relative to the **Shading** dictionary’s **ColorSpace** attribute.

If a **Background** color is defined in the **Shading** dictionary, then that color is used first to fill the background of the object being painted. Logically, this is

equivalent to executing a **fill** operation or other painting operation first with the background color and then with the gradient pattern. The convenience of the **Background** color specification is provided because performing this sequence of operations explicitly would be verbose, especially for text or image masks. This is only of interest for gradient patterns that do not cover the entire area of the object being painted.

Some values of the **ShadingType** key allow arbitrarily large streams of data as an attribute of the **Shading** dictionary. Since gradient paints defined by **PatternType** 2 pattern resources may be used multiple times, the data must be provided in a reusable form. Data stored in a string are reusable, but are limited to 64 kilobytes. Files pointed to by **currentfile** or files stored on hard disk are not reusable. Data from such sources must be converted into a reusable stream by means of the **ReusableStreamDecode** filter. A nonreusable stream of **Shading** data may only be used with the **shfill** operator.

4.5 In-RIP Trapping

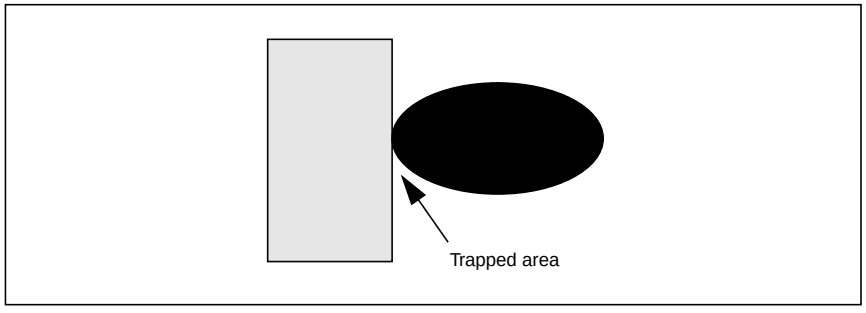


Quality reproduction of color documents on offset presses requires trapping to correct for misregistrations that result from the physical limitations of the printing press. *Trapping* is the process of creating an overlap between two adjacent colors to avoid the occurrence of white space; the trap is the area where the colors overlap.

Trapping supported by the PostScript language is referred to as in-RIP trapping. It enables the user to specify regions on the page and a set of trapping parameters, and then it automatically calculates traps for each region.

Figure 4.11 shows an example of trapping. The area where the two shapes touch would be trapped.

Figure 4.11 *An example of trapping*



4.5.1 Setting Trapping Zones for In-RIP Trapping



Areas or trapping zones within which the RIP will trap a graphic object are specified by the user. A graphic object will be trapped if it falls within one or several trapping zones, but only the part of the object that falls within the trapping zone or trapping zones will be trapped.

Trapping zones can be defined by the user through the trapping zone operator **settrapzone**, described in Chapter 8 on page 206. Each trapping zone has a unique set of trapping parameters. Default trapping zones are defined by the **Install** procedure of the **setpagedevice** operator and apply to all pages that use the defined page device.

Trapping zones can overlap all or part of another trapping zone. Whenever trapping zones overlap, the parameters controlling the most recently defined trapping zone (the “top” trapping zone) take effect.

Trapping zones should not be defined over existing marks on the page. The results are implementation dependent and unreliable.

A trapping zone defined in a page description is erased when the page is printed or the page device is redefined. A default trapping zone is reestablished at the beginning of each page, before the **BeginPage** procedure is executed, and is erased when the page device is redefined.

4.5.2 Trapping Parameters



Within a trapping zone, trapping is controlled by two sets of parameters:

- *The **Trapping** and **TrappingDetails** page device parameters* that apply to all zones on pages printed with a given page device. These parameters are used primarily to specify ink characteristics. See Section 4.6.1, “Page Device Parameters,” on page 104 and Table 4.17 on page 115.

  The **TrappingDetails** dictionary has three levels. The top level of the dictionary holds global parameters and a **ColorantDetails** dictionary. The **ColorantDetails** dictionary (Table 4.21 on page 119) must have one entry for each process color and for each entry in **SeparationColorNames**. Each entry is a dictionary (Table 4.22 on page 120).

  Page device parameters are stored in the page device dictionary and cannot be changed after a page device has been defined.

- *Zone-specific parameters* that specify the desired trapping behavior within the zone. This trapping parameter set is cloned from the current trapping parameter set at the time the trapping zone is defined. Thus, zones with different parameter sets can be defined by alternately changing the current

trapping parameter set and defining zones. See Section 4.5.3, “Trapping Parameter Dictionary.”

The current trapping parameter set is stored as a dictionary in VM and obeys the rules of the **save** and **restore** operators. After a zone has been defined, its parameter set cannot be changed.

4.5.3 Trapping Parameter Dictionary



Entries in a trapping parameter dictionary define trapping and ink parameters that are zone specific. Table 4.15 defines entries that are valid for trapping method **Type** 1001, as specified in the **TrappingDetails** dictionary. See Section 4.6.3, “TrappingDetails and ColorantDetails Dictionaries,” on page 119.

The **ImageInternalTrapping**, **ImageToObjectTrapping**, and **ImageTrapPlacement** entries apply to images but not to image masks. An image mask is not trapped as an image, but as a fill.

Table 4.15 *Entries in a trapping parameter dictionary*

Key	*Type*	*Semantics*
BlackColorLimit	number	Specifies the lowest color value required for trapping a colorant based on the black trapping rule. This entry uses the subtractive notion of color, where 0 is white or no colorant, and 1 is full colorant. Range: 0–1
BlackDensityLimit	number	Specifies the lowest neutral density of a colorant for trapping based on the black trapping rule. Range: Positive
BlackWidth	number	Trap width for trapping based on the black trapping rule, specified in units of the **TrapWidth** entry. A value of 1 means that the black trap width is one **TrapWidth** wide. The resulting black trap width is subject to the same device limits as **TrapWidth**. Range: Positive
ColorantZoneDetails	dictionary	(*Optional*) Contains colorant-specific parameters. See Table 4.16 on page 103.
Enabled	boolean	If *true*, trapping is enabled in this zone.
HalftoneName	name or *null*	(*Optional*) Name of a **Halftone** resource with which to mark traps. It is recommended that the resource be defined in global VM. The **Halftone** resource may be used at unpredictable times, including at a lower save level than the current save level. If the entry is *null* or undefined, the halftone in effect just before the traps are marked will be used, which may create unexpected results.
ImageInternalTrapping	boolean	Controls whether the planes of a color image are trapped against each other.

Table 4.15 *Entries in a trapping parameter dictionary (continued)*

Key	*Type*	*Semantics*
ImageResolution	integer	Minimum resolution in dots per inch for downsampled images. Images can be downsampled by a power of two before traps are calculated. The downsampled image is used only for calculating traps, while the original image is used for printing the image.
ImageToObjectTrapping	boolean	Controls whether images are trapped to other objects.
ImageTrapPlacement	name	Controls the placement of traps for images, as defined below: /Center: Center trap is centered on the edge between the image and the adjacent object. /Choke: Choke trap is placed in the image. /Spread: Spread trap is placed in the adjacent object. /Normal: Normal trap is placed based on the colors of the areas.
SlidingTrapLimit	number	Specifies when to slide traps towards a center position. If the neutral density of the lighter area is greater than the neutral density of the darker area multiplied by the value of **SlidingTrapLimit**, then the trap slides. This applies to vignettes and nonvignettes. Range: 0–1 (No slide occurs at 1.)
StepLimit	number	Specifies the smallest relative step required in the color value of a colorant to trigger trapping at a given boundary. If the higher color value at the boundary exceeds the lower value by an amount that is equal to or greater than the larger of the **StepLimit** value multiplied by the lower value or 0.05, that is, (*low* + *max* (**StepLimit** × *low*, 0.05)), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. This entry can be overridden for a colorant if the colorant has an entry in the **ColorantZoneDetails** dictionary with the corresponding key defined. Range: 0–1
TrapColorScaling	number	Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. A value of 1 means the trap has the combined color values of the darker and the lighter area. A value of 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. This entry can be overridden for a colorant if the colorant has an entry in the **ColorantZoneDetails** dictionary with the corresponding key defined. Range: 0–1
TrapSetName	name, string, or *null*	(*Optional*) This entry can be used to identify the trapping parameter set by the user. It is not used in the RIP. This key should be defined only when the argument *dict* reflects the contents of a resource with this name.

Table 4.15 *Entries in a trapping parameter dictionary (continued)*

Key	*Type*	*Semantics*
TrapWidth	number	Trap width in points. Also defines the unit used in trap width specifications for certain types or objects, such as the **BlackWidth** key. The valid range is device dependent, usually at least 1–40 pixels in device space.
	Note	*Trap widths are defined in points, where point is a unit of the default user coordinate system (1/72 inch). The unit is not affected by the current scaling factor, just as the line frequency of a screen is not affected by the current matrix.*

4.5.4 ColorantZoneDetails Dictionary



The **ColorantZoneDetails** dictionary can contain zero or more entries. Each key in the dictionary is the name of a color or a device colorant. The value of the entry is always a colorant dictionary, and the dictionary contains the parameters for the color or device colorant (Table 4.16). If no dictionary is defined for a specific color or device colorant, the relevant parameters in the trapping parameter dictionary are used. If the named color is not part of the **ProcessColorModel** or **SeparationColorNames** dictionary when the **settrapparams** operator is called, the entry is not placed in the trapping parameter set.

Table 4.16 *Entries in a colorant dictionary*

Key	*Type*	*Semantics*
StepLimit	number	(*Optional*) Specifies the smallest step required in the color value of a colorant to trigger trapping at a given boundary. If the higher color value at the boundary exceeds the lower value by an amount that is equal to or greater than the larger of **StepLimit** multiplied by the lower value or 0.05, (*low* + *max* (**StepLimit** × *low*, 0.05)), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. If omitted, then the **StepLimit** entry in the trapping parameter dictionary is used.
TrapColorScaling	number	(*Optional*) Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. A value of 1 means the trap has the combined color values of the darker and the lighter area. A value of 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. If omitted, then the **TrapColorScaling** entry in the trapping parameter dictionary is used.

Commonly used trapping parameter sets and color defaults can be stored as resources, as describe in Chapter 3, “Language.”

4.6 Device Setup

4.6.1 Page Device Parameters

Table 4.17 describes page device dictionary entries, commonly called *page device parameters*, that have been added or modified since the publication of the *PostScript Language Reference Manual, Second Edition*. These page device parameters are part of a dictionary of key-value pairs that is used to model the internal state of a *page device*, where the page device is usually a raster output device. The keys in the dictionary represent features or processing options; the values represent the current settings of those features or options and reflect the device setup. The values are altered or read using the PostScript operators **setpagedevice** and **currentpagedevice.**

Note *The PostScript interface to fax is not supported in versions of the PostScript software beyond version 2017.*

For further information about the PostScript language facilities for setting up a raster output device, see Section 4.11, "Device Setup," in the *PostScript Language Reference Manual, Second Edition*.

The following entries in a page device dictionary (parameters) are described in the *PostScript Language Reference Manual, Second Edition*. The description of these entries is unchanged:

AdvanceDistance
AdvanceMedia
BeginPage
Collate
CutMedia
Duplex
EndPage
HWResolution
ImagingBBox
Install
ManualFeed
MediaColor
MediaType
MediaWeight
MirrorPrint
NegativePrint
NumCopies
Orientation
OutputAttributes
OutputFaceUp
OutputType
Policies
Tumble

Page Device Dictionary

Table 4.17 describes entries in the page device dictionary that have been defined or modified since the publication of the *PostScript Language Reference Manual, Second Edition.*

Note ***BindDetails, BookletDetails, CollateDetails, FoldDetails, PostRenderingEnhanceDetails, PreRenderingEnhanceDetails,*** *and* ***StapleDetails*** *are details dictionaries that define how a product feature functions. The feature itself is enabled or disabled by a page device*

parameter. For example, ***Staple*** *is the key in the page device dictionary that enables stapling; the* ***StapleDetails*** *dictionary defines how the staple feature functions for that particular device. Details dictionaries are described in Section 4.6.2, "Details Dictionaries," on page 116. The* ***TrappingDetails*** *dictionary is described in Table 4.20 on page 119. The* ***ColorantDetails*** *dictionary is described in Table 4.21 on page 119.*

Table 4.17 *Entries in the page device dictionary*

Key	*Type*	*Semantics*
Bind	integer	Requests that the document be bound. The integer values are defined as follows:[a] 0 Do not bind. 1 Bind at device deactivation. 2 Bind at the end of the job. 3 Bind after each set. 4 Bind after each **showpage** or **copypage** operation.
BindDetails	dictionary	Describes product-specific details on how a document is to be bound.
Booklet	boolean	Requests that the document be stapled, trimmed, and folded into booklet form.
BookletDetails	dictionary	Describes product-specific details on how a document is to be stapled, trimmed, and folded.
CollateDetails	dictionary	Describes product-specific details on how a document is to be collated.
DeferredMediaSelection	boolean	A value of *false* indicates that media selection is to be performed as described in Section 4.11.4 of the *PostScript Language Reference Manual, Second Edition*. A value of *true* indicates that the product will be responsible for verifying the media requests. A value of *true* is usually selected if the product is handing off requests to another "entity" that will guarantee that the media requests will be satisfied at printing time. Thus, the name **DeferredMediaSelection** —the selection of the media is deferred until after this execution of the **setpagedevice** operator. *Note* *If the value of* ***DeferredMediaSelection*** *is true and* ***PageSize*** *is rejected by a product,* ***Policies PageSize*** *0 and 3 through 6 will result in a* ***configurationerror****. If under the same circumstances,* ***Policies PageSize*** *7 is used, whether or not media will be pulled from the current tray is product dependent.* When the value of **DeferredMediaSelection** transitions between *true* and *false*, some media parameters may need to be reinitialized. For example, each media selection model may have its own default page size that should be used if **PageSize** has not been explicitly specified.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
DeviceRenderingInfo	dictionary	Provides a location for individual OEMs or products to specify their own device rendering parameters. The only required parameter is **Type**, which is an integer. When the **setpagedevice** operator is executed, the entries in the **DeviceRenderingInfo** dictionary will be checked only if the **Type** value matches the value expected by the printing device. If the value does not match, the **setpagedevice** implementation will consult the **Policies** dictionary. If **Type** is not specified in the **DeviceRenderingInfo** dictionary, an **undefined** error will result, as described in Section 4.6.4 on page 120.
ExitJamRecovery	boolean	If the value is *true*, pages that jam in the exit path are reprinted. If the value is *false* (jam recovery disabled), performance might be improved because more overlapping of page processing is possible.
Fold	integer	Requests that the document be folded. The integer values are defined as follows:[a] 0 Do not fold. 1 Fold at device deactivation. 2 Fold at the end of the job. 3 Fold after each set. 4 Fold after each **showpage** or **copypage** operation.
FoldDetails	dictionary	Describes product-specific details on how a document is to be folded.
ImageShift	array	The array [*x y*] specifies the distance, in default user space, that each image on a page is to be shifted in the *x* direction for *x* units and in the *y* direction for *y* units. For page images that are to appear on a front side, the horizontal shift is to the right if $x > 0$, or to the left if $x < 0$. The vertical shift is to the top if $y > 0$, or to the bottom if $y < 0$. For page images that are to appear on a back side, the horizontal shift is to the left if $x > 0$, or to the right if $x < 0$. The vertical shift is to the bottom if $y > 0$, or to the top if $y < 0$.
InputAttributes	dictionary or *null*	Normally contains an entry for each input media source. The entry consists of an integer representing the input media slot and another associated dictionary. Some products may support the **InsertSheet** Boolean entry in the slot dictionaries that are subdictionaries of **InputAttributes**. The **InsertSheet** Boolean entry indicates whether or not the slot holds special insert sheet media. **InsertSheet** is used during the media matching process and is compared against the **setpagedevice** key **InsertSheet**. For further information, see "Matching Requests With Attributes," in Section 4.11.4, in the *PostScript Language Reference Manual, Second Edition.* The **PageSize** array within a slot dictionary is an unordered pair. That is, unlike the root level **PageSize** key, there is no difference between having an array [*x y*] with $x > y$ or $y > x$. A value of $x > y$ does not imply a landscape orientation.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
		If **InputAttributes** is *null*, the PostScript interpreter has no previous knowledge of the available media. Thus, when the operator **setpagedevice** is executed, the interpreter simply presents media selection requests to the device implementation, which is solely responsible for determining if they can be satisfied. This arrangement exists in products where actual printing of the output is deferred to some process not directly under the control of the PostScript interpreter.

InsertSheet	boolean	Specifies whether or not to select inserted media. The **setpagedevice** operator compares this **InsertSheet** value with the **InsertSheet** value, if any, in the **InputAttributes** entries for all the media that it considers. See Section 4.11.4, in the *PostScript Language Reference Manual, Second Edition*, remembering that **InsertSheet** is also an input media entry within the **InputAttributes** dictionary.
		
		If the **setpagedevice** operator is executed with **InsertSheet** equal to *true* and if an **InsertSheet** slot is selected, the imageable area is set to a zero-area region to ensure that nothing is imaged on the inserted sheet. That is, the inserted sheet is explicitly not imaged. The **InsertSheet** slot is such that the media coming from this slot is not sent through the fuser (a hot device in laser printers that fuses the toner to the paper) on its way to the output bin. Because media pulled from an **InsertSheet** slot does not go through the normal paper path, this slot can be used for any material that cannot tolerate being imaged to or sent through the fuser—photographic material, for example.
		
		The following example illustrates how to use **InsertSheet**:
		
		`%... PostScript language code for page n`
		`%... page n+1 is an inserted sheet`
		`%`
		`  save`
		`  <</InsertSheet true>> setpagedevice`
		`  % selects InsertSheet media`
		`  showpage`
		`  %send the InsertSheet media on to the`
		`  %output bin as page n+1`
		`  restore`
		`  %implicitly go back to using the regular media`
		`%`
		`%...PostScript language code for page n+2`

Jog	integer	Requests that output pages be “jogged”—that is, physically shifted in the output bin or diverted to another output bin—at specific times indicated by an integer code. Changes to the **Jog** parameter are implemented only after all pages that precede the current page description containing the requested changes have been delivered or aborted. Multiple jog requests with no intervening pages act as a single jog request. The request codes are:[a]

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*

		0 Do not jog pages at all.
		
		1 Jog at device deactivation. A jog request is triggered when the **Jog** parameter in the deactivating device has the value 1.
		
		The following example illustrates the effect of a **Jog** value of 1:
		
		`<< /Jog 0 >> setpagedevice`
		`showpage% page 1`
		`<< /Jog 1>> setpagedevice`
		`showpage% page 2`
		`<< /Jog 0>> setpagedevice`
		`showpage% page 3`
		
		Jogging occurs before page 3 is printed; pages 2 and 3 are separated in the output bin.
		
		2 Jog at the end of the job. Jogging between jobs is controlled by the value of **Jog** for the page device that is current between jobs. Thus, this feature can be turned on or off only by executing the **setpagedevice** operator as part of an unencapsulated job.
		
		The following example illustrates multiple **Jog** requests without intervening pages being printed. Assume that jogging between jobs is in effect and that **Jog** option 2 is set in an unencapsulated job:
		
		`showpage % Job, page 1`
		`^D % EOJ`
		`(Job 2 prints no pages)`
		`^D  % EOJ`
		`showpage % Job 3, page 2`
		`^D % EOJ`
		
		Although jog requests occur at the end of jobs 1 and 2, only a single **Jog** action will occur before page 2 is printed.
		
		3 Jog before each set and at device deactivation.

Laminate	boolean	If the value is *true*, the page is laminated. If the value is *false*, the page is not laminated. How a page is laminated is product specific.
LeadingEdge	integer or *null*	Specifies the edge of the media that will enter the engine or imager first and across which data will be imaged. The **setpagedevice** operator compares the value of **LeadingEdge** (unless its value is *null*) with the values of **LeadingEdge** in the **InputAttributes** entries for all defined media. See Section 4.11.4, in the *PostScript Language Reference Manual, Second Edition.* Values of **LeadingEdge** reflect positions relative to a canonical page, the orientation of which matches the default user space that would be selected by **PageSize** [*x, y*] **Orientation** 0, where *x* and *y* are the dimensions of the media, with $x < y$. Possible values are as follows:

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
		null No request for media orientation. 0 Short edge; top of canonical page. 1 Long edge; right side of canonical page. 2 Short edge; bottom of canonical page. 3 Long edge; left side of canonical page. When duplex printing is enabled, the canonical page orientation describes only the front side of the media. The orientation of the reverse side is defined by **Tumble** and is independent of the value of **LeadingEdge**.
ManualFeedTimeout	integer	The number of seconds the printer waits for a page to be fed manually before generating a time-out error. A zero value means no time-out (infinite wait).
Margins	array	If the device supports multiple resolutions (that is, different values of **HWResolution**), the margin values are interpreted according to some canonical default resolution and are scaled appropriately at other resolutions. This ensures that the values represent the same physical distance when the resolution is varied. The canonical default resolution is product dependent. For further information on **Margins**, see Table 4.11 in the *PostScript Language Reference Manual, Second Edition*.
MaxSeparations 	integer	(*Read-only*) Specifies the number of separations that are managed by the device, regardless of whether the device will be producing separations. Legal values: 1–250.
MediaClass	string or *null*	Specifies the class of media. The value of **MediaClass** is product specific. If **MediaClass** is not *null*, the **setpagedevice** operator compares this value with the **MediaClass** values, if any, in the **InputAttributes** entries for all the defined media. For further information, see Section 4.11.4 of the *PostScript Language Reference Manual, Second Edition*. If **MediaClass** is *null*, this key plays no role in the media selection process. See **MediaType**, which is an arbitrary string for media selection, in Table 4.10 in the *PostScript Language Reference Manual, Second Edition*. A product should support this key if the media selected for the output requires some action that may affect the output; for example, CRD selection or paper/transparency selection. See the **PageDeviceName** entry in this table for CRD selection.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
MediaPosition	integer or *null*	Indicates the slot that is to be used. The interpretation of the integer is product specific because slot numbers themselves are product specific. If media matching is in effect, **MediaPosition** does not override the matching process but its specification may produce unexpected results. For example, if **MediaPosition** can satisfy the page device request in a manner that is consistent with media matching, even if the selection is not optimum, then the slot represented by the **MediaPosition** value will be used. Thus, the selected slot is not necessarily the best match for the page device requests. If the requested slot is not installed or inserted, or if it cannot be chosen in the manner described above, **MediaPosition** will be rejected according to its policy. For example, consider a printer with two slots, slot 0 containing legal paper and slot 1 containing letter-size paper. Assume the policies of **PageSize** and **MediaPosition** are both 1. The command `<</PageSize [612 1008] /MediaPosition 1>> setpagedevice` would result in slot 1 (with letter-size paper) being selected, even though slot 0 is the perfect match for the request page size. This is because slot 1, as specified by **MediaPosition**, can satisfy the page device request (by ignoring the **PageSize** request), and so this slot is chosen. The same result holds if the policy of **PageSize** is 3. Once again, this is because slot 1 can satisfy the page device request (by scaling and adjusting the page). If the policy of **PageSize** is 0, then slot 1 cannot satisfy the page device request (the **setpagedevice** operator will raise a **configurationerror**). In this case, **MediaPosition** will be rejected. Since its policy is 1 in this example, its value is ignored and media matching will be retried on all slots. The result is that slot 0 (containing legal paper) is chosen. If media matching is not in effect (for example, **DeferredMediaSelection** is supported and the value is *true*), then it is a product decision as to how to resolve potential conflicts between the various media requests and the **MediaPosition** request. If **MediaPosition** is *null*, this key plays no role in the media selection process. If the **Policies PageSize** is 7 or if manual feed is used, **MediaPosition** is ignored. See Section 4.11.4 in the *PostScript Language Reference Manual, Second Edition*.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
OutputDevice	name or string	Selects an output device in environments in which the PostScript interpreter can generate output for multiple page devices. In some environments, **OutputDevice** selects from different types of output devices, such as a printer and a display screen, or a printer and an imagesetter. In other environments, it may select from similar devices, such as two or more imagesetters. The value of **OutputDevice** should match the name of an instance in the **OutputDevice** resource category. When the value of **OutputDevice** changes, the usual inheritance of any values not specified in the operand to the **setpagedevice** operator changes; all new values are generated in a product-specific manner. Also, the set of acceptable keys for the **setpagedevice** operator can change when the value of **OutputDevice** changes because different devices have different features that can be controlled or queried.
OutputPage	boolean	If the value is *true*, processing is normal. If the value is *false*, no pages are actually printed but all other processing, including rasterizing to a frame buffer, is done as if the page were to be printed. Thus, when the value of **OutputPage** is *false*, the time to process a page includes everything except the time spent waiting for the marking engine. Furthermore, rasterization occurs synchronously with execution of the **showpage** operator, rather than being overlapped with processing of subsequent pages. This facilitates measuring the complete cost of page execution.
PageDeviceName	string, name, or *null*	Used by the **findcolorrendering** operator. **PageDeviceName** allows a specific device setup to be named. See the section “Selecting CRDs by Rendering Intent: The findcolorrendering Operator” on page 162 for details on the **findcolorrendering** operator. See the section “Modifying the CRD Selection Process” on page 162 for information on how **GetPageDeviceName** uses the value of **PageDeviceName**. See also **MediaClass** (in this table), which can affect CRD selection.
PageOffset	array	Contains two numbers that are used to relocate the page image on the media. Relocation is *x* units in the device *x* coordinate direction and *y* units in the device *y* coordinate direction. *x* and *y* are always expressed in units of 1/72 of an inch. This positioning is typically accomplished by altering the CTM. On some products, however, this positioning can be accomplished by device-dependent means that are independent of the graphics state (particularly the CTM). The **PageOffset** key is typically found on imagesetters and is used to control where the image is to appear on the media.
PageSize	array	See Table 4.10 on page 232 in the *PostScript Language Reference Manual, Second Edition,* for a description of the **PageSize** entry. The following discussion is intended to clarify how the **PageSize** matching tolerance is used during media matching.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
		When roll media is used, the media is considered to be a match (with respect to **PageSize**) when the requested media size is less than or equal to the actual size of the physical media plus 5 additional default user space units. Hence, even if a match succeeds, the requested dimensions may in fact be larger than the actual dimensions of the physical media. For non-roll media, the requested size must be between the actual size minus 5 default user space units and the actual size plus 5 units in order to be considered a match (in other words, the absolute difference between the actual and request sizes is no more than 5 units). When non-roll media is matched, the dimensions used are the actual dimensions of the actual media selected. Failure to match any available media within this tolerance triggers the **PageSize** recovery policy in either case.
PostRenderingEnhance	boolean	If the value is *true*, product-specific image enhancements are enabled. These enhancements are made after the page is rasterized in memory.
PostRenderingEnhanceDetails	dictionary	Describes product-specific details related to postrendering image enhancement.
PreRenderingEnhance	boolean	If the value is *true*, product-specific image enhancements are enabled. These enhancements are made before the image is rasterized in memory.
PreRenderingEnhanceDetails	dictionary	Describes product-specific details related to prerendering image enhancement.
ProcessColorModel	name or string	Specifies the model used for rendering process colors in the device. It affects rendering for all color spaces, with the exception of **Separation** and **DeviceN** color spaces that actually produce colorants of specified separations. It does not affect the interpretation of color values in any color space; it controls only the rendering method. Legal values are /DeviceGray, /DeviceRGB, /DeviceCMYK, /DeviceCMY, /DeviceRGBK, and /DeviceN. For example, /DeviceRGB specifies that the process colorants are named /Red, /Green, and /Blue; /DeviceCMYK specifies /Cyan, /Magenta, /Yellow, and /Black. /DeviceN has no predefined process colorants; see **DeviceN** in Table 3.4 on page 39. These are the names used to select halftones in a type 5 halftone dictionary and to control the production of separations in **SeparationColorNames** and **SeparationOrder**. For information on using a **ProcessColorModel** with a value of /DeviceN, see Section 6.4, “Extensions to Process Color Models,” on page 175.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
		Each of the **ProcessColorModel** values implies a specific native color space for the device. The native color space is the PostScript language device color space into which user-specified colors are converted, if necessary. These conversions are product specific for the **DeviceN** color model. See Section 6.2, "Conversions Among Device Color Spaces," on page 303 in the *PostScript Language Reference Manual, Second Edition*. • /DeviceGray, /DeviceRGB, and /DeviceCMYK select the correspondingly named native device color space. • /DeviceCMY and /DeviceRGBK both select /DeviceRGB as the native device color space, but they cause the device to render the /DeviceRGB color values in special ways. For /DeviceCMY, the device renders the RGB colors using the complementary subtractive colors. For /DeviceRGBK, the device uses a separate rendering method for RGB color values that represent pure shades of gray.
	Note	*If a device cannot resolve a combination of **ProcessColorModel** /DeviceN and **SeparationColorNames** requests, it may revert to a previous state, or (for some capable printing systems) to some new state, where it can convert all color spaces. All color spaces will always be printed; however, the colorants used may differ from what was requested.*
RollFedMedia	boolean	If the value of **RollFedMedia** is *true*, the media is roll fed. Note that the matching criteria for the **PageSize** parameter differ for roll-fed and non-roll-fed media. See the **PageSize** page device parameter in this table.
SeparationColorNames	array	Specifies all colorants that are valid values for **Separation** and **DeviceN** color spaces. Process colors are implicitly included in this array. This array can contain either names or strings, for example, [/Pink /Green] or [(Pink) (Green)] or [/Pink (Green)]. Duplicate entries are ignored and the order of entries is not significant. If the name used in a [/Separation name...] or [/DeviceN names...] **setcolorspace** operation is included in this array, that colorant will be used rather than *alternativeSpace*. Any other name (including any process color) will be mapped to one or more of the named colors through the *alternativeSpace* and *tintTransform* parameters of the **setcolorspace** operator. This is described in Section 4.8.4 of the *PostScript Language Reference Manual, Second Edition*. The names of the colorants of the native color space are included implicitly, regardless of the contents of the array. Thus:

Table 4.17 *Entries in the page device dictionary* *(continued)*

Key	*Type*	*Semantics*
		• For /DeviceCMY, the empty array [ ] is equivalent to [/Cyan /Magenta /Yellow] • For /DeviceCMYK, the empty array [ ] is equivalent to [/Cyan /Magenta /Yellow /Black] • For /DeviceRGB, the empty array [ ] is equivalent to [/Red /Green /Blue] • For /DeviceRGBK, the empty array [ ] is equivalent to [/Red /Green /Blue /Black] • For /DeviceGray, the empty array [ ] is equivalent to [/Gray]  • For /DeviceN, the device colorants must be specified explicitly in the **SeparationColorNames** array. See also Section 6.4, “Extensions to Process Color Models,” on page 175.
SeparationOrder	array	(*Optional*) Specifies the colorants to be produced by the device. This array can contain either names or strings—for example, [/Pink /Green] or [(Pink) (Green)] or [/Pink (Green)]. Legal values are the current process color names and any additional names specified by **SeparationColorNames**. The semantics of this parameter differs depending on the value of **Separations**. If **Separations** is *true*, a separation will be produced for each occurrence of a name; multiple occurrences will produce multiple separations. Only those colorants named in **SeparationOrder** will be output although all named separation colorants are defined (as opposed to reverting to the *alternativeSpace*). Each separation will be produced in the order specified by this array. If **Separations** is *false*, **SeparationOrder** specifies which colorants are to be applied. The order of entries is not considered when producing a composite page. An empty array [ ] requests that all colors of the native color space (i.e. its **ProcessColorModel**, as well as all colorants requested by **SeparationColorNames**) be produced in an unspecified order.
Separations	boolean	When the value of **Separations** is *true*, a device should produce each page as an individual separation—one page for each named colorant. When the value is *false*, the device should produce each page as a single composite page with all colors, if any, combined on the same page.

Table 4.17 *Entries in the page device dictionary (continued)*

Key	*Type*	*Semantics*
Signature	boolean	If the value is *true*, the job will be "signatured." That is, pages of a document will be arranged so that, when folded, the pages will be in the correct order. How this process is performed is device dependent. On some devices, the engine provides the resources (memory, disk space) to generate the signature for the job. On other devices, the interpreter may have to reorder the virtual pages in order to deliver the pages to the engine in the correct order. In the latter case, a **Signature** value of *true* implies that the interpreter must store the results of executing the page description for multiple pages in order to deliver the pages in the correct order. This use of **Signature** is supported by relatively few products and is subject to resource limits in products that do support it.
SlipSheet	integer	Requests that slip sheets be inserted (slip sheet media selection is product specific). There is no way to render a slip sheet (the imageable area is set to a zero-area region); the engine is simply told when to insert one. A slip sheet can be a colored sheet of paper, for example, whose sole purpose is to visually separate multiple copies. A slip sheet may go through the fuser or a separate paper path. Compare with the description of **InsertSheet** above. Slip sheets will be inserted at specific times indicated by an integer code:[a] 0 Do not insert slip sheets. 1 Insert slip sheet at device deactivation. 2 Insert slip sheet at the end of the job. 3 Insert slip sheet at the end of the set. 4 Insert slip sheet after each **showpage** or **copypage** operation.
Staple	integer	Requests that the job be stapled. The job will be stapled at a specific time indicated by an integer code:[a] 0 Do not staple. 1 Staple at device deactivation. 2 Staple at the end of the job. 3 Staple after each set. 4 Staple after each **showpage** or **copypage** operation.
StapleDetails	dictionary	Describes product-specific details related to how a document is to be stapled.
Trapping	boolean	If the value of **Trapping** is *true*, pages are trapped, and **TrappingDetails** dictionary must be present. If the value is *false* or the device has only one color plane, pages are not trapped. The current trapping method applies to devices with multiple color planes. Pages are trapped within the defined trapping zones according to the trapping parameters for each trapping zone.

Table 4.17 *Entries in the page device dictionary* *(continued)*

Key	*Type*	*Semantics*
TrappingDetails 	dictionary	Describes details related to trapping. The **TrappingDetails** dictionary holds a set of page device-level trapping parameters and a **ColorantDetails** dictionary. The contents of the **TrappingDetails** dictionary are given in Table 4.20 on page 119. The contents of the **ColorantDetails** dictionary are given in Table 4.21 on page 119.
TraySwitch	boolean	If the value is *true*, automatic tray switching is provided. This option is offered by some devices with multiple input trays. When one input tray runs out of media, another tray with the same type of media will be used automatically, without alerting the user that the printer is out of media. Tray switching can take place between any of a printer's trays that have the same characteristics (for example, **PageSize** and **MediaColor**) as the selected tray. These alternative trays and their priorities are a product-dependent feature that does not depend on the **Priority** array; however, the **Priority** array may be used for this purpose.
Trim	integer	Requests that the job be trimmed. The job will be trimmed at a specific time indicated by an integer code:[a] 0 Do not trim. 1 Trim at device deactivation. 2 Trim at the end of the job. 3 Trim after each set. 4 Trim after each **showpage** or **copypage** operation.
UseCIEColor	boolean	If *true*, enables the substitution of CIE-based color spaces for device color spaces (see Section 6.1, "UseCIEColor," on page 158). If *false*, no such substitution is performed.

a. See the description of the **Collate** and **Jog** keys in Section 4.11.3 in the *PostScript Language Reference Manual, Second Edition*, for a definition of job, set, and device deactivation.

4.6.2 Details Dictionaries

Some page device features have many variables that determine precisely how the feature functions; these variables may be quite different for different products. Features of this type are enabled or disabled by an entry in the page device dictionary, whereas the exact way in which the feature functions is determined by entries in the details dictionary for that page device entry. This allows an application that is not knowledgeable about the details of a feature to enable and disable it, while more sophisticated utilities can be used to configure the details.

An example of this is the stapling feature. Many applications will want to either enable or disable stapling with the assumption that the number, location, and orientation of the staples has been configured correctly. The

nature of the configuration will be dependent on the printing device. For example, on some engines it may be possible to specify an arbitrary staple location on the sheet, while on others, staples may be placed only in one of the four corners.

Primary page device entries for such features have values that are either Boolean values or integers. If the value is a Boolean, then the feature is enabled if the value is *true* and disabled if the value is *false*. If the value is an integer, the feature is disabled if the value is zero. A non-zero value enables the feature in different ways that are consistent across all products. For example, the binding feature can be enabled for binding at the end of device deactivation, at the end of a job, at the end of each set, or after each **showpage** or **copypage** operation.

A consistent naming convention is used for details dictionaries. The name of the dictionary is the name of the primary key with **"Details**" appended. For example, if the **Staple** feature is present and has a details dictionary, this dictionary is named **StapleDetails**.

A details dictionary will be present for a given feature on a given product only if additional information beyond that of the primary entry is needed to control that feature. For example, a product supporting a postrendering enhancement feature that can only be enabled or disabled but has no further control will not have a details dictionary for this feature. However, a printer with more configurable postrendering enhancement would have a details dictionary. Applications that simply enable and disable a feature should never reference a details dictionary. Applications intended to control a details dictionary should never assume that one is present unless the exact nature of the printing device on which they are executing is known.

During the execution of the **setpagedevice** operator, the entries in any details dictionary are checked for syntactical correctness only if the **Type** value matches what is expected by the printing device. If the **Type** key is not specified in the details dictionary, an **undefined** error will result. If the **Type** value is a number unknown to the printing device, policy is consulted. If the **Type** value is known, the validity of the values within the details dictionary is checked only if the feature is to be enabled for the page device in effect as a result of a **setpagedevice** operation. As with all page device entries, syntactically incorrect settings generate PostScript errors (for example, **typecheck**), and invalid values result in policy being consulted.

Type

Every details dictionary has a **Type** key whose integer value determines how the details dictionary entries affect the feature. For example, if two different products have details dictionaries for the same feature and the **Type** entry is the same for each, then the dictionaries will have exactly the same named

entries and the syntax and semantics of each entry will be the same. This allows an application, based solely on the value of the **Type** entry, to change entries in a details dictionary for a feature.

If details dictionary entries are being set, whether the new dictionary overwrites the current dictionary or is merged with it is determined by the **Type** entry. The criteria for merging versus overwriting are product dependent. Details dictionaries and their associated **Type** entries are registered with Adobe.

Table 4.18 and Table 4.19 describe some typical entries in the **CollateDetails** dictionary and the **PostRenderingEnhanceDetails** dictionary, respectively. A complete description of available keys for a given product is provided by the OEM.

Table 4.18 *Typical entries in a CollateDetails dictionary*

Key	*Type*	*Semantics*
ProofSet	boolean	If the value is *true*, the initial copy of a collated job will be printed before remaining copies are output. The printer will remain in a "hold" mode until some product-specific action is taken, such as aborting or printing the remaining copies.
Type	integer	Identifies the type of **CollateDetails** dictionary. A given type of **CollateDetails** dictionary can contain a specific set of keys whose values are interpreted in a particular way. Different dictionary types are independent, and a given product supports only certain types.

Table 4.19 *Typical entries in a PostRenderingEnhanceDetails dictionary*

Key	*Type*	*Semantics*
REValue	integer	Provides a range of values for rendering enhancement. A value of zero reflects the least (or no) amount of enhancement.
PrintQuality	integer	Provides a range of output quality levels as a positive integer. A value of zero reflects the lowest or draft quality.
Type	integer	Identifies the type of **PostRenderingEnhanceDetails** dictionary. A given type of **PostRenderingEnhanceDetails** dictionary can contain a specific set of keys whose values are interpreted in a particular way. Different dictionary types are independent, and a given product supports only certain types.

4.6.3 TrappingDetails and ColorantDetails Dictionaries



Table 4.20 and Table 4.21 give the entries in the **TrappingDetails** dictionary and the **ColorantDetails** dictionary, which is a subdictionary of the **TrappingDetails** dictionary.

Table 4.20 *Entries in a TrappingDetails dictionary*

Key	*Type*	*Semantics*
ColorantDetails	dictionary	(*Optional*) Defines trapping-related parameter values for individual device colorants.
TrappingOrder	array	The trapping engine will trap colorants as if they are laid down on the media in the order specified in **TrappingOrder**. The colorant order may affect which colors to spread, especially if opaque inks are used. This order can be different from the actual printing order on the press or the **SeparationOrder** entry in **setpagedevice**. The first entry in the array of colorant names identifies the first color laid down on the medium. After all the colors listed in the array have been laid down, or if the array is empty, any additional colors are laid down in the order defined by the **ProcessColorModel** and **SeparationColorNames** entries in **setpagedevice**. Colors named in the array that are not part of **ProcessColorModel** and not found in **SeparationColorNames** are ignored.
Type	integer	(*Required*) Identifies the trapping method. **Type** determines the syntax and semantics of the **TrappingDetails** dictionary entries, the **Trapping ProcSet**, and the trap parameter set entries, and any other information that is specific for each trapping method. **Type** has at least four decimal digits. The three right-most digits identify the version of the trapping method. The other digits identify distinctly different trapping methods. Legal value: 1001

The **ColorantDetails** dictionary must have one entry for each process color and for each entry in the **SeparationColorNames** array. The contents of the **ColorantDetails** dictionary are given in Table 4.21.

Table 4.21 *Entries in a ColorantDetails dictionary*

Key	*Type*	*Semantics*
ColorantSetName	name, string, or *null*	(*Optional*) User-defined name or string that identifies the parameter set to the user. This value is not used for trapping. It is recommended that this entry be used only when the dictionary reflects the contents of a resource with this name.

Table 4.21 *Entries in a ColorantDetails dictionary (continued)*

Key	*Type*	*Semantics*
A device colorant name	dictionary	(*Required/Optional*) This is a variable key name that defines parameters for a device colorant. There must be one entry for each device colorant; additional entries are optional. The contents of a device colorant name dictionary are listed in Table 4.22.

Table 4.22 *Entries in a device colorant name dictionary*

Key	*Type*	*Semantics*	
ColorantName	name, string, or *null*	(*Optional*) User-defined name of the ink or colorant—vendor name or part number, for example. This value is not used for trapping support.	
ColorantType	name	(*Required*) Controls how trapping is performed for this ink.	
		/Normal	Traps marks made with this ink, covered by this ink, and placed on top of this ink.
		/Transparent	Does not trap marks made with this ink. The trapping engine need not generate a color plane for this colorant. /Transparent can be used for varnish.
		/Opaque	Does not trap marks covered by this ink. /Opaque can be used for metallic inks.
		/OpaqueIgnore	Does not trap marks made with this ink and marks covered by this ink. /OpaqueIgnore can be used for metallic inks.
NeutralDensity	number	(*Required*) Gives the neutral density of this ink or colorant. Range 0.001–10.	

4.6.4 DeviceRenderingInfo Dictionaries

Some products must be able to specify their own device rendering parameters, and these parameters may be very different for different products. These device rendering parameters are applied as needed for rendering to occur; they are not disabled by a primary page device entry as are page device features. See Section 4.6.2, “Details Dictionaries,” on page 116.

During the execution of the **setpagedevice** operator, entries in the **DeviceRenderingInfo** dictionary are checked for syntactical correctness only if the **Type** value matches what is expected by the printing device. If **Type** is not specified in the **DeviceRenderingInfo** dictionary, an **undefined** error will result. If the **Type** value is unknown to the printing device, policy is consulted. As with all page device entries, syntactically incorrect settings will

generate a PostScript error (for example, **typecheck**) and invalid values will result in policy being consulted.

Type

Every **DeviceRenderingInfo** dictionary has a **Type** key whose integer value determines how the dictionary entries affect the rendering capability. For example, if two different products have **DeviceRenderingInfo** dictionaries with the same **Type**, then the dictionaries have exactly the same named entries and the syntax and semantics of each entry will be the same. This allows an application, based solely on the value of the **Type** entry, to change entries in a **DeviceRenderingInfo** dictionary for the rendering capability.

Table 4.23 describes some typical **DeviceRenderingInfo** keys being used in products. A complete description of available keys for a given product is provided by the OEM.

Table 4.23 *Typical entries in a DeviceRenderingInfo dictionary*

Key	*Type*	*Semantics*
ValuesPerColorComponent		
	integer	Specifies the number of values each color component may have, or in the monochrome case, the number of gray levels.
AntiAlias	boolean	When *true*, enables the initializing feature if that feature is available. Currently, for initializing to be enabled, **ValuesPerColorComponent** must be greater than 1 and ColorBurst™ must be present.
Type	integer	Identifies the type of **DeviceRenderingInfo** dictionary. A given type of **DeviceRenderingInfo** dictionary will contain a specific set of keys, the values of which are interpreted in a particular way. Different dictionary types are independent, and a given product supports only specific types.

4.6.5 Errors Generated by Page Device Parameters

If a product does not support a particular feature, then the PostScript interpreter invokes policy, as defined in the **Policies** dictionary, for that feature; the type of the value is not checked. For most products, the default policy for unknown features is to ignore them. Alternatively, a **configurationerror** is generated, and the job is rejected.

In addition to generating a **configurationerror**, the **setpagedevice** operator can also generate a **typecheck**, **rangecheck**, **undefined**, **limitcheck**, or **invalidaccess** error, as described in the following sections.

typecheck Errors

A **typecheck** error is generated under the following conditions:

- The type of the value for a feature or for a component value within a compound value is incorrect. Each of the following examples would generate a **typecheck** error:

```
<< /BeginPage 4 >> setpagedevice
<< /Margins [0 true] >> setpagedevice
<< /InputAttributes << 0 23 >>  >> setpagedevice.
```

- A literal array is given for a value that should be a procedure. Note that an executable array is acceptable if an array value is expected, and packed arrays are acceptable wherever an array is acceptable. The first two examples below would generate a **typecheck** error; the third is correct and would not generate an error:

```
<< /Install [2 3 4] >> setpagedevice.
<< /Policies <</PolicyReport [5 6 7] >>  >> setpagedevice
<< /PageSize {612 792} >> setpagedevice
```

- The operand to the **setpagedevice** operator is not a dictionary. The following example would generate a **typecheck** error:

```
true setpagedevice
```

rangecheck Errors

A **rangecheck** error is generated under the following conditions:

- An array of the wrong length is given as the value for a feature or for a component of a value within a compound value. Both of the following examples would generate a **rangecheck** error:

```
<< /HWResolution [300] >> setpagedevice

<< /InputAttributes << 0 << /PageSize [600 700 800] >>  >>  >>
setpagedevice
```

- A value of the correct type but beyond the range of acceptable values is given as the value for a feature or for a component of a value within a compound value. Both of the following examples would generate a **rangecheck** error:

```
<< /PreRenderingEnhanceDetails << /Type –1 >>  >>
setpagedevice

<< /Jog 10 >> setpagedevice
```

undefined Errors

An **undefined** error is generated under the following conditions:

- The **Type** key is not specified in a details dictionary.

invalidaccess Errors

An **invalidaccess** error is generated under the following conditions:

- A string, array, or dictionary value is given for a feature or a component value within a compound value, and the access for the value is more restrictive than *read-only.* If the value is a procedure, the value can be *execute-only.* In the following examples, the first two would generate **invalidaccess** errors, the third is correct would not generate an error:

```
<< /MediaColor (blue) noaccess >> setpagedevice
<< /PageSize {612 792} executeonly >> setpagedevice
<< /BeginPage {pop} executeonly >> setpagedevice
```

- The operand to the **setpagedevice** operator is a dictionary, the access for which is more restrictive than *read-only.* The following example would generate an **invalidaccess** error:

```
<< /PageSize [612 792] >> noaccess setpagedevice
```

limitcheck Errors

A **limitcheck** error is generated if the storage area is insufficient during invocation of the **setpagedevice** operator.

4.6.6 Input Media Selection: Envelope Orientation in User Space

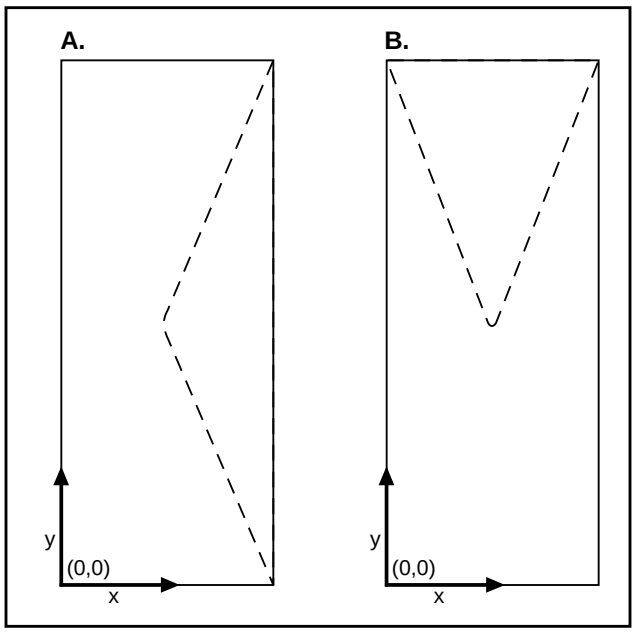
This section describes how default user space is oriented relative to the flap on an envelope. This discussion assumes that the **Install** procedure does not alter the default transformation matrix.

If the **PageSize** value is portrait ([*width height*] with *width* < *height*), then default user space is set up so that the origin is on the opposite edge of the envelope from the flap and in the diagonally opposite corner from the return address (on a U.S. business envelope). The default user space is set up this way, regardless of how envelopes are fed into the printer on a particular product.

Figure 4.12 illustrates two envelopes: one with its flap along the long edge of the envelope, and one with its flap along the short edge of the envelope. The dashed line indicates that the flap is on the side of the envelope facing down.

If the flap is along the long edge of the envelope, then default user space for a portrait **PageSize** is set up as in Figure 4.12A. If the flap is along the short edge of the envelope, then the default user space for a portrait **PageSize** is set up as in Figure 4.12B.

Figure 4.12 *Default user space orientation for an envelope with a PageSize value of portrait*



For landscape **PageSize** values ([*width height*] with *width* > *height*), the orientation of default user space is defined relative to the orientation for portrait **PageSize** values. This relationship is described in Table 4.10 in the *PostScript Language Reference Manual, Second Edition*.

4.6.7 New Entries in the Policies Dictionary

When a print job makes a request that a device cannot satisfy, the **setpagedevice** operator consults the **Policies** dictionary to determine how to deal with the request, as described in Section 4.11.5 in the *PostScript Language Reference Manual, Second Edition*. Table 4.24 describes changes to the **Policies** dictionary since the publication of the *PostScript Language Reference Manual, Second Edition.*

Table 4.24 *Entries in a Policies dictionary*

Key	*Type*	*Semantics*
PageSize	integer	Specifies the recovery policy to use when the **PageSize** entry cannot be matched (within a tolerance of 5 units) with any available media. The policy values 0 to 6 are described in Table 4.14 in Section 4.11.5 in the *PostScript Language Reference Manual, Second Edition*. The policy value 7 is defined below:

Table 4.24 *Entries in a Policies dictionary (continued)*

Key	*Type*	*Semantics*
		7 Disables media selection altogether and imposes the requested **PageSize** value on the previously selected medium without adjusting it in any way. That is, the page device is set up as if the selected medium were of the requested size, ignoring the actual size of the medium. However, if the requested **PageSize** is not within 5 units of any one of the page sizes supported by the product, the product rejects **Policies PageSize** 7 and a **configurationerror** results. How the page image is positioned on the medium is product dependent and unpredictable. A **Policies PageSize** of 7 takes effect during every execution of the **setpagedevice** operator. This is unlike all other policies, which take effect only if a request cannot be satisfied. This policy exists solely for use in the emulations of certain LanguageLevel 1 compatibility operators that perform media selection and page device setup separately. **Policies PageSize** 7 should never be used beyond LanguageLevel 1. Its semantics violate later PostScript page device models, and documents that use it are not portable.

Note *In Table 4.14, page 246 of the PostScript Language Reference Manual, Second Edition, under the entries for both **PolicyNotFound** and **PageSize**, the description for integer code 1 should be changed to read, "Do not consider this feature in media selection."*

CHAPTER 5

Additions and Modifications to Fonts

This chapter supplements Chapter 5 in the *PostScript Language Reference Manual, Second Edition.* It provides information about Type 32 and Type 42 font dictionaries, CID fonts, CMap resources, composite fonts, CFF fonts (Type 2), and Chameleon® fonts (Type 14).

Overview of This Chapter

5.1 Multiple Master Font Dictionary 128

Entries specific to a multiple master font dictionary (Table 5.1) 129

5.2 Additional Base Font Types 129

5.2.1 Bitmap Fonts: Font Type 32 129

Entries in a Type 32 font dictionary (Table 5.2) 130

Operators for Type 32 Fonts 131

Behavior of Painting Operators with Type 32 Fonts 131

5.2.2 TrueType Fonts: Font Type 42 131

Additional entries in a Type 42 font dictionary (Table 5.3) 132

5.2.3 CFF and Chameleon Fonts: FontType 2 and FontType 14 132

FontSet Resource 133

Integration of CFF and Chameleon Fonts into the PostScript Language 133

5.3 CID-Keyed Fonts 135

Terms Used in This Section 135

5.3.1 CID Font Resources 136

CIDFontType and FontType values (Table 5.4) 137

Entries in all types of CIDFont dictionaries (Table 5.5) 138

CIDFontType 0 CID Font File 139

Additional entries in a CIDFontType 0 CIDFont dictionary (Table 5.6) 140

Entries in a dictionary in the FDArray (Table 5.7) 142

Entries in the Private Dictionary of a CIDFontType 0 CID Font 142

Additional entries in a Private dictionary of a CIDFontType 0 CIDFont (Table 5.8) 142

CIDFontType 1 CID Font 143

Additional entries in a CIDFontType 1 CIDFont dictionary (Table 5.9) 143

CIDFontType 2 CID Font 143
Additional entries in a CIDFontType 2 CID fonts (Table 5.10) 144
5.3.2 CIDSystemInfo Dictionary Entries 145
Entries in a CIDSystemInfo dictionary (Table 5.11) 145
5.3.3 Subsets and Incremental Definition of Glyph Procedures 145
Semantics of GlyphDirectory for CIDFontType 0 146
Semantics of GlyphDirectory for CIDFontType 2 146
Memory Use 147
save and restore 148
5.3.4 CMap Resources 148
Entries in a CMap dictionary (Table 5.12) 148
5.3.5 CIDInit ProcSet 149
General Operators 149
Operators that Use Character Names or Character Codes as Selectors 150
Operators that Use CIDs as Selectors 150
notdef Operators 150
5.3.6 CMap Example 150

5.4 Composite Font Extension 153
5.4.1 Type 0 Dictionary Entries 153
Entry specific to a Type 0 (composite) font dictionary (Table 5.13) 153
5.4.2 Character Mapping 153
New FMapType mapping algorithm (Table 5.14) 154
5.4.3 Undefined (notdef) Character Handling 154
5.4.4 show String Parsing 155
5.4.5 Nesting 155

5.1 Multiple Master Font Dictionary

The multiple master font format is an extension of the Type 1 font format that allows the generation of a wide variety of typeface styles from a single font format. The entries in a multiple master font dictionary are the entries in the Type 1 font dictionary plus those listed in Table 5.1. (For details of the construction and uses of multiple master fonts, refer to the *Adobe Type 1 Font Format Supplement* listed in Appendix G, “Bibliography,” on page 395.)

Table 5.1 *Entries specific to a multiple master font dictionary*

Key	*Type*	*Semantics*
WeightVector	array	(*Required*) Specifies the contribution of each master design to the current font instance. The array contains *k* elements, where *k* is the number of master designs. The elements must add up to 1.0 (within a tolerance of 0.001). The values in this array are used for calculating the weighted interpolation. The multiple master font program can have a default for setting the **WeightVector** entry. The recommended default is the normal Roman design for the typeface.
Blend	dictionary	(*Required*) Contains an entry for the interpolated values for the **FontBBox** key, plus definitions for the **Private** and **FontInfo** subdictionaries.
$Blend	procedure	(*Required*) Calculates the weighted average of values from the master designs using the values specified by the **WeightVector** key.

5.2 Additional Base Font Types

A number of font dictionaries types have been added to the PostScript language. These font dictionaries types are described in the following sections. For further information about fonts in general, see Chapter 5 in the *PostScript Language Reference Manual, Second Edition*.

5.2.1 Bitmap Fonts: Font Type 32

A Type 32 font dictionary is a means for managing device resolution bitmap characters that have been rendered on the host computer prior to transmission to the PostScript interpreter. The host application or driver downloads the bitmaps into the PostScript font cache and manages those characters directly.

This method is a space- and time-efficient alternative to the traditional method of defining fonts as Type 3, where the Type 3 **BuildChar** procedure renders the bitmaps using the **imagemask** operator.

Note *For correct results, the host application or driver must know certain device-dependent details of the target device, including the resolution and orientation of device space and the capacity of the font cache. Therefore, Type 32 fonts should be used only for attached printer systems that are under direct control of host software. They should not be used in a PostScript language document file that is intended to be portable.*

The primary characteristics of Type 32 fonts are as follows:

- They print with bitmaps of characters, and only glyphs found in the document are rendered and incrementally downloaded to the printer.

- The host performs rendering based on the resolution of the printer.
- They occupy storage in the cache only. They do not use VM. (In contrast, Type 3 fonts use storage in the cache and in VM.)
- They are character identifier (CID) fonts. For further information on CID fonts, see Section 5.3.1, “CID Font Resources,” on page 136.
- Though they are host generated, they are compatible with all PostScript interpreters that support Type 32 fonts, despite differences in printer resolution and/or printer orientation, as is true with all Adobe fonts and PostScript files. (Support for Type 32 fonts can be determined by querying the **FontType** resource category.) Type 32 fonts can also be rotated and/or scaled.

Note *Rotation and scaling degrade the printed image quality of Type 32 bitmap fonts in the same way that they degrade the quality of Type 3 fonts.*

Some restrictions apply to Type 32 fonts. In general, embedded EPS files and other PostScript code inserted by pass-through in the driver cannot see Type 32 fonts defined in the enclosing document. (This restriction is directly due to the limitations of incremental downloading.) Additionally, transformations that require accessing outline definitions cannot be performed with a Type 32 font. This means that operations such as stroking the outline, shadowing, and clipping with the **charpath** operator must be performed in the driver or application prior to downloading.

Table 5.2 gives the entries in a Type 32 font dictionary.

Note *Other entries not listed in the table below that are specific to Type 1 fonts (for example, the **Metrics**, **Metrics2**, **CDevProc**, and **PaintType** keys) are not applicable to Type 32 fonts and are ignored.*

Table 5.2 *Entries in a Type 32 font dictionary*

Key	*Type*	*Semantics*
CIDFontType	integer	(*Required*) Must be 4.
FontMatrix	array	(*Required*) Transforms the character coordinate system to the user coordinate system. For further information, see Table 5.1 on page 266 in the *PostScript Language Reference Manual, Second Edition*. In the case of a Type 32 font, the **FontMatrix** should be the inverse of the transformation from default user space to device space of the device for which the bitmap is designed to be used. Initially, the translation components of this matrix should be zero. Positioning of the individual glyph bitmaps is accomplished via metric information provided while defining each glyph. For further information, see the definition of the **addglyph** operator in Chapter 8, “Additions and Modifications to the Operators,” on page 193.

Table 5.2 *Entries in a Type 32 font dictionary (continued)*

Key	*Type*	*Semantics*
FontBBox	array	Font bounding box. For further information on **FontBBox**, see Table 5.2 on page 266 in the *PostScript Language Reference Manual, Second Edition.*
FontType	integer	(*Optional*; *inserted by the* ***defineresource*** *operator*) Must be 32.

Operators for Type 32 Fonts

Three operators are used for incremental downloading and management of glyphs in Type 32 fonts. These operators are defined in the **BitmapFontInit ProcSet** rather than in **systemdict**.

The **addglyph** operator incrementally downloads Type 32 fonts. The operator **removeall** removes all glyphs defined for Type 32 fonts. The operator **removeglyphs** removes specified glyphs**.** For details of these operator, see Chapter 8, “Additions and Modifications to the Operators,” on page 193.

Behavior of Painting Operators with Type 32 Fonts

At the time the **show** operator is executed, the font machinery checks to see if the concatenation of the **FontMatrix** entry (including any prior **makefont** transformation) and of the CTM is the identity; any translation is disregarded.

If the concatenation of the **FontMatrix** entry and the CTM is the identity, the bitmap that was defined by the **addglyph** operation is painted directly on the current page. If not, the bitmap is used as an image source and is painted on the current page with the suitable transformation, as if by the **imagemask** operator.

The **charpath** operator produces an empty path.

5.2.2 TrueType Fonts: Font Type 42

PostScript interpreters can optionally include support for Type 42 fonts. (Support for Type 32 fonts can be determined by querying the **FontType** resource category.) The Type 42 font format is a TrueType™ font with a PostScript language wrapper to make it conform to the PostScript language font model. A printer driver can determine if a product supports Type 42 fonts by using the PPD file to extract the appropriate query. The query looks at the resource category **FontType** to determine if Type 42 is supported. **FontType** instances are listed in Table 3.7 on page 43.

The Type 42 font format is described in the *Type 42 Font Format Specification* listed in Appendix G, , “Bibliography,” on page 395.

Table 5.3 *Additional entries in a Type 42 font dictionary*

Key	*Type*	*Semantics*
FontMatrix	array	*(Required)* Transforms the glyph coordinate system into the user coordinate system. Type 42 fonts, unlike Type 1 fonts, are usually defined in terms of an identity transform, so the value of **FontMatrix** should be [1 0 0 1 0 0].
CharStrings	dictionary	*(Required)* Associates glyph names with glyph descriptions. If an entry's value is an integer, it is used as an index into the TrueType loca table, which contains the byte offsets of glyph definitions in the TrueType glyf table. If the value is a procedure (executable array or packed array), it is interpreted as described in Section 5.6.3 of the *PostScript Language Reference Manual, Second Edition*. This dictionary must have an entry whose key is .notdef.
sfnts	array	*(Required)* An array of one or more PostScript language string objects. These strings, concatenated, are treated as the binary representation of the TrueType font. See also the *Adobe Technical Note # 5012*.

5.2.3 CFF and Chameleon Fonts: FontType 2 and FontType 14



The Compact Font Format (CFF) is a compact representation of one or more single master, multiple master, synthetic, and CID-keyed fonts. Unlike previous PostScript font formats, CFF allows multiple fonts to be stored together in a unit called a font set. It retains full fidelity to the original Type 1 fonts, while achieving significant space reduction due to a compact binary representation and sharing of data that is common to multiple fonts. The PostScript language representation of a CFF font has **FontType** 2.

Unlike previous PostScript font formats, CFF allows multiple fonts to be stored together in a unit called a *font set*. A font set is a single file that contains information that is shared among multiple fonts. The file contains the internal structure, including a directory of fonts and the various data structures. It saves space by using a compact binary representation for most of the information, sharing common data between fonts, and defaulting frequently occurring data. CFF uses the **FontType** 2 font format for its charstrings.

The Chameleon font format is an implementation of a “shape library” that allows compact representations of Roman font shapes for book and display faces. As with CFF, Chameleon fonts use a wrapper for the Chameleon master font and font descriptor database. The master font can be tailored to address the desired font needs of a product. Prescriptions (termed *font descriptors*) define how to extract fonts of interest from the master. The Chameleon font technology uses the **FontType** 14 format.

CFF is documented in the Compact Font Format specification listed in Appendix G, “Bibliography,” on page 395. The Chameleon font format, though using the same wrapper format as CFF, has an entirely different internal representation, which is not documented.

FontSet Resource



The external representation for a **FontSet** resource is a minimal PostScript wrapper enclosing the binary data that comprise the CFF. An example of the syntax follows:

```
%!PS-Adobe-3.0 Resource-FontSet
% 4-binary-bytes
%%DocumentNeededResources: ProcSet (FontSetInit)
%%IncludeResource: ProcSet (FontSetInit)
%%BeginResource: FontSet
%%Version: realversion [intrevision]
/FontSetInit /ProcSet findresource begin
%%BeginData m Binary Bytes
name n mname StartData space n-binary-data-bytes
%%EndData
%%EndResource
%%EOF
```

Note *The mname operand is optional. The **StartData** operator takes two or three operands, depending on the type of the last operand. The mname operand is present only in a **FontSet** resource dictionary that contains Chameleon font descriptors; it gives the name of the associated master font.*

When this code is executed, the **StartData** operator in the **FontSetInit** procedure set processes the binary data, builds a **FontSet** resource dictionary, and executes the **defineresource** operator. The contents of the **FontSet** resource dictionary are undocumented and subject to change.

Note ***StartData** is the only operator in the **FontSetInit ProcSet** that is documented as part of the PostScript language. Other entries in the **FontSetInit ProcSet** are private to the implementation of font sets and should not be accessed by a PostScript language program.*

Integration of CFF and Chameleon Fonts into the PostScript Language



The fonts contained in all CFF and Chameleon **FontSet** resources also appear as instances of the **Font** resource category, which makes them individually accessible via the normal LanguageLevel 2 font resource operators, such as **findfont**, **findresource**, and **resourceforall**. The mechanism by which this is achieved is not specified as part of the PostScript language.

CFF and Chameleon fonts are integrated into the PostScript language font model in a way that is compatible with both LanguageLevel 1 (filesystem) and LanguageLevel 2 (resource) methods for accessing fonts. The fonts exist in a fictitious file system whose “files” are the constituent fonts in all available **FontSet** resource instances. This file system is named %fontset% and has the following characteristics and behavior:

- The file system is read only and is searchable. That is, it is among the file systems considered when no file system name is specified explicitly.

- The file system appears to have names of the form (fonts/Palatino). The prefix is the constant string *fonts/*, which is the standard **FontResourceDir** system parameter. The suffix is a constituent font in some **FontSet** resource instance.

- Font look-up operations will tend to cause all **FontSet** resource instances to be loaded into VM as a side effect. A **FontSet** dictionary takes minimal VM (a few hundred bytes) if loaded from a positionable file (ROM, cartridge, or disk file system). Furthermore, **FontSet** instances are expected to be few in number. (Most printers will have at most three—one for all the CFF fonts and two for the Chameleon master font and font descriptor database.)

- File systems are searched in order of their **SearchOrder** device parameter. Thus, the %fontset% file system will be consulted before or after other file systems, such as cartridges or disks, according to their relative **SearchOrder** values. Specifying %fontset% to be searched first will yield the best performance for fonts present in the **FontSet** dictionaries (since no other file systems need to be consulted), but it will preclude overriding a **FontSet** font by installing a font with the same name on disk.

- The **Searchable** and **SearchOrder** parameters of the %fontset% file system can be changed by the **setdevparams** operator.

Opening a file using

```
(%fontset%fonts/Palatino) (r) file
```

or

```
(fonts/Palatino) (r) file
```

where the %fontset% file system is the one chosen, causes the **FontSet** resource instances to be enumerated until one containing the named font is found. (**FontSet** instances are loaded into VM as a side effect of this enumeration.) A fictitious file is then fabricated and made available for reading.

When this fictitious file is executed, it causes the font dictionary to be built and defined as an instance of the **Font** resource category. The contents of the file are undocumented and subject to change.

Note *The fictitious file does not have any DSC boilerplate. This means that* ***Font resourcestatus*** *will not find a* `%%VMusage` *comment and will return –1 as the size result.*

The resulting font dictionary has **FontType** 2 (for CFF) or 14 (for Chameleon), plus all the required entries for a Type 1 base font dictionary, as defined in Tables 5.1, 5.2, and 5.3 on pages 266 and 267 in the *PostScript Language Reference Manual, Second Edition*. Note that the dictionary does not have a **Private** entry, and it has additional entries that are undocumented and subject to change.

A PostScript program can copy the font dictionary and insert or modify entries as specified in Tables 5.1, 5.2, and 5.3. These modifications have the same effects for Type 2 and Type 14 fonts as for Type 1 fonts.

Executing the status operator for a valid file name in the %fontset% file system yields four meaningless numbers followed by *true* if the named font exists in any **FontSet**; otherwise it yields *false*.

5.3 CID-Keyed Fonts

The PostScript language extensions for CID-keyed fonts provide a convenient and efficient method for defining multibyte character encodings, base fonts with a large number of character descriptions, and composite fonts which use these base fonts and character encodings. Additionally, the language extensions provide straightforward methods for rearranging existing fonts. With these extensions, character codes can easily be remapped to select different character descriptions from the same font or from different fonts altogether.

These language extensions provide built-in PostScript interpreter support for CIDFont and CMap files. Specifically, these extensions define **CIDFont** and **CMap** resource categories, introduce the **composefont** operator, and redefine other PostScript resource categories, operators, and procedures to work with this technology. The VM structures created by redefined functions may have different content when compared with similar structures created by earlier PostScript interpreters. CID-keyed composite fonts and font rearrangement are consequences of these extensions.

For detailed information on creating CIDFont and CMap files, see *Adobe CMap and CIDFont Files Specification,* listed in Appendix G, “Bibliography,” on page 395.

Terms Used in This Section

The following terms are used throughout this section on CID fonts:

- A *character collection* consists of an ordered set of all characters needed to support one or more popular character sets for a particular language.

- A *CID*, or *character identifier*, is an unsigned integer that identifies a character from a character collection and is used to access glyph descriptions in CIDFonts. It is often referred to as a CID value or as a CID number. CIDs are used instead of glyph names or character codes to access glyph data and are analogous to the character names used in other types of base fonts. The order of characters in a character collection determines the CID number for each character.

- A *CIDFont* constitutes a new base font type. A CIDFont produces a dictionary that can be used as a base font in Type 0 composite fonts with **FMapType** 9. A CIDFont is an instance of the **CIDFont** resource. The dictionaries of the **CIDFont** resource can be used to access large collections of character outlines that are stored within a single dictionary.

- A *character selector* is a name, CID, or character code used to select glyph data from within a CIDFont or other font type.

- A *CMap* is an instance of the **CMap** resource that is used to map character codes to character selector and font numbers. It is a dictionary that can be used as the **CMap** entry in Type 0 composite fonts with **FMapType** 9.

- A *CID-keyed font* is a composite font (Type 0) made using a CMap and one or more CIDFonts.

- A *font number* is an integer used to select an element in a composite font's **FDepVector** which is an array of font dictionaries.

5.3.1 CID Font Resources

Instances of the **CIDFont** resource category are dictionaries that represent glyph procedures. The glyph procedures may be defined as Type 1 **CharStrings**, PostScript **BuildGlyph** procedures, or TrueType glyph procedures. An integer character identifier (CID) is used to access glyph outlines in CID fonts.

Instances of the **CIDFont** resource category are defined and accessed by resource operators, such as **defineresource** and **findresource**. See Section 3.9.1 in the *PostScript Language Reference Manual, Second Edition*. The **CIDFont** resource category is independent of the **Font** resource category.The **findfont** operator or /Font **findresource** cannot be used to load CID fonts into VM, nor will /Font **resourceforall** list CID fonts.

A CID font file may be stored on an external storage device or included in a PostScript program. A file stored on an external storage device is loaded by the **findresource** operator and built in global VM. A file defined by a PostScript program is built in the VM allocation mode that is active when the CID font is created.

Instances of the **CIDFont** resource category can be used with the **makefont**, **scalefont**, **selectfont**, **setfont**, and **currentfont** operators. If a **CIDFont** resource category instance is the current font, however, **glyphshow** is the only show operator that can be used. The **glyphshow** operator can take either an integer or a name object as an argument. When the current font is a CID font, the integer will be used as the CID to find and show the glyph.

The following show operators cannot be used when the current font is a **CIDFont** resource instance (an **invalidfont** error will occur):

ashow	**show**	**xshow**
cshow	**stringwidth**	**xyshow**
kshow	**widthshow**	**yshow**

Note *These **show** operator limitations do not apply to CID-keyed fonts that are produced as the composite of a CMap and one or more CID fonts.*

CIDFont dictionaries can also be used as base fonts of composite fonts (**FontType** 0) with **FMapType** 9. The **CMap** entry in a composite font with **FMapType** 9 provides the mapping from character code to CID. The CID is then used to access the glyph in a CID font. CID fonts are independent of character mapping and can be used with a number of different character encodings.

The types of CID fonts are shown in Table 5.4. To be used as base fonts, these CID fonts must have a corresponding **FontType** value. The **FontType** value is added to the **CIDFont** dictionary by the **defineresource** operator for the **CIDFont** resource category.

Table 5.4 *CIDFontType and FontType values*

CIDFontType	*FontType*	*Description*
0	9	Type 1-based glyph procedures. See the section “CIDFontType 0 CID Font File” on page 139.
1	10	PostScript **BuildGlyph** procedure, similar to Type 3 fonts. See the section “CIDFontType 1 CID Font” on page 143.
2	11	TrueType glyph procedure. See the section “CIDFontType 2 CID Font” on page 143.

Table 5.4 *CIDFontType (continued)and FontType values*

CIDFontType	*FontType*	*Description*
4	32	Bitmap font. See Section 5.2.1, “Bitmap Fonts: Font Type 32,” on page 129.

Table 5.5 lists the entries present in all **CIDFont** dictionaries, regardless of the **CIDFontType** value.

Table 5.5 *Entries in all types of CIDFont dictionaries*

Key	*Type*	*Semantics*
CIDFontName	key	*(Required)* Identifies the name of the **CIDFont** resource instance.
CIDFontType	integer	*(Required)* Tells what is in the **CIDFont** resource instance, how it is organized, and how it is represented.
CIDSystemInfo	dictionary	*(Required)* Identifies the character collection used by the CID font. For a complete listing of the entries contained in the **CIDSystemInfo** dictionary, see Table 5.11 on page 145.
FontBBox	array	*(Required)* An array of four numbers in the character coordinate system that gives the lower-left *x*, lower-left *y*, upper-right *x*, and upper-right *y* of the font bounding box. See the *PostScript Language Reference Manual, Second Edition*, Table 5.2.
FontMatrix	array	(*Required, except for* ***CIDFontType*** *0*) Defines the transformation from character-space to user space coordinates. All values that are in character-space units (for example, **FontBBox**, **Metrics**, **StrokeWidth** entries) must be in terms of the character-space units defined by this **FontMatrix** entry. See Table 5.6 on page 140 for special considerations that apply to **CIDFontType** 0 only.
FontInfo	dictionary	*(Optional)* See Table 5.4 on page 268 in the *PostScript Language Reference Manual, Second Edition*.
FontType	integer	*(Optional; set by* ***defineresource*** *if not present)* If this entry is missing from a **CIDFont** dictionary, the correct **FontType** is added by the **defineresource** operator, as indicated in Section 5.3.1, “CID Font Resources,” on page 136. (This entry is present for compatibility with applications that expect to find a **FontType** entry in every base font.)
UIDBase	integer	(*Optional*) Used only in PostScript interpreters lacking CID support. **UIDBase** is used in combination with the **UIDOffset** in a CMap to form a unique ID for the base fonts of a composite font.
WMode	integer	*(Optional)* Indicates which of the two sets of metrics will be used when characters are shown from the CID font. See the *PostScript Language Reference Manual, Second Edition*, Section 5.4 on page 271. When used in a CID-keyed composite font, this value is overridden by the **WMode** value obtained from a **CMap** resource instance. Default value: 0

Table 5.5 *Entries in all types of CIDFont dictionaries (continued)*

Key	*Type*	*Semantics*
XUID	array	*(Optional)* An array of integers that uniquely identifies this CID font or any variant of it. See the *PostScript Language Reference Manual, Second Edition*, Section 5.8 on page 283.

CIDFontType 0 CID Font File

A **CIDFontType** 0 CID font file may be considered to be partitioned into two contiguous sections, as follows:

- A *PostScript section* that defines the top-level **CIDFont** dictionary, as described below (Table 5.6 on page 140).
- A *binary data section* that consists of two offset tables that point to the glyph subroutine data and glyph procedure table. All four tables are in the binary data section.

The two sections are separated by an invocation of the **StartData** operator in the **CIDInit ProcSet**. The binary data section begins immediately following the white space character that terminates the invocation of **StartData**. The binary data's length is given as an operand of **StartData**. For further information on the construction of a **CIDFont**, see the *Adobe CMap and CID Font Files Specification* in Appendix G, , "Bibliography," on page 395.

The distinguishing characteristic of **CIDFontType** 0 CID font files is that the binary data section contains a Type 1 glyph procedure. That is, the glyph procedures conform to the same format as the procedure in the **CharStrings** dictionary in a Type 1 font, as described in Chapter 6 in *Adobe Type 1 Font Format*.

The PostScript section defines the **CIDFont** dictionary and various subsidiary data structures that contain information required to interpret the binary data section (Table 5.6 on page 140).

The **FDArray** entry is an array of subsidiary dictionaries, each of which is a stripped down font dictionary containing a **Private** dictionary and a few other entries. Each subsidiary dictionary in the **FDArray** is used by a subset of the glyph definitions in the **CIDFont** to obtain **Private** dictionary information, including subroutines.

The binary data section includes an offset table, whose position is specified by **CIDMapOffset** in the **CIDFont** dictionary. This table is indexed by CID. For each CID, it contains an **FDArray** index (**FDBytes** long) followed by a charstring offset (**GDBytes** long). The **FDArray** index specifies which **FDArray** subsidiary dictionary to use for **Private** information; the charstring offset gives the location of the Type 1 charstring for the glyph definition.

Table 5.6 gives the entries specific to a **CIDFontType** 0 **CIDFont** dictionary.

Note *The charstrings in a **CIDFontType** 0 CID font cannot use the Type 1 font program charstring **seac** command for generating accented characters.*

Table 5.6 *Additional entries in a CIDFontType 0 CIDFont dictionary*

Key	*Type*	*Semantics*
CDevProc	procedure	*(Optional)* Algorithmically derives global changes to the metrics of a font. When this procedure is executed, eleven elements are put on the stack. For CID fonts, the eleventh operand is the CID value. See Section 5.6.2 in the *PostScript Language Reference Manual, Second Edition*, page 277, for further information.
CIDCount	integer	*(Required)* Specifies the count of valid CIDs in the CID font. Valid CIDs are in the range 0 to [(value of **CIDCount**) − 1]. The CID font must handle all valid CIDs.
CIDMapOffset	integer	*(Required)* Byte offset to the CIDMap (glyph procedure offset) table in the binary data section of the CID font. The offset is relative from the beginning of the binary data section and is typically 0 (for those CID fonts whose binary data section begins with the CIDMap table). The CIDMap table is indexed by CID, and each entry in the table consists of an **FDArray** index (**FDBytes** long), followed by an offset to the charstring data (**GDBytes** bytes long).
FDArray	array	*(Required)* Array of dictionaries, each of which contains information used with charstring data to render glyphs. A **CIDFontType** 0 CID font must have an **FDArray** with at least one dictionary, but typically has more. Glyphs having common hinting requirements reference the same dictionary within the **FDArray**. See the *Adobe Type 1 Font Format Specification* for further information.
FDBytes	integer	*(Required)* Number of bytes used to store the **FDArray** index for each CID in the CID font. If **FDBytes** is equal to 0, this table contains no **FDArray** indices, and the **FDArray** index of 0 is assumed for all CIDs.
FontMatrix	array	*(Optional; inserted by the **defineresource** operator if not present)* The **FontMatrix** entry in the top-level **CIDFont** dictionary defines the transformation from character-space to user-space coordinates. All values that are in character-space units (for example, **FontBBox**, **Metrics**, **StrokeWidth**) must be in the character-space units defined by this **FontMatrix** entry. At character rendering time, any **FontMatrix** entry in the **FDArray** dictionary is concatenated with the **FontMatrix** entry at the top level of the CID font. Since the various **FontMatrix** entries in the **FDArray** dictionaries can be different, this allows the definition of glyphs that use different character-space units in the same **CIDFontType** 0 CID font.

Table 5.6 *Additional entries in a CIDFontType 0 CIDFont dictionary (continued)*

Key	*Type*	*Semantics*
		If a CID font does not contain a **FontMatrix** entry at the top level, a [.001 0 0 .001 0 0] matrix is added automatically by the **defineresource** operator. This matrix defines 1000 character-space units to a 1-point user space transformation. The **FontMatrix** entry in each **FDArray** dictionary is replaced by the concatenation of its original value and the inverse of the [.001 0 0 .001 0 0] matrix.
GDBytes	integer	*(Required)* Number of bytes used to store the charstring byte offset for each CID in the CID font.
GlyphData	string, array, or integer	(*Required; inserted by* ***StartData***) If the CIDFont has been defined by direct execution of a CIDFont file embedded in the job stream, **GlyphData** is either a string or an array of strings containing the binary data section of the CIDFont. However, if the CIDFont has been loaded from a random-access filesystem by **findresource**, **GlyphData** is an integer whose meaning is implementation dependent. In the latter case, the binary data section is not loaded into VM; instead, portions of it are accessed dynamically as needed during character rendering.
GlyphDirectory	array or dictionary	*(Optional)* Container for the incremental addition of character descriptions to a CID font. See Section 5.3.3, “Subsets and Incremental Definition of Glyph Procedures,” on page 145 for further information.
Metrics	dictionary	*(Optional)* Width and sidebearing information for writing mode 0. This entry is not normally present in the original definition of a CID font. Adding a **Metrics** dictionary to a CID font overrides the widths of the sidebearings encoded in the character descriptions. For CID fonts, the keys in the **Metrics** dictionary are integers representing CID values. See the *PostScript Language Reference Manual*, *Second Edition*, Section 5.4 on page 271 and Section 5.6.2 on page 276 for further information on the **Metrics** dictionary. The units of the values in the **Metrics** dictionary are given in terms of character-space units defined by the **FontMatrix** entry in the top level **CIDFont** dictionary.
Metrics2	dictionary	*(Optional)* Metric information for writing mode 1. For CID fonts, the keys in the **Metrics2** dictionary are integers representing CID values. See the *PostScript Language Reference Manual*, *Second Edition*, Section 5.4 on page 271 and Section 5.6.2 on page 276 for further information. The units of the values in the **Metrics** dictionary are given in terms of character-space units defined by the **FontMatrix** entry in the top level **CIDFont** dictionary.
PaintType	integer	*(Optional)* A code indicating how the characters of the CID font are to be painted: 0 Character outlines are filled. 2 Character outlines (designed to be filled) are stroked.

Table 5.6 *Additional entries in a CIDFontType 0 CIDFont dictionary (continued)*

Key	*Type*	*Semantics*
		CIDFontType 0 CID fonts are ordinarily created with a **PaintType** of 0. A program desiring to convert to a stroked outline CID font can copy the **CIDFont** dictionary, change the **PaintType** from 0 to 2, add a **StrokeWidth** entry, and define a new CID font using this dictionary.
StrokeWidth	integer	*(Optional)* Stroke width (in units of the character coordinate system) for stroke-outlined CID fonts (**PaintType** 2). This field is not initially present in filled CID font descriptions. It must be added when creating a stroked font from an existing filled font. Default value: 0

Table 5.7 gives the entries in a dictionary in the **FDArray**.

Table 5.7 *Entries in a dictionary in the FDArray*

Key	*Type*	*Semantics*
FontName	string	(*Optional*) Name of the font.
FontMatrix	array	*(Required)* Transformation from the character-space for glyphs associated with this **FDArray** subsidiary dictionary to the character-space for the CID font overall.
Private	dictionary	*(Required)* See the following section, “Entries in the Private Dictionary of a CIDFontType 0 CID Font.”

Entries in the Private Dictionary of a CIDFontType 0 CID Font

The *Adobe Type 1 Font Format Specification* details how to construct a **Private** dictionary in the **FDArray** subdictionary of a **CIDFontType** 0 CID font. However, there is an alternative representation for the subroutine data, and this alternative representation is useful with CID fonts.

Instead of the **Subrs** array, the subroutine data can be contained in the binary data portion of the CID font. In this case, the **Subrs** array is replaced by the entries in Table 5.8.

Table 5.8 *Additional entries in a Private dictionary of a CIDFontType 0 CIDFont*

Key	*Type*	*Semantics*
SDBytes	integer	*(Required)* Number of bytes used to represent each offset in the subroutine offset table for glyphs referencing this dictionary in the **FDArray** entry.

Table 5.8 *Additional entries in a Private dictionary of a CIDFontType 0 CIDFont (continued)*

Key	*Type*	*Semantics*
SubrMapOffset	integer	*(Required)* Byte offset of the start of the subroutine offsets table relative to the beginning of the binary portion of the CID font. This table is indexed by subroutine number. Each element of the table is **SDBytes** long and is interpreted as the offset of the start of the subroutine data relative to the beginning of the binary portion of the CID font.
SubrCount	integer	*(Required)* Number of subroutines, numbered from 0 to (**SubrCount** – 1).

CIDFontType 1 CID Font

CIDFontType 1 CID fonts use PostScript language procedures to construct glyphs. This **CIDFontType** is the CID font analog to Type 3 fonts. However, instead of using a character code or character name to select the character to build, a CID is used. **CIDFontType** 1 CID fonts do not have a binary data section; they consist of PostScript code only.

When a character is shown from a **CIDFontType** 1 CID font and the character is not cached, the PostScript interpreter will:

1. Push the current **CIDFont** dictionary, followed by an integer CID, onto the operand stack.

2. Execute the font's **BuildGlyph** procedure. The **BuildGlyph** procedure must remove these two objects from the operand stack and use this information to construct the requested character.

The **BuildGlyph** procedure must follow the guidelines described in the *PostScript Language Reference Manual, Second Edition,* Section 5.7 on page 278. In particular, the character metrics for the glyph must be supplied by executing one of the **setcachedevice**, **setcachedevice2**, or the **setcharwidth** operators.

Table 5.9 gives the entries specific to the **CIDFontType** 1 **CIDFont** dictionary.

Table 5.9 *Additional entries in a CIDFontType 1 CIDFont dictionary*

Key	*Type*	*Semantics*
BuildGlyph	procedure	*(Required)* Constructs the requested character. The **CIDFont** dictionary, followed by the CID for the character, is on the stack when the procedure is called.

CIDFontType 2 CID Font

TrueType font programs that use multibyte (1 to 4 byte, possibly mixed-byte) encodings are supported by **CIDFontType** 2 CID fonts. **CIDFontType** 2 CID

fonts do not have a binary data section, but are composed of PostScript code, though some individual dictionary entries may contain binary data. As with other CID font types, these CID fonts typically contain a large number of glyphs. The CID font format for TrueType fonts was chosen for compatibility with PostScript interpreters that have Type 42 font support but lack CID font support (assuming the presence of the CID Support Library). When used in PostScript interpreters containing CID font support, some entries are ignored.

Table 5.10 gives the entries specific to the **CIDFontType** 2 CID fonts.

Table 5.10 *Additional entries in a CIDFontType 2 CID fonts*

Key	*Type*	*Semantics*
CharStrings	dictionary	*(Required)* Required for backward compatibility with the existing Type 42 font format. When this font is used as a **CIDFont** resource instance, this entry is ignored. At a minimum, the **CharStrings** dictionary must have a .notdef entry with a glyph index value of 0.
CIDCount	integer	*(Required)* Number of valid CIDs in the CID font. Valid CIDs are in the range 0 to (**CIDCount** – 1), inclusive.
CIDMap	array or string	*(Required)* In TrueType fonts, glyphs are accessed by a glyph index. However, the glyph index for a particular glyph may vary from one font to another (for example, the glyph index for “A” may be 33 in one font and 43 in another font). For glyphs to be accessed by CIDs, the mapping between CID and glyph index must be provided for each TrueType font. The **CIDMap** is a table that contains the glyph index for each CID. It must be **CIDCount** × **GDBytes** long. The table may be represented as a single string or as an array of strings. When represented as an array of strings, each string must be a multiple of **GDBytes** long. Logically, the table is an array of glyph indices, indexed by CID. Each entry is **GDBytes** long, representing the glyph index, high-order byte first.
Encoding	array	*(Required)* Required for backward compatibility with the existing Type 42 font format. When this font is used as a CID font, this entry is ignored.
FontType	integer	*(Required)* For backward compatibility with the existing Type 42 font format, the **FontType** must be 42. However, in an interpreter with CID font support, **defineresource** will replace this entry with FontType 11.
GDBytes	integer	*(Required)* Gives the number of bytes used to store the TrueType glyph index in the **CIDMap** table. In most cases this will be 2 bytes, allowing glyph indices in the range from 0 to 65535 inclusive.
GlyphDirectory	dictionary or array	*(Optional)* Container for incremental addition of glyph descriptions. See Section 5.3.3, “Subsets and Incremental Definition of Glyph Procedures,” on page 145.

Table 5.10 *Additional entries in a CIDFontType 2 CID fonts (continued)*

Key	*Type*	*Semantics*
sfnts	array	*(Required)* Array of strings that contains the TrueType font data. If the **sfnts** data will not fit within one string (typically, 65535 bytes), the data should be broken up into an array of strings. When a TrueType CID font is downloaded to a Type 42 compatible imaging system, the **sfnts** array will be stored in VM. See also the *Adobe Technical Note, # 5012*. If the CID font is located in a file system (disk or cartridge) and is loaded automatically by the **findresource** operator, the data in the **sfnts** array will not be loaded into VM. Instead, the font machinery will access the data from the file on demand when characters are shown. For most efficient access, the data in the **sfnts** array should be defined using a binary format. For complete information on construction of **sfnts**, see *Type 42 Font Format Specification* listed in Appendix G, "Bibliography," on page 395.

5.3.2 CIDSystemInfo Dictionary Entries

The **CIDSystemInfo** dictionary entries in CIDFonts and CMap dictionaries specify the character collection that these resources assume. A character collection consists of an ordered set of all the characters needed to support one or more popular character sets. The order of the characters in the character collection determines the CID number for each character.

A character collection is uniquely identified by the **Registry**, **Ordering**, and **Supplement** keys in the **CIDSystemInfo** dictionary, as described in Table 5.11. Character collections, whose **Registry** and **Ordering** values are the same, are compatible. The **CIDSystemInfo** entry of a CMap must be compatible with all of the CID fonts with which it is used.

Table 5.11 *Entries in a CIDSystemInfo dictionary*

Key	*Type*	*Semantics*
Ordering	string	*(Required)* Uniquely names an ordered character collection available within a **Registry** string. For example, (Japan1).
Registry	string	*(Required)* Identifies an issuer of character collections. Assigned by the UniqueID Coordinator at Adobe. For example, (Adobe).
Supplement	integer	*(Required)* Identifies the supplement number of the character collection. Supplements do not alter the ordering of previous supplements and are numbered beginning with 0. This value is not used in determining the compatibility between a CMap and CID fonts.

5.3.3 Subsets and Incremental Definition of Glyph Procedures

Typically, **CIDFont** resource instances are quite large and contain many glyph procedures. Any particular PostScript program may need to access only a

subset of these glyph procedures. If the CID font is defined as part of the PostScript program rather than loaded in from an external storage device, using only the glyph procedures actually needed by the PostScript program can greatly reduce the size of the PostScript program. Consequently, instances of a **CIDFont** resource category can be defined using only a subset of the original CID font's glyph procedures. This practice is called *subsetting*.

A CID font can also be defined to allow glyph procedures to be added incrementally during the execution of a PostScript program. This practice is called *incremental definition.* With incremental definition, the glyph's procedures need be defined only prior to the first use.

A **CIDFontType** 0 or 2 CID font can optionally contain a **GlyphDirectory** entry whose value is either an array or a dictionary. This entry is used for incremental downloading or subsetting of glyph procedures. The semantics of the **GlyphDirectory** are different for each **CIDFontType**, and these differences are described in the sections that follow. Subsets and incremental definitions for **CIDFontType** 1 CID fonts are also permitted; however, these depend on how the PostScript program defines the **BuildGlyph** procedure.

Semantics of GlyphDirectory for CIDFontType 0

If the **GlyphDirectory** entry is an array, its length must be the value of the **CIDCount** entry plus one. Each array element can be either *null* (indicating an empty entry) or a string. If **GlyphDirectory** is a dictionary, each key is an integer CID. The value is a string, as described above.

Each string consists of an optional **FDArray** index followed by a Type 1 charstring for the glyph procedure. The **FDArray** index is present only if **FDBytes** is non-zero; it is **FDBytes** long. The charstring may be encrypted, but the encryption applies only to the charstring data itself, not to the bytes representing the **FDArray** index.

After defining such a CID font, a PostScript program may insert entries into the **GlyphDirectory** containing new glyph descriptions, either by replacing *null* array entries with strings or by inserting new dictionary entries. It may not replace non-*null* entries; any attempts to do so will yield unpredictable results due to font caching.

As is true for all CID fonts, any attempt to use the **show** operator on a character whose CID selects a *null* or missing entry in the **GlyphDirectory** entry will cause the CMap to be consulted to see if there is a notdef CID for this character code. If there is, it will be used; otherwise, a CID of 0 will be used. If the **GlyphDirectory** entry for CID 0 is not present or is *null*, an **invalidfont** error will occur.

The binary data section of a CID font with a **GlyphDirectory** entry does not need an offset table for the glyph procedures; neither does it need to contain charstring data (if present, this data will be ignored). However, the binary data section must contain subroutine offset tables and subroutines used by any charstrings that are downloaded. The information contained in **FDArray** must be supplied when the CID font is created. No provision is made for downloading any of this data incrementally.

Semantics of GlyphDirectory for CIDFontType 2

For a **CIDFontType** 2, the CID is used as an index into the CIDMap table to obtain a glyph index. In the absence of a **GlyphDirectory**, the glyph index is used to access the TrueType data directly. In the presence of a **GlyphDirectory**, the glyph index is used as a key in **GlyphDirectory** to contain the glyph definitions.

If **GlyphDirectory** is an array, its length must be greater than the highest glyph index used in the font. Each array element can be either *null* (indicating an empty entry) or a string containing the TrueType glyph description for a given glyph index.

If **GlyphDirectory** is a dictionary, each key is an integer glyph index**.** The value is a string containing the TrueType glyph description.

After defining a CID font, a PostScript program may insert entries into the **GlyphDirectory** entry containing new glyph descriptions, either by replacing *null* array entries with strings or by inserting new dictionary entries. It may not replace non-*null* entries; any attempts to do so will yield unpredictable results due to font caching.

As is true for all CID fonts, if an attempt is made to perform a **show** operation on a character whose CID selects a *null* or missing entry in the **GlyphDirectory** entry, the CMap will be consulted to see if there is a notdef CID for this character code. If there is, it will be used; otherwise, a CID of 0 will be used. The **GlyphDirectory** entry for CID 0 must be present and non-*null*; otherwise, an **invalidfont** error will occur.

The **GlyphDirectory** key can be used only with a TrueType font (contained in the **sfnts** strings) meeting the following requirements:

- The TrueType font tables named (tagged) “loca” and “glyf” must not be present.
- There must be a TrueType font table named (tagged) “gdir,” whose size and offset are 0. This is simply an indication that the TrueType font was constructed for incremental downloading; the “gdir” table contains no useful information.

A TrueType glyph can contain references to one or more other glyphs in the same font—for example, to produce accented characters. These references are by glyph index. An incrementally downloaded font containing such glyphs must also contain all component glyphs that they reference.

A glyph's metrics are not part of the glyph description itself; rather, they come from a parallel table "hmtx," which is also indexed by glyph index. No provision is made for downloading the contents of the "hmtx" table incrementally; this information must be present at the time the font is defined.

Memory Use

The client can choose to represent the **GlyphDirectory** entry either as an array or as a dictionary. If an array is used, the array must be allocated with enough entries to store the highest CID or glyph index value that is expected. Any unused entries in the array will be wasted space. An array of a given length consumes about 40 percent as much memory as a dictionary of the same maxlength. Thus, the dictionary representation is best for a sparsely populated font containing less than 40 percent of its glyphs.

save and restore

If a glyph is added to the **GlyphDirectory** entry between **save** and **restore** operations, the **restore** operator will remove it. This is equivalent to replacing a non-*null* entry with *null*, which produces unpredictable results. To avoid this problem, the driver must download the same glyph definition again before the next attempt to perform a **show** operation on that character.

5.3.4 CMap Resources

Instances of the **CMap** resource category contain character encoding information. These instances define the mapping from a character code to a character selector and a font number. The character selector can be a CID, a glyph name, or a character code. The font number is an integer used to select a font from an array of fonts.

Character encodings that use single-byte or multibyte character codes are supported by **CMap** resource instances.

Instances of the **CMap** resource category are suitable for use as the **CMap** entry of Type 0 (composite) font dictionaries with the **FMapType** entry set to 9. They are created by executing the operators defined in the **CIDInit ProcSet** dictionary. These operators define mappings from character codes to a character selector and a font number.

As with all resource instances, **CIDFont** resource instances are defined and accessed by means of the **defineresource**, **findresource**, and other resource operators described on pages 86–87 in the *PostScript Language Reference*

Manual, Second Edition. Instances of the **CMap** resource category can contain the entries listed Table 5.12.

Table 5.12 *Entries in a CMap dictionary*

Key	*Type*	*Semantics*
CMapName	name	*(Required)* Identifies the name of the **CMap** resource instance.
CMapVersion	integer	*(Optional)* Version number of this CMap.
XUID	array	(*Optional)* Array of integers that uniquely identifies the CMap or any variant of it. See Section 5.8.2 in the *PostScript Language Reference Manual, Second Edition,* for further information.
UIDOffset	integer	*(Optional)* Used only on PostScript interpreters lacking CID support. **UIDOffset** is used in combination with the **UIDBase** entry in a CID font to form a unique ID for the base fonts of a composite font.
WMode	integer	*(Required)* Used as the value of the **WMode** entry in any font constructed by the **composefont** operator using this CMap. **WMode** in the CMap overrides any **WMode** in any font or CID font referred to by the CMap file. See the *PostScript Language Reference Manual, Second Edition,* Section 5.4 on page 271 for further information. Default value: 0
CIDSystemInfo	dictionary or array	*(Required)* A CMap selects characters from one or more fonts. The descendent font(s) selected can be CID fonts, other base-font types, or other composite fonts. The **CIDSystemInfo** entry identifies the character collection compatibility for each descendent font. If the CMap selects only one descendent font and the descendent font is a CID font, this entry can be a dictionary. Otherwise, this entry is an array of dictionaries indexed by the **usefont** number. To indicate that a descendent font is not a CID font, the corresponding array element is *null*. For details of the **CIDSystemInfo** dictionaries, see Section 5.3.2, “CIDSystemInfo Dictionary Entries,” on page 145.
CodeMap	PostScript object	*(Required)* Contains the information used to map from character codes to a character selector. The contents and format are implementation dependent. **CodeMap** is used by a decoder built into the interpreter and is dependent on the version of the interpreter being used. It is created during the definition of the **CMap** resource instance by executing the operators defined in the **CIDInit ProcSet** resource category.

5.3.5 CIDInit ProcSet

CIDInit is a **ProcSet** resource category that defines the operators needed to create a CIDFont or CMap. It also defines other operators and procedures so that CID-keyed font capabilities are supported. The **CIDInit ProcSet** is put on the dictionary stack by executing:

```
/CIDInit /ProcSet findresource begin
```

The following is a summary of the operators contained in the **CIDInit ProcSet** that are used to define CMaps and to perform font rearrangement. These operators fall into the following four groups:

General Operators

/CMapName	**usecmap –**	Uses another CMap's VM resource.
fontID	**usefont –**	Specifies font used subsequently.
fontID	**beginusematrix –**	
[a b c d tx ty]	**endusematrix –**	Transformation matrix to use with font.
int	**begincodespacerange**	
$srcCode_1$ $srcCode_2$	**endcodespacerange** –	Sets valid input codes.
–	**begincmap –**	
–	**endcmap –**	Brackets CMap definition.
/newFontName [component fonts array]	**beginrearrangedfont –**	Identifies fonts used in rearrangement.
–	**endrearrangedfont –**	Builds font on existing template.

Operators that Use Character Names or Character Codes as Selectors

int	**beginbfchar –**	
srcCode dstCode		
or		
srcCode dstCharname	**endbfchar –**	Specifies one base font glyph.
int	**beginbfrange –**	
srcCodeLo srcCodeHi dstCodeLo		
or		
srcCodeLo srcCodeHi		
[/$dstCharname_1$.../$dstChrname_n$]	**endbfrange**	Specifies range of base font glyphs.

Operators that Use CIDs as Selectors

int	**begincidchar –**	
srcCode dstCID	**endcidchar –**	Specifies one CIDFont character.
int	**begincidrange –**	
srcCodeLo srcCodeHi dstCIDLo	**endcidrange –**	Specifies range of CIDFont characters.

notdef Operators

int	**beginnotdefchar –**	
srcCode dstCID	**endnotdefchar –**	Specifies one notdef character.
int	**beginnotdefrange –**	
srcCodeLo srcCodeHi dstCIDLo	**endnotdefrange –**	Specifies range of notdef characters.

Note *In addition to the operators listed above, the* ***CIDInit ProcSet*** *includes the* ***StartData*** *operator, which is used in defining CID fonts. Other entries in the* ***CIDInit ProcSet*** *are private and should not be accessed by a PostScript program.*

5.3.6 CMap Example

The following **CMap** example demonstrates the use of several of the **CIDInit ProcSet** operators to define mappings from character codes to a character selector and font number.

```
/CIDInit /ProcSet findresource begin
12 dict begin
begincmap

% This CMap maps character codes to three different
% fonts; therefore, the CIDSystemInfo entry must be
% an array with three elements.

/CIDSystemInfo
    [
    3 dict dup
      begin
      /Registry (Adobe) def
      /Ordering (Japan1) def
      /Supplement 1 def
      end
    null  % null is used for base fonts that are not
          % CIDFonts.
    null
    ]
def

/CMapName /ExampleCMap def

/CMapVersion 1 def
/CMapType 1 def

/XUID [1000000 10 ] def

/WMode 0 def

% The following codespace operators define the valid
% code ranges for a mixed single byte and double byte
% encoding.
```

```
4 begincodespacerange
    <00>  <80>
    <8140><9FFC>
    <A0>  <DF>
    <E040><FCFC>
endcodespacerange

0 usefont
% The following operators will define mappings
% for font number 0. This is a CIDFont using
% Adobe-Japan1-1 character collection.
% The following operator defines a mapping from
% character codes to CID.

4 begincidrange
    <20>  <7e>    231
    <8140><817e>  633
    <8180><81ac>  696
    <8940><897e>  1219
endcidrange

1 usefont
% The following operators will define mappings
% for font number 1. These mappings show examples
% of mapping codes to other codes or
% to character names.
2 beginbfrange
    <61><63>  <63>
    <41><43>  [/A /B /C]
endbfrange

% Codes can be mapped one code at a time.
3 beginbfchar
    <6A>  /j        % /j
    <6B>  <6B>      % /k
    <6C>  <6C>      % /l
endbfchar

% The following operators define a new matrix
% for font number 1. This matrix rotates the
% characters counterclockwise by 90 degrees.
1 beginusematrix
    [0 1 –1 0 0 0]
endusematrix

2 usefont
1 beginbfrange
    <64><66>  <44>  % /D /E /F
endbfrange
```

```
3 beginbfchar
    <6D>   /m          % /m
    <6E>   <6E>        % /n
    <6F>   <6F>        % /o
endbfchar

endcmap
currentdict

CMapName exch /CMap defineresource pop

end                    % CMap dictionary
end                    % CIDInit ProcSet
```

The above **CMap** can be used with base fonts to create a composite font with the **composefont** operator, as follows:

```
/NewFont /ExampleCMap [Ryumin-Light Helvetica Times-Roman]
      composefont pop
```

5.4 Composite Font Extension

A composite font is a collection of base fonts organized hierarchically. The font at the top level of the hierarchy is the *root font*. Fonts at a lower level of the hierarchy are called *descendant fonts*. When the current font is composite, the **show** operator and its variants behave differently than with base fonts. These operators use a mapping algorithm that decodes **show** strings to select characters from descendant base fonts. The mapping algorithm is identified by the **FMapType** entry in the root font dictionary. A value of **FMapType** 9 has been added to support composite fonts that use CMaps to define the character encoding. Composite fonts with **FMapType** 9 require a **CMap** entry and use the mapping algorithm described in Section 5.4.2, “Character Mapping.”

5.4.1 Type 0 Dictionary Entries

The following dictionary entry is specific to Type 0 (composite) fonts using the mapping algorithm described in Section 5.4.2, “Character Mapping,” and should be seen as an addendum to Table 5.5 on page 286 in the *PostScript Language Reference Manual, Second Edition*.

Table 5.13 *Entry specific to a Type 0 (composite) font dictionary*

Key	*Type*	*Semantics*
CMap	dictionary	*(Optional)* Instance of the **CMap** resource category, used only when the value of **FMapType** is 9. The dictionary contains a **CodeMap** entry used by a built-in decoder to decode the **show** string and generate a font number and a character code, a glyph name, or a CID.

5.4.2 Character Mapping

The following is the mapping algorithm used to support composite fonts that use a **CMap** resource:

1. Using the information in the **CodeMap** entry of a **CMap** dictionary, decode bytes from the **show** string and determine a font number and a character selector, which is a character code, glyph name, or CID.

2. Use the font number as an index to the **Encoding** array of the composite font to give an integer.

3. Use the integer from the previous step as an index to the **FDepVector** array to select a descendant font.

4. Use the character selector to select a glyph from the descendant font, in whatever way is appropriate for that font.

For further information, see the *PostScript Language Reference Manual, Second Edition*, Section 5.9.1, “Character Mapping,” on page 287.

Table 5.14 describes the new mapping algorithm that the **FMapType** value can select. It should be seen as an addendum to Table 5.6 on page 287 in the *PostScript Language Reference Manual, Second Edition*.

Table 5.14 *New FMapType mapping algorithm*

Algorithm	*FMapType*	*Explanation*
CMap mapping	9	One or more bytes are extracted from the **show** string according to information in the **CMap** dictionary. A built-in decoder produces a font number and either a character code, glyph name, or CID.

Glyph selection from the descendent font depends on the type of font and whether the decoder returns a name, character code, or CID.

If the character selector is a name, the descendent font must be a base font that is not a CID font. The equivalent of a **glyphshow** operation is executed on the name using the descendent font as the current font. If the name is not in the descendent font, the .notdef character is used instead.

If the character selector is a code, the descendant font may be either a base font (such as Type 1 or Type 3) or another composite font (Type 0), but not a CID font. In the case of a base font, the **Encoding** entry in the base font is used to determine which glyph to render. In the case of a composite font, the character code is interpreted according to the mapping algorithm specified by the **FMapType** value of the font. During interpretation of the character code by the descendant composite font, no additional bytes will be used from the **show** string.

If the character selector is a CID, the descendent font should usually be a **CIDFont** resource instance. The CID is used to look up the glyph in the **CIDFont**.

5.4.3 Undefined (notdef) Character Handling

The **CMap** entry in an **FMapType** 9 composite font can define different notdef CIDs for different code ranges. These notdef code ranges can overlap the defined code ranges. During a **show** operation, if the CID selected by the **CMap** mapping algorithm has no corresponding data in the CID font, then the **CMap** is consulted again to determine if the code has a notdef CID defined. If it does, then this CID is used; if it does not, then CID 0 is used. All CID fonts must contain a glyph definition for CID 0. Also, notdef ranges are allowed only with CID fonts and not with other types of base fonts.

Note *For **CIDFontType** 1 CID fonts, the **BuildGlyph** procedure of the font must handle undefined CIDs by creating the glyph for CID 0. The **BuildGlyph** procedure cannot consult the CMap again.*

If the **CMap** entry defines a mapping to a base font other than a **CIDFont** (for example, Type 1 or Type 3) and the glyph identifier is not in the base font, then the .notdef character of the base font is used.

Codes that are outside the codespace defined by the **CMap** resource dictionary will use the appropriate notdef glyph identifier for font number 0. If the font corresponding to font number 0 is a CID font, then the notdef character selector will be CID 0; if it is another type of base font, then the glyph name will be .notdef. If the font is a composite font, then the notdef character used is implementation dependent.

5.4.4 show String Parsing

The number of bytes consumed from the **show** string is determined by the codespace entries in the CMap. If the initial bytes are within any of the code ranges defined in the codespace, then the number of bytes consumed is the same as the number of bytes in that code range. If the initial bytes do not match any codespace entries, then the number of bytes consumed is the number of bytes in the shortest codespace range.

5.4.5 Nesting

Composite fonts with an **FMapType** value of 9 may be nested up to five levels and follow the rules for nonmodal fonts described in Section 5.9.3 in the *PostScript Language Reference Manual, Second Edition*. If the descendant font of an **FMapType** 9 font is a composite font, the decoder must return a character code (not a CID or name). This character code is input to the descendant font's mapping algorithm. The character code may be one or

more bytes long. However, the character code must be long enough for the descendant font's mapping algorithm to terminate. Additional bytes from the original **show** string cannot be used.

The **FontMatrix** entries of nested composite fonts with **FMapType** 9 are treated the same way as other composite fonts. The value of **FontMatrix** of the root font is concatenated to the value of **FontMatrix** of all descendant composite fonts (not the base fonts) by the **definefont**, **makefont**, or **scalefont** operator. When a character is shown, the **FontMatrix** of the immediate parent is concatenated to the **FontMatrix** of the base font. If the base font is a **CIDFontType** 0 CID font, the matrix is the concatenation of the immediate parent's **FontMatrix**, the CID font's **FontMatrix**, and the **FontMatrix** in the dictionary contained in the **FDArray** array.

CHAPTER 6

Additions and Modifications to the Rendering Model

This chapter supplements Chapter 6 in the *PostScript Language Reference Manual, Second Edition.* It provides information about device-dependent color spaces, CRD selection, synchronizing CRDs and ICC profiles, halftone dictionaries, extensions to process color models, and transfer functions.

Overview of This Chapter

6.1 UseCIEColor 158

6.2 Color Rendering 159

6.2.1 CRD Selection Based on Rendering Intent 159

Rendering intents (Table 6.1) 161

Relationship to Graphics State Parameters 161

Selecting CRDs by Rendering Intent: The findcolorrendering Operator 162

Modifying the CRD Selection Process 162

6.2.2 Synchronizing CRDs and ICC Profiles 163

Format of crdInfoTag (Table 6.2) 164

6.3 Halftone Dictionaries 166

6.3.1 Changes to the Type 1 through Type 5 Halftone Dictionaries 167

New entry for halftone dictionaries of type 1 through type 5 (Table 6.3) 167

6.3.2 New Halftone Dictionaries 167

Halftone dictionaries (Table 6.4) 167

6.3.3 Type 6 Halftone Dictionary 168

Entries in a type 6 halftone dictionary (Table 6.5) 168

6.3.4 Type 9 Halftone Dictionary 169

Entries in a type 9 halftone dictionary (Table 6.6) 169

6.3.5 Type 10 Halftone Dictionary 170

Halftone cell graphically represented in device space (Figure 6.1) 170

Halftone cell divided into two squares (Figure 6.2) 170

Halftone cell and two squares tiled across device space (Figure 6.3) 171

Entries in a type 10 halftone dictionary (Table 6.7) 172

6.3.6 Type 16 Halftone Dictionary 172

Tiling the device space in a type 16 halftone dictionary (Figure 6.4) 173
Entries in a type 16 halftone dictionary (Table 6.8) 173
6.3.7 Type 100 Halftone Dictionary 174
Entries in a type 100 halftone dictionary (Table 6.9) 174
6.4 Extensions to Process Color Models 175
6.4.1 Device-Dependent Color Conversions for ProcessColorModel DeviceN 176
6.5 Transfer Functions 176

6.1 UseCIEColor

When a PostScript language program specifies colors in a device's color space (**DeviceGray**, **DeviceRGB**, **DeviceCMYK**, or a device color space implicitly selected by operators such as **setgray** or **image**), it may be desirable to systematically remap those colors into a CIE-based color space, without altering the original program. This allows any needed color corrections or rendering intents to be supported.

The **UseCIEColor** parameter in the page device dictionary enables this remapping. If the **UseCIEColor** parameter is *true*, all colors that are specified in the **DeviceGray**, **DeviceRGB**, or **DeviceCMYK** color space are remapped into color spaces that must previously have been defined in the **ColorSpace** resource category as **DefaultGray**, **DefaultRGB**, and **DefaultCMYK**, respectively. Their values must be as follows:

- **DefaultGray** must be either a **CIEBasedA** color space array or [/DeviceGray] (no remapping)
- **DefaultRGB** must be either a **CIEBasedABC** color space array, a **CIEBasedDEF** color space array, or [/DeviceRGB] (no remapping)
- **DefaultCMYK** must be either a **CIEBasedDEFG** color space array or [/DeviceCMYK] (no remapping)

If the color space is **DeviceN**, **UseCIEColor** will only affect *alternativeSpace*. If *alternativeSpace* is a device color space, and if **UseCIEColor** is *true*, then the *alternativeSpace* will be remapped.

In the case of **Indexed**, **Separation**, and **Pattern** color spaces, if the underlying color space is a device color space, and if **UseCIEColor** is *true*, then that color space will be remapped. If the color space is **Separation**, **UseCIEColor** will affect only *alternativeSpace*.

Enabling **UseCIEColor** does not alter the current color space or current color values in the graphics state. The remapping of device colors into CIE-based colors is entirely internal to the implementation of the PostScript color operators. Once transformed, the colors are then processed according to the current color rendering dictionary, as is normally done for CIE-based colors.

6.2 Color Rendering

6.2.1 CRD Selection Based on Rendering Intent

CRDs can be selected in a device-independent manner in a page description. This selection accounts for the rendering intent of the reproduction, the device setup, and the current halftone.

A *rendering intent* is information about the rendering of colors in addition to their colorimetric specification. For example, you might want to specify that a scanned image be rendered in a “pleasing” manner, much like a photograph. *Device setup* refers to the state of the device. This information is kept in the page device dictionary and consists of parameters, such as **MediaType** and **HWResolution**. Halftone information is resident in the graphics state.

The **findcolorrendering** operator has been created for the purpose of selecting CRDs by rendering intent. It is not meant for use with LanguageLevel 1 and early LanguageLevel 2 interpreters. Applications, printer drivers, and utilities should test to see if it is known on any product before trying to use it. The syntax of the **findcolorrendering** operator is given in Chapter 8, “Additions and Modifications to the Operators,” on page 193.

The **findcolorrendering** operator should be called subsequent to all commands that influence either the halftone or the device setup to ensure that all parameters that may be considered for selection of a CRD are accounted for correctly.

The following example illustrates the usage of the **findcolorrendering** operator to select a perceptual CRD:

```
/findcolorrendering where
{
  % findcolorrendering available
  pop
  /Perceptual findcolorrendering
  {
    % CRD found which satisfies combination of
    % rendering intent, device setup, and halftone
    /ColorRendering findresource setcolorrendering
  } {
```

```
      % Exact match for CRD not found.
      % Use it, or find a CRD another way.
      % In this example we'll use it if it's
      % not DefaultColorRendering.
      dup
      /DefaultColorRendering eq {
        pop
      } {
        /ColorRendering findresource setcolorrendering
      } ifelse
    } ifelse
  } {
    % findcolorrendering not available.
    % In this example we'll use the current CRD,
    % so we do nothing.
  } ifelse
```

This example first checks for the existence of the **findcolorrendering** operator. If found, it uses the **findcolorrendering** operator to attempt to find a CRD for a perceptual rendering intent. If successful, it installs the CRD in the graphics state. If the **findcolorrendering** operator returns *false*, there are three possible actions:

- Use the substitution CRD that is returned.
- Select a different CRD using your own method.
- Leave the graphics state's current CRD installed.

In this example, a test is first made to see if the instance **DefaultColorRendering** is returned by the **findcolorrendering** operator. In general, this signifies that a useful substitution is not possible. In this case, the best choice is to leave the graphics state's current CRD installed. Installation of the returned CRD is appropriate if the substitution name is different from **DefaultColorRendering**.

The current list of rendering intents recognized by Adobe is kept by the Adobe Developers Association. Note that not all possible rendering intents will be supported by a particular device. In addition, devices may support rendering intents beyond the standard set.

There is purposely a close correspondence between the Adobe rendering intents and the rendering intents of the ICC format. This correspondence is a function of the ICC format version number.

The initial list of recognized rendering intents and their descriptions are shown in Table 6.1.

Table 6.1 *Rendering intents*

Rendering intent	*Description*
AbsoluteColorimetric	Colors are represented with respect to the illuminant. No correction is made for the media's white—for example, unprinted paper. In this case, a monitor's white, which is bluish compared to the printer's paper white, would be reproduced with a blue cast. Given this style of colorimetry, in-gamut colors are reproduced exactly; out-of-gamut colors are mapped to the border of the reproducible gamut. This style of reproduction has the advantage of providing exact color matches from printer to printer. It has the disadvantage of causing colors with *Y* values between the paper's white and 1.0 to be out of gamut. Typical usage would be for logos and solid colors that require an exact reproduction across different media.
RelativeColorimetric	Colors are represented with respect to the combination of the illuminant and the media's white—for example, unprinted paper. In this case, a monitor's white, which is bluish compared to the printer's paper white, would be reproduced by using the unprinted paper. Given this style of colorimetry, in-gamut colors are reproduced exactly; out-of-gamut colors are mapped to the border of the reproducible gamut. This style of reproduction has the advantage of adapting for the media's white. It has the disadvantage of not providing exact color matches between printers with different media. Typical usage would be for vector graphics.
Saturation	Colors are represented in a manner that preserves or emphasizes saturation. In-gamut colors may or may not be colorimetric. Typical usage would be for business graphics or other objects where saturation is the most important attribute of the color.
Perceptual	Colors are represented in a manner that provides a pleasing or perceptual appearance. This generally means both in-gamut and out-of-gamut colors are modified from their colorimetric representation in order to preserve color relationships. Typical usage would be for scanned images.

Relationship to Graphics State Parameters

The only graphics state parameter that is currently considered by the **findcolorrendering** operator is the halftone. Other parameters of the graphics state, such as black generation, undercolor removal, and transfer functions, are not accounted for since they do not require per-object modification. Halftoning can require such modification.

Selecting CRDs by Rendering Intent: The findcolorrendering Operator

The **findcolorrendering** operator forms the name of a CRD from the rendering intent, the device setup, and the halftone. The resulting name takes the form

renderingintent.devicesetup.halftone

where *renderingintent* is taken verbatim from the *renderingintent* operand, and *devicesetup* and *halftone* are found indirectly through procedures resident in the **ColorRendering** instance of the **ProcSet** resource category. *devicesetup* is returned by a call to the **GetPageDeviceName** operator in the **ColorRendering** instance of the **ProcSet** resource category. *halftone* is returned by a call to the **GetHalftoneName** operator in the **ColorRendering ProcSet**. The syntax of the operators **GetPageDeviceName** and **GetHalftoneName** is given in Chapter 8, "Additions and Modifications to the Operators."

Modifying the CRD Selection Process

The **findcolorrendering** operator resides in the dictionary **systemdict** for LanguageLevel 2 products with versions 2015.100 and beyond. For earlier versions of LanguageLevel 2 products, the **findcolorrendering** operator may be downloaded by a utility outside the server loop, or it may be downloaded on a per-job basis. An implementation of the **findcolorrendering** operator and a generic implementation of its associated machinery can be obtained from the Adobe Developers Association. However the operator ends up in the printer, the **findcolorrendering** operator is not meant to be overridden to achieve a modification to the CRD selection process. Its purpose is to delegate portions of this task to the operators **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD**.

It is anticipated that the operators **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD** may be overridden, and that each procedure's specific implementation will vary from device to device to account for the different resident CRDs.

Adobe supplies a baseline version for the operators **GetPageDeviceName**, **GetHalftoneName**, and **GetSubstituteCRD**. Customizing this baseline version can be done on a product-by-product basis. This baseline version satisfies the stated requirements for these three procedures, as well as provides a template for modifying this selection process.

To aid the **GetPageDeviceName** operator in returning meaningful device setup information, Adobe has added to the page device dictionary a **PageDeviceName** key. In general, the **GetPageDeviceName** operator first looks in the page device dictionary for a **PageDeviceName** key. This key's

value can be set using the **setpagedevice** operator. If this key is not present or if its value is *null*, the **GetPageDeviceName** operator constructs a name for the device setup from the current page device parameters—for example, **MediaType** or **MediaClass**—or may simply return /none. This fallback device setup name construction is device dependent and undocumented. You can override this fallback construction by replacing **GetPageDeviceName**.

To aid the **GetHalftoneName** operator in returning meaningful halftone information, Adobe advocates that generators of halftone dictionaries include a **HalftoneName** key. In general, the **GetHalftoneName** operator first looks in the current halftone dictionary for a **HalftoneName** key. If found, this key's value is returned. If not found, the **GetHalftoneName** operator may analyze the current halftone and attempt to form a name or may simply return /none. This fallback halftone name construction is device dependent and undocumented. You can override this fallback construction by replacing **GetHalftoneName**.

6.2.2 Synchronizing CRDs and ICC Profiles

ICC profiles on the host and PostScript CRDs in the printer can contain identical information for color transformations. To reduce printer memory requirements and PostScript file transmission times for color transformations, ICC profiles on the host and CRDs in the printer can be synchronized.

CRDs should be created with a **CreationDate** key indicating the date and time of CRD creation or most recent modification. The **CreationDate** key is optional. This date and time information should correspond to the date and time entry of any companion profiles. A *companion* profile embodies the same transformation, but in a different format—profile versus CRD. Date and time information is available from the profile's header and the **calibrationDateTimeTag** key. Even if no companion profile is constructed, date and time information should still be supplied in the CRD.

The optional CRD key **CreationDate** is a PostScript string whose format closely follows that defined by the international standard ASN.1, defined in CCITT X.208 or ISO/IEC 8824. This string is of the form:

(YYYYMMDDHHmmSSOHH'mm')

where:

YYYY	Year
MM	Month (01–12)
DD	Day (01–31)
HH	Hour (00–23)

mm	Minutes (00-59)
SS	Seconds (00–59)
O	Relationship of local time to Greenwich Mean Time (GMT) (A plus sign (+) indicates that local time is later than GMT, a minus sign (–) indicates that local time is earlier than GMT, and *Z* indicates that local time is GMT.)
HH'	Absolute value of the offset from GMT in hours
mm'	Absolute value of the offset from GMT in minutes

Fields after the year are optional. The default values for day and month are 1; all other numerical fields default to 0. If no GMT information is specified, the relationship of the specified time to GMT is considered to be unknown. Whether or not the time zone is known, the date should be specified based on local time.

Profiles should be extended with the optional ICC key **crdInfoTag**. The **crdInfoTag** key contains the PostScript-product name to which this profile corresponds and the names of the companion CRDs. (Note that a single profile can generate multiple CRDs.)

The format of the **crdInfoTag** key is given in Table 6.2.

Table 6.2 *Format of crdInfoTag*

Byte(s)	*Content*
0–3	‘crdi’ (0x63726469) type descriptor
4–7	Reserved, must be set to 0
8–11	PostScript-product name character count, including terminating null
12–*m*–1	PostScript-product name string in 7-bit ASCII
m–*m*+3	Rendering intent 0, CRD name character count, including terminating null
m+4–*n*–1	Rendering intent 0, CRD name string in 7-bit ASCII
n–*n*+3	Rendering intent 1, CRD name character count, including terminating null
n+4–*p*–1	Rendering intent 1, CRD name string in 7-bit ASCII
p–*p*+3	Rendering intent 2, CRD name character count, including terminating null
p+4–*q*–1	Rendering intent 2, CRD name string in 7-bit ASCII

Table 6.2 *Format of crdInfoTag (continued)*

Byte(s)	*Content*
q–*q*+3	Rendering intent 3, CRD name character count, including terminating null
q+4–*r*	Rendering intent 3, CRD name string in 7-bit ASCII

If no companion CRD is available for a given profile, then the character count entry is zero, and there is no string.

The keys **CreationDate** and **crdInfoTag** can be synchronized differently depending on whether bidirectional communications are available between the host and the output device and whether the CRD was supplied with the output device or was downloaded from a host in the field.

Bidirectional communication allows the output device to be queried to determine the availability of a given CRD and its associated **CreationDate** key. In the absence of bidirectional communications, the list of output device-resident CRDs and their **CreationDate** entries are available through the output device's PPD and the host profile registry.

PPDs currently contain the names of the CRDs that ship with an output device. The PPD format will be extended to contain the **CreationDate** entry for each CRD. The registry should be updated whenever CRDs are downloaded or created in the field. The existence and form of the registry may vary between platforms.

There are three cases to consider:

- *CRDs and ICC profiles are generated together.* The driver determines whether it needs to construct and download a CRD for a given profile as follows:

  The driver optionally checks whether the profile corresponds to the output device by comparing the PostScript-product name field in the **crdInfoTag** key with the output device's product name. The product name for the output device is obtained from the **product** operator or from the PPD. This comparison limits the selection of profiles to only those appropriate for the given output device.

  Based on the desired rendering intent, the driver may check whether the output device has a CRD with the name specified in the **crdInfoTag** key. CRDs are located in the **ColorRendering** resource category. If there is a profile that corresponds to the output device's product name, the driver compares the profile's date and time field to the CRD's **CreationDate** entry. If the two match, it is not necessary to download the profile because

the companion CRD already exists. If no CRD with the name specified in the **crdInfoTag** is found, then CRDs are generated from ICC profiles, as described below.

- *CRDs are generated from ICC profiles and then downloaded.* A driver can download CRDs for a particular job, in which case there will be no companion CRDs for this synchronization for subsequent jobs. Alternatively, the driver can make the CRD persistent in the output device, in which case it will place a **CreationDate** entry in the CRD, update the registry, and update the profile to have the correct **crdInfoTag** information.

- *No profile exists for CRDs in the output device*. This situation occurs primarily with existing CRDs that have no companion profiles. Synchronize a companion profile with CRDs as follows:

  Use the CRD **CreationDate** key, if available, for the date and time field of the profile.

  Alternatively, update the CRD in the output device and registry using the **CreationDate** key corresponding to the date and time field of the new profile.

  In either case, the **crdInfoTag** key must be filled in correctly. Note also that in this case the CRD updates may be volatile to power cycles of the output device. Such power cycles should result in the registry being updated.

6.3 Halftone Dictionaries

For information about the concepts and terms used in this section, see Section 6.4 on page 309 in the *PostScript Language Reference Manual, Second Edition*.



Although the PostScript interpreter can handle 256 gray levels, most printers can print only 32 to 72 gray levels. The operators **setscreen** and **setcolorscreen** and the key **HalftoneType** 1 use very small halftone cells with very few printable gray levels, which can result in banding in blends and contouring in images of faces or of the sky, for example. Cells are defined by a frequency and an angle, and the number of printable gray levels can be estimated using the formula:

$$\textit{Number of grays} = (\textit{Resolution} / \textit{Frequency})^2 + 1$$

All screens defined by spot functions are implemented using a supercell that increases the number of gray levels by a factor of four, to 256 grays. A supercell is an aggregate of four halftone cells that are manipulated as a single unit. The supercell is controlled by the user parameter **MaxSuperScreen** (Table 10.1 on page 246), which sets an upper limit for the

number of pixels in the supercell. The highest effective value for **MaxSuperScreen** is 1016 (254×4) when 256 input levels are printed on bilevel devices. Higher parameter values will not add more printable gray levels, and a change in the parameter value will not affect screens that are already created. The following conditions must be true for the supercell to be created (if any condition is not met, the supercell is not created and the original halftone cell is used):

- The number of pixels in the supercell must be less than or equal to **MaxSuperScreen**.
- The supercell must be within the limits of the **MaxScreenItem** and **MaxScreenStorage** keys.
- The number of pixels in the original halftone cell must be less than the number of usable thresholds.

6.3.1 Changes to the Type 1 through Type 5 Halftone Dictionaries

Table 6.3 describes the entry that has been added to the halftone dictionaries of types 1 through 5.

Table 6.3 *New entry for halftone dictionaries of type 1 through type 5*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.

6.3.2 New Halftone Dictionaries

Several halftone dictionaries have been added to the PostScript language. These Halftone dictionaries are listed in Table 6.4 and described in the subsequent sections.

Table 6.4 *Halftone dictionaries*

Dictionary	
HalftoneTypes 1–5	(*Required*) Described in Section 6.4.3 on page 312 in the *PostScript Language Reference Manual, Second Edition*.
HalftoneType 6	(*Required*) See Section 6.3.3, “Type 6 Halftone Dictionary,” below.
HalftoneType 9	(*Optional*) See Section 6.3.4, “Type 9 Halftone Dictionary,” on page 169.

Table 6.4 *Halftone dictionaries (continued)*

Dictionary	
HalftoneType 10	(*Required*) See Section 6.3.5, “Type 10 Halftone Dictionary,” on page 170.
HalftoneType 16	(*Required*) See Section 6.3.6, “Type 16 Halftone Dictionary,” on page 172.
HalftoneType 100	(*Optional*) See Section 6.3.7, “Type 100 Halftone Dictionary,” on page 174.

6.3.3 Type 6 Halftone Dictionary

The type 6 halftone dictionary defines a halftone screen directly by specifying a threshold array at device resolution. This dictionary is similar to a type 3 halftone dictionary, but the threshold array is obtained from a file instead of a string object. Thus, threshold arrays can be larger than 65535 bytes (the implementation limit for strings), although smaller threshold arrays can also be defined this way.

When presented with a type 6 halftone dictionary, the **sethalftone** operator reads **Width** × **Height** characters from the **Thresholds** file and saves the resulting threshold array in internal storage. A **rangecheck** error is raised if the file does not contain **Width** × **Height** characters.

If the current halftone is a type 6 halftone dictionary, the **currenthalftone** operator returns a halftone dictionary whose **Thresholds** file can be used to access the contents of the current threshold array as if it were a read-only file. (That is, the **Thresholds** file object returned by **currenthalftone** is different from that given to **sethalftone**.) This file treats the contents of the current threshold array as a circular buffer that can be read repeatedly; EOF will never be reached.

Table 6.5 *Entries in a type 6 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.
HalftoneType	integer	(*Required*) Must be 6.
Height	integer	(*Required*) Height of the threshold array, in pixels.

Table 6.5 *Entries in a type 6 halftone dictionary (continued)*

Key	*Type*	*Semantics*
Thresholds	file	*(Required)* When the **sethalftone** operator is used to make a type 6 halftone dictionary the current halftone, the next **Width** × **Height** characters are read from the file referenced by *file* and become the current threshold array. So *file* must reference a file opened for read or read/write access at the time the **sethalftone** operator is called. The *file* object can, of course, be the object returned by the **currentfile** operator. In that case, the next **Width** × **Height** characters are read from the input stream and saved as a threshold array. The **sethalftone** operator closes *file* if it encounters an EOF; otherwise, *file* is left open.
TransferFunction	procedure	(*Optional*) If present, overrides the transfer function specified by the **settransfer** or **setcolortransfer** operator. Required in a type 6 halftone dictionary that is used as an element of a type 5 halftone dictionary for a nonprimary color component.
Width	integer	(*Required*) Width of the threshold array, in pixels.

6.3.4 Type 9 Halftone Dictionary

The type 9 halftone dictionary defines a halftone whose data is proprietary. This type of halftone will be present only in those products whose manufacturers have specifically requested this type of support. If it is not present, attempting to set a type 9 halftone will result in a PostScript language error. It is not possible for PostScript code to gain any information about the contents or appearance of a type 9 halftone. As a general rule, an application should not explicitly attempt to set a type 9 halftone. If one is present, it will usually be the default halftone; consequently, any printing that an application does will take advantage of it (unless the application performs its own **sethalftone** call). If it is important to determine whether a type 9 halftone is being used, check the **HalftoneType** key for the dictionary returned by the **currenthalftone** operator.

Table 6.6 *Entries in a type 9 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.
HalftoneType	integer	(*Required*) Must be 9.

6.3.5 Type 10 Halftone Dictionary

The type 10 halftone dictionary can be used to specify a threshold array that represents a halftone cell with a non-zero screen angle. Either the type 3 or type 6 halftone dictionary can be used to specify a threshold array representing a zero-angle halftone cell, but there is no provision for other angles. Zero-angle halftone cells are easy to specify because they line up well with scan lines and because it is not difficult to determine where a sampled point goes. The type 10 halftone applies a simple transformation to the halftone cell that converts it into two squares, thus making it easier to specify non-zero angle cells.

Figure 6.1 and Figure 6.2 illustrate a halftone at 300 dpi with a frequency of 38.4 and an angle of 50.2 degrees. Figure 6.1 shows how this halftone cell can be graphically represented in device space; each asterisk is an (*x*,*y*) coordinate in device space that is mapped to a specific location in the threshold array. Figure 6.2 shows how this cell can be divided into two squares.

Figure 6.1 *Halftone cell graphically represented in device space*

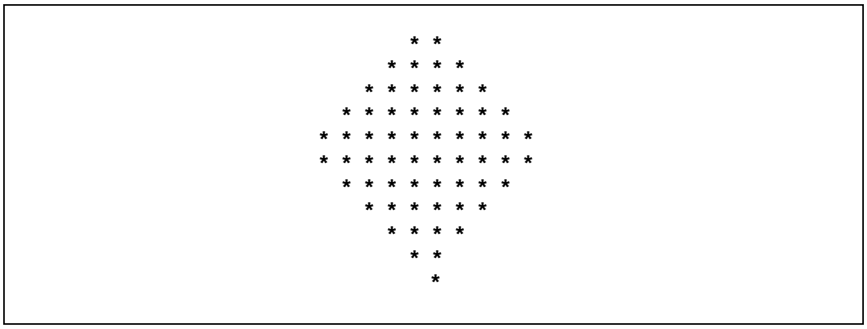
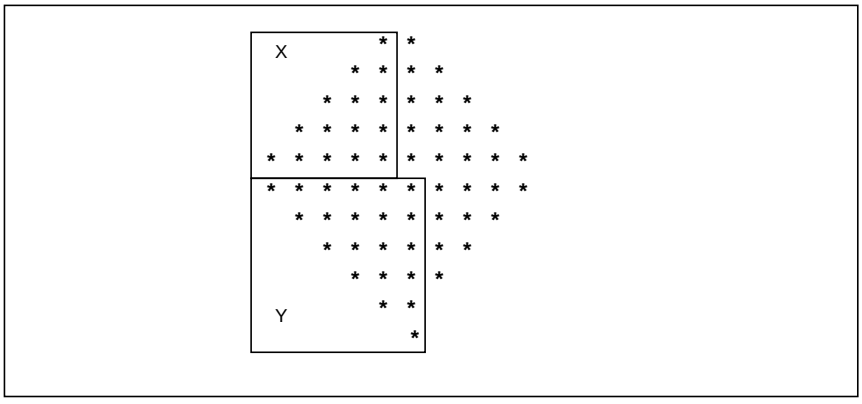
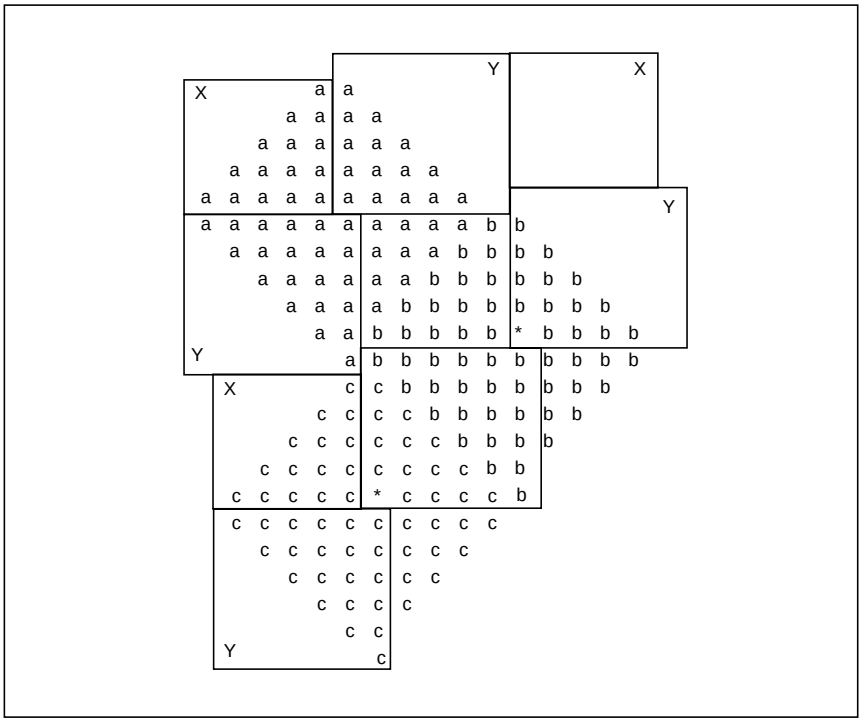


Figure 6.2 *Halftone cell divided into two squares*



If the two squares and the original halftone cell are tiled across device space, the area to the right of the upper square exactly maps into the empty area of the lower square, and the area to the right of the lower square exactly maps into the empty area of the upper square, as Figure 6.3 shows.

Figure 6.3 *Halftone cell and two squares tiled across device space*



Any halftone cell will map into two squares in this fashion. The length of one side of the upper X square will equal the distance along the *x* axis from a point in one halftone cell to the corresponding point in the adjacent cell (the asterisks in Figure 6.3 show two such corresponding points). The length of the lower Y square will equal the distance along the *y* axis between the same points. In other words, the frequency of a halftone constructed from two squares *X* and *Y* (as described above) is:

$$frequency = \frac{resolution}{\sqrt{X^2 + Y^2}}$$

and the angle is:

$$angle = \text{atan}\left(\frac{Y}{X}\right)$$

The two squares are much easier to handle and store than the original cell, yet the squares contain the same information. In addition, the squares can be easily mapped into the internal representation used for all rendering.

Table 6.7 lists the entries in a type 10 halftone dictionary.

Table 6.7 *Entries in a type 10 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.
HalftoneType	integer	(*Required*) Must be 10.
Xsquare	integer	(*Required*) Length of one side of the upper square.
Ysquare	integer	(*Required*) Length of one side of the lower square.
Thresholds	string or file	(*Required*) Specified as either a *string* or *file* object, as with type 3 or type 6 halftones, respectively, and subject to the same rules. If it is a *string*, it must be **Xsquare** × **Xsquare** + **Ysquare** × **Ysquare** bytes in length. If it is a *file*, the stream must contain at least that many bytes (as with type 6 halftones, it is not necessary for a stream to reach the EOF precisely after the requisite number of bytes). In either case, the value of **Thresholds** is ordered with the value of **Xsquare** square first. The order of pixels is the same as for a sampled image mapped directly onto device space, with the first sample at device coordinates (0, 0) and with *x* coordinates changing faster than *y* coordinates. Note that this is the same ordering used by type 3 and type 6 threshold arrays. The value of **Ysquare** immediately follows that of **Xsquare** and is laid out in the same fashion. The **currenthalftone** operator will return the original dictionary if the value of the **Thresholds** key is a *string* (as for a type 3 halftone). If the **Thresholds** value in the original dictionary is a *file*, it will be replaced by a new *file* linked to that threshold array (as with a type 6 halftone) in the dictionary returned by the **currenthalftone** operator.
TransferFunction	procedure	If present, overrides the transfer function specified by the **settransfer** or **setcolortransfer** operator.

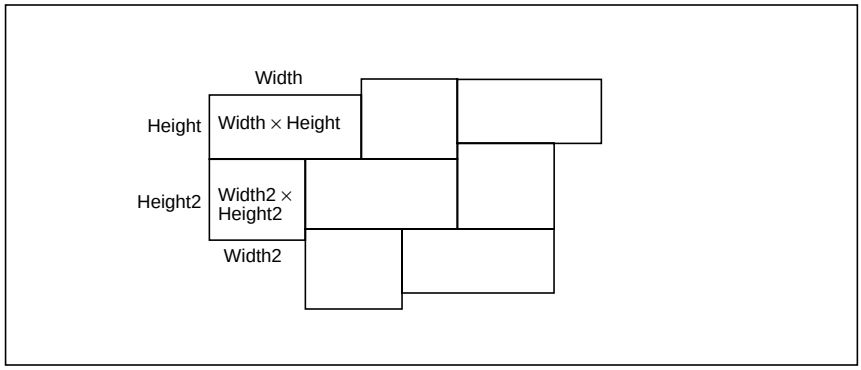
6.3.6 Type 16 Halftone Dictionary



As with a type 6 halftone dictionary, the type 16 halftone dictionary defines a halftone screen directly by specifying a threshold array at device resolution. In a type 16 halftone dictionary, however, each element of the threshold array is 16 bits wide rather than 8 bits wide. This allows the threshold array to specify 65,536 levels of color rather than 256 levels.

The threshold array can consist of either one or two rectangles. The first or only rectangle is defined by **Width** × **Height**; the second, optional rectangle, is defined by **Width2** × **Height2**. If two rectangles are specified, the rectangles will tile the device space as shown in Figure 6.4. The last row in the first rectangle is immediately adjacent to the first row in the second rectangle, and the rows start in the same column.

Figure 6.4 *Tiling the device space in a type 16 halftone dictionary*



When presented with a type 16 halftone dictionary, the **sethalftone** operator reads the first **Width** × **Height** × 2 characters from the **Thresholds** file and then, if a second rectangle is defined, reads **Width2** × **Height2** × 2. The operator saves the resulting threshold array in internal storage. If the **Thresholds** file ends prematurely (that is, not enough data is supplied), a **rangecheck** error is raised. The **sethalftone** operator closes the **Thresholds** file if it encounters an EOF; otherwise, it leaves it open.

If the current halftone is a type 16 halftone dictionary, the **currenthalftone** operator returns a halftone dictionary whose **Thresholds** file refers to the current threshold array in internal storage. That is, the **Thresholds** file object returned by the **currenthalftone** operator may be different from that given to the **sethalftone** operator. This file in the current halftone dictionary is a nonreadable file, and its access attribute is noaccess. If used in a call, the **sethalftone** operator will recognize this file in the current halftone dictionary as a reference to an internally held threshold array.

Table 6.8 *Entries in a type 16 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.
HalftoneType	integer	(*Required*) Must be 16.
Height	integer	(*Required*) Height of the rectangle (or the first rectangle if there are two) in the threshold array, in pixels.
Height2	integer	(*Optional*) Height of the optional second rectangle in the threshold array, in pixels. If the **Height2** key is specified, then the **Width2** key must also be specified; if the **Height2** key is omitted, the **Width2** key cannot be specified and the threshold array has no second rectangle.

Table 6.8 *Entries in a type 16 halftone dictionary (continued)*

Key	*Type*	*Semantics*
Thresholds	file	(*Required*) Must hold **Width** × **Height** × 2 (or **Width** × **Height** × 2 + **Width2** × **Height2** × 2) characters of threshold data, where each threshold is 16 bits wide. The high-order character of a threshold is stored first. The file object can be that returned by the **currentfile** operator.
TransferFunction	procedure	If present, overrides the transfer function specified by the **settransfer** or **setcolortransfer** operators.
Width	integer	(*Required*) Width of the first or only rectangle in the threshold array, in pixels.
Width2	integer	(*Optional*) Width of the optional second rectangle in the threshold array, in pixels. If the **Width2** key is specified, then the **Height2** key must also be specified. If the **Width2** key is omitted, then the **Height2** key cannot be specified and the threshold has only one rectangle.

Note ***Width2*** *and* ***Height2*** *must both be defined, or both must be omitted. If both are omitted, the threshold array has only one rectangle.*

6.3.7 Type 100 Halftone Dictionary

The type 100 halftone dictionary is almost the same as a type 9 halftone dictionary, with the exception that the designer of the dictionary may include additional keys. These keys may be used by the printer firmware to specify different halftones under different circumstances. As with a type 9 halftone, it is impossible for PostScript code to determine anything about the contents or appearance of the type 100 halftone. While any optional keys will be visible in the dictionary returned by the **currenthalftone** operator, there is no way to know what they control or what the permissible values are. As a result, an application should not explicitly attempt to set a type 100 halftone. If it is important to determine whether a type 100 halftone is being used, check the **HalftoneType** key of the dictionary returned by the **currenthalftone** operator.

Table 6.9 *Entries in a type 100 halftone dictionary*

Key	*Type*	*Semantics*
HalftoneName	name or string	(*Optional*) If present, supplies the name of the halftone dictionary to the **findcolorrendering** operator that forms the name of a CRD. The **HalftoneName** key is used by the **GetHalftoneName** operator that is part of the **ColorRendering ProcSet**.
HalftoneType	integer	(*Required*) Must be 100.
other	any type	(*Optional*) Product specific.

6.4 Extensions to Process Color Models



The **ProcessColorModel** key specifies the model used for rendering process colors in a device. The legal values for **ProcessColorModel** are extended to include /DeviceN. This value is used for a device where the process color model is not described by the standard process color models. The **DeviceN** color space can be used in a page description regardless of whether the value of **ProcessColorModel** is /DeviceN. The former is for color specification; the latter is for color rendering. The **ProcessColorModel** key is described in Table 4.17 on page 112.

If a device cannot resolve a combination of **ProcessColorModel** /DeviceN and **SeparationColorNames** requests, it may revert to a previous state, or (for some capable printing systems) to some new state, where it can convert all color spaces. All color spaces will always be printed; however, the colorants used may differ from those requested.

When the value of **ProcessColorModel** is /DeviceN, the names of the device colorants must all be given explicitly in the **SeparationColorNames** array. There are no defaults as there are for the standard process color models. (The **SeparationColorNames** key is described in Table 4.17 on page 113.)

A device that supports more than four colors could do so using either a combination of a standard process color model and some spot colors or a **DeviceN** color model. The results are different, however. For example, consider a six-color device where the colorants are cyan, magenta, yellow, black, orange, and green. This could be modeled in one of two ways:

- *The **ProcessColorModel** is **DeviceCMYK** and two additional colorants orange and green are added as spot colors.* In this case, any colors specified in device-dependent color spaces (**DeviceGray**, **DeviceRGB**, and **DeviceCMYK**) are transformed into **DeviceCMYK** color space using the existing color conversion rules. The same is true for output produced by a type 1 CRD. Only the orange and green colorants would need to be listed in the **SeparationColorNames** array; cyan, magenta, yellow, and black would be automatically included because the value of **ProcessColorModel** is /DeviceCMYK.

- *The **ProcessColorModel** is **DeviceN,** where N = 6.* In this case, colors specified in the device-dependent color spaces or by a type 1 CRD are converted to the **DeviceN** colorants in an implementation-specific manner. All six of the colorants must be listed in the **SeparationColorNames** array.

In both cases, source colors in the **DeviceN** color space would produce the same results if they use only colorants listed in the **SeparationColorNames** array.

6.4.1 Device-Dependent Color Conversions for ProcessColorModel DeviceN



A device can support device-dependent color conversions when the **ProcessColorModel** is **DeviceN**, by using either of the following:

- The **UseCIEColor** key to map the device-dependent color specifications to device-independent color specifications. CRDs are subsequently used to describe the device's color characteristics.

- Custom conversion routines that map the device-dependent color specifications to the specific device colorants—for example, a custom conversion from **DeviceCMYK** color space to cyan, magenta, yellow, black, orange, and green associated with PANTONE Hexachrome.

The use of undercolor removal and black generation on a device with a **ProcessColorModel** of **DeviceN** is implementation specific. For example, a **DeviceN** device where *N = 4* might use the above conversion routines to transform from **DeviceRGB** color space to the native color space of the device. These routines are normally used to transform from **DeviceRGB** color space to a device with the **DeviceCMYK** process color model.

For any device, a job may request that the device behave as a **DeviceN** device with a certain set of colorants, as specified in **SeparationColorNames**. In this case, a decision is made at the execution of **setpagedevice** as to whether the device can behave as the requested **DeviceN** device. The **setpagedevice** operator will succeed if there are custom conversion routines available to map all device-dependent color spaces to the requested **DeviceN** colorants of the device.

6.5 Transfer Functions



If the **ProcessColorModel** is **DeviceN**, transfer functions are set using the **TransferFunction** key in the halftone dictionary for each named component. The transfer functions established by **settransfer** or **setcolortransfer** are not used in this case. Transfer functions are not intended for page descriptions; rather, they are intended for describing device-specific behavior.

For further information on transfer functions, see Section 6.3 in the *PostScript Language Reference Manual, Second Edition*, and Section 3.4, "Functions," on page 56 of this *Supplement*.

CHAPTER 7

Additions and Modifications to Display PostScript®

This chapter supplements Chapter 7 in the *PostScript Language Reference Manual, Second Edition.* It provides information about the type 2 image dictionary and the device pixel color space.

Overview of This Chapter

7.1 Type 2 Image Dictionary 178

7.1.1 Uses of Type 2 Images 178

7.1.2 Entries in a Type 2 Image Dictionary 179

Entries in a type 2 image dictionary (Table 7.1) 179

7.1.3 Type 2 Image Dictionary Extensions 180

UnpaintedPath 180

DataSource 181

Imaging source sample data into a destination device (Figure 7.1) 182

Coordinate Implementation Transformation Details 183

Transforming from image space to current user space (Figure 7.2) 183

PixelCopy 184

7.2 The Device Pixel Color Space 185

7.2.1 Drawing with a Device Pixel Color Model 185

Drawing with a typical display system's color mechanism (Figure 7.3) 186

7.2.2 Drawing with the Display PostScript Color Model 186

Drawing with the Display PostScript color mechanism (Figure 7.4) 187

7.2.3 Specifying Custom Colors 187

7.2.4 Performing Color Map Animation 188

7.2.5 Coloring Overlapping Areas 189

Frame buffer with overlapping areas rendered on screen (Figure 7.5) 189

X window system drawing functions and their logical operations (Table 7.2) 190

Two irregular overlapping areas (Figure 7.6) 191

Example colors and index assignments for OR drawing function (Table 7.3) 192

7.2.6 Device Pixel Color Space and Layered Windows 192

7.1 Type 2 Image Dictionary

The image dictionaries have been expanded to include a type 2 for the Display PostScript system. The image dictionary documented in Section 4.10.5 in the *PostScript Language Reference Manual, Second Edition*, is a type 1 image dictionary. In the following sections, the term *pixmap* refers to a window-system specific memory buffer.

The type 2 image dictionary allows the **image** operator to use pixel data from a pixmap, the current window, or another window as source when copying into the current window. The source need not have been produced by a PostScript interpreter.

A **typecheck** error is raised if a type 2 image dictionary is used with the **imagemask** operator.

7.1.1 Uses of Type 2 Images

In the Display PostScript system, extending the **image** operator with type 2 image dictionaries provides the following capabilities:

- *Copying a rectangular area from one window to another.* Although an application could use the native window system's pixel copy feature to copy pixels (as with **XCopyArea** in Xlib, for example), the **image** operator requires fewer calculations. The pixel copy approach requires the application to calculate pixel coordinates in both the native window coordinate system and the PostScript coordinate system. The pixel copy approach might also require the application to track events from the window server (as is the case with **XCopyArea**). Using the **image** operator limits the calculations to the PostScript coordinate system.

- *Reducing communication costs by working with image data already accessible to the window server in a pixmap*. This approach improves performance because the image data does not need to be transferred repeatedly from the client to the server. Reducing client-server communications is especially important if the server executes on a different system or processor from the client. For example, a Display PostScript application can use the **image** operator to copy pixels from a pixmap or from another window on the screen into the current window.

- *Copying a source image from one window into a second window that has a different pixel representation*. The **image** operator can copy pixels from one pixel representation to another, applying the proper color conversion.

- *Presenting large photographic images, such as satellite data*. The application could store the image in a pixmap and use the **image** operator for panning and zooming. The **image** operator can scale and rotate source

data from the pixmap into the viewing window. The application could pan by copying selected parts of the data from the off-screen window to the viewing window; it could zoom by simultaneously copying and scaling the data. In such an application, scaling could permit all of the image data to be seen when necessary.

- *Magnifying off-screen pixels for editing in the current window with a bitmap editor.* After the user edits the magnified pixels, the source sample data in the pixmap can be changed to reflect the edits.

- *Using transfer functions to tune the colors in sampled image data*. An application could display an image and provide a slider for adjusting brightness. When the user moves the slider, the transfer function could change, causing the window to be re-imaged in place or from a pixmap.

7.1.2 Entries in a Type 2 Image Dictionary

Table 7.1 defines the type 2 image dictionary keys, some of which are also present in type 1 image dictionaries.

Table 7.1 *Entries in a type 2 image dictionary*

Key	*Type*	*Semantics*
ImageType	integer	(*Required*) Must be 2. Specifies a type 2 image dictionary.
DataSource	gstate	(*Required*) A graphics state object that represents the source sample data.
XOrigin	real	(*Required*) *X* origin of source rectangle in the user space coordinate system specified by the transformation in the **DataSource** graphics state object.
YOrigin	real	(*Required*) *Y* origin of source rectangle in the user space coordinate system specified by the transformation in the **DataSource** graphics state object.
Width	real	(*Required*) Width of source rectangle in the user space coordinate system specified by the transformation in the **DataSource** graphics state object.
Height	real	(*Required*) Height of source rectangle in the user space coordinate system specified by the transformation in the **DataSource** graphics state object.
UnpaintedPath	various	(*Read Only*) If this key is present and portions of the output page could not be painted because some source pixels were not available (for example, because of clipping), then the **image** operator will return a user path in the current (destination) user space coordinates that encloses the area that could not be painted. If all of the source pixels that were needed were successfully accessed, the **UnpaintedPath** key will contain a null object.

Table 7.1 *Entries in a type 2 image dictionary (continued)*

Key	*Type*	*Semantics*
PixelCopy	boolean	(*Optional*) If the value is *true*, indicates that the source pixels should be copied directly, without going through the normal color conversion, transfer function, or halftoning. The number of bits per pixel of the source must match the number of bits per pixel of the destination; otherwise, a **typecheck** error will occur. If the **PixelCopy** key is either *false* or absent, the pixels will be imaged in the usual way, including the application of color conversion, transfer functions, and halftone screens.
ImageMatrix	array	(*Required*) An array of six numbers that define a transformation from current user space to image source space.

7.1.3 Type 2 Image Dictionary Extensions

This section provides more detail about the **UnpaintedPath**, **DataSource**, and **PixelCopy** keys in the type 2 image dictionary and gives coordinate implementation transformation details.

UnpaintedPath

Depending on the representation of the source data, portions of the output page may not be paintable because the specified source data is not available. For example, in a Display PostScript application the source rectangle may include areas of the source window that are obscured by an overlapping window. The **UnpaintedPath** key provides a way for the **image** operator to return information that specifies the unpainted region.

If the **UnpaintedPath** key is present in the type 2 image dictionary, the **image** operator will replace its value with a user path object that represents the area of the destination that was not painted due to unavailable source data. If all of the source data was available, the **UnpaintedPath** key will contain a null object.

The returned user path can be used as a clip in the destination graphics state to restrict painting to areas that need to be updated. After the clip has been set, the PostScript code that created the source can be rendered in the destination to paint the area that was left unpainted by the **image** operator. The unpainted path is guaranteed to return all parts of the image that were not painted; however, it may in some circumstances return parts of the image that actually were painted.

A common use of the **image** operator is to map the source samples to the unit square with **ImageMatrix** and scale the CTM to the correct width and height. When this method is used, the scaling of the CTM needs to be undone immediately afterwards so that the PostScript program image is rendered at the proper scale.

The techniques just discussed are illustrated in Example 1. The code takes a source rectangle from one window and rotates it in the current window. Pixels that were not available in the source are then repainted with the PostScript procedure **DrawSource**, which created the original source.

Example 1 *Rotating from source to the current window*

```
50 50 translate
45 rotate
200 100 scale
/imageDict <<
  /ImageType 2
  /XOrigin 50
  /YOrigin 50   % origin of source rectangle in
                % mysourcewindow
  /Width 200
  /Height 100   % size of source rectangle
  /ImageMatrix [200 0 0 100 0 0]  % mapping to unit square
  /DataSource mysourcewindow   % gstate of the source window
  /UnpaintedPath null
>> def

imageDict image
imageDict /UnpaintedPath get xcheck {
  gsave
  imageDict /UnpaintedPath get   % get the missing area
  newpath
  uappend clip                   % set the clip
  1 200 div 1 100 div scale      % undo the scale
  DrawSource                     % repaint the unpainted area
} if
```

Note *Take care when using user path operators, such as **uappend**, because the CTM may be altered during the execution of the operator. For further information, see Section 4.6.4 in the PostScript Language Reference Manual, Second Edition.*

DataSource

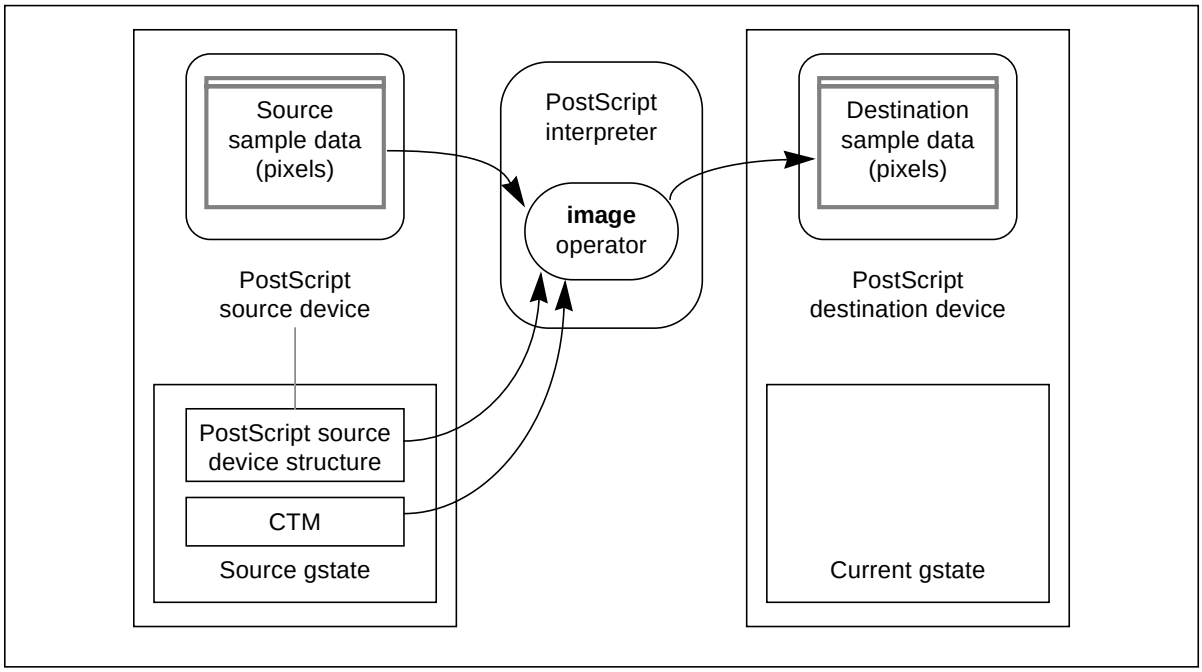
The **DataSource** key holds the address of a graphics state object, called the "source gstate" in the following discussion.

The source gstate includes a reference to the PostScript device that holds the source pixel values. The source pixel values and their representation are known only to the implementation of the PostScript device.

The only source gstate information used by the **image** operator is the CTM and the PostScript source device structure. In particular, the color space in the source gstate is ignored by the **image** operator. The **image** operator uses the color space of the source PostScript device (referenced by the source gstate)

to identify the meaning of the source sample data. For example, if the source device is an RGB device, then the color space used by the **image** operator is **DeviceRGB**.

Figure 7.1 *Imaging source sample data into a destination device*



No assumptions are made by the PostScript interpreter about how the image source sample data was created. The client is responsible for obtaining the desired results from the application of the **image** operator to the given source sample data (including use of the current transfer function and halftone screen, if desired).

If the source gstate is the current graphics state and the source rectangle overlaps the destination, the image will be rendered as if all of the source pixels had been read before imaging.

If the source is a pixmap, then the user must supply Display PostScript with a color map for that pixmap. This should be done with **setdrawablecolor**, a window-specific PostScript custom operator. Without a color map, the **image** operator will not be able to interpret the pixel values in the pixmap, and a **typecheck** error will occur. A color map is not necessary if the **PixelCopy** entry is used. See the section “PixelCopy” on page 184.

The PostScript **image** operator restricts the bits per component to 1, 2, 4, 8, or 12. This restriction also applies to the **image** operator for type 2 image dictionaries. Therefore, a gray device that is a source must be either 1, 2, 4, 8, or 12 bits per pixel. Similarly, an RGB device that is a source must be either

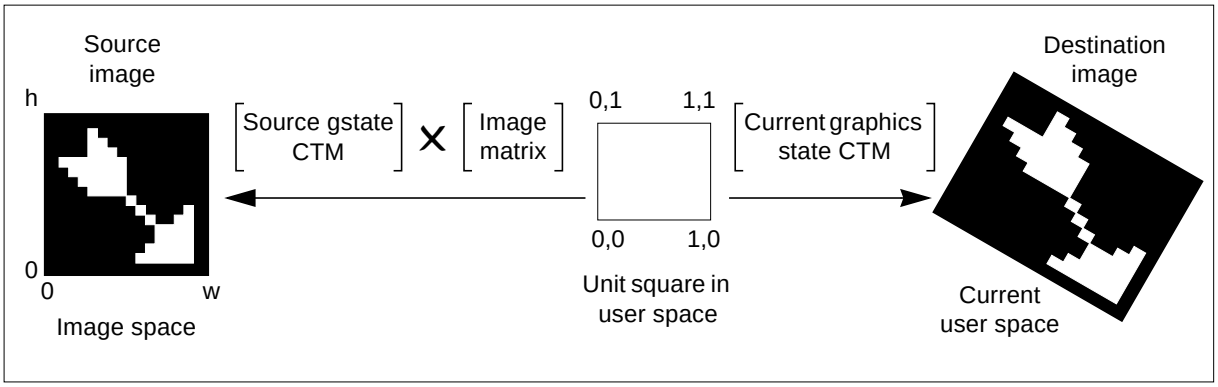
3, 6, 12, or 24 bits per pixel. These sizes see the significant portions of the pixel used for color data, not the pixel size itself. For example, a 24-bit pixel stored in a 32-bit word is acceptable.

Coordinate Implementation Transformation Details

Using the **image** operator with graphics state sources is similar to using the **image** operator with any other sample data. One notable difference is how image space (the source image coordinate system) is converted to current user space. Because the image space is identical to the source gstate's user space coordinate system, the source gstate's CTM is used in the transformation. Therefore, the mapping from the image space to the current user space involves the use of three matrices: the image matrix, the source gstate's CTM, and the current graphic state's CTM.

Figure 7.2 illustrates this transformation. The source gstate's CTM and the image matrix can be used to map the image space to a unit square of image space, and the current graphic state's CTM maps the unit square to the current user space.

Figure 7.2 *Transforming from image space to current user space*



Although these transformation details are largely hidden from the application programmer, they point out that any alteration of the source gstate CTM will affect the transformation from image space to user space. For further information, see Section 4.10 in the *PostScript Language Reference Manual, Second Edition*.

Example 2 implements a simple copy from the window referenced by the graphics state `mysourcewindow` to the current window. No scaling or other transformation is performed during the copy—the location and size of the image in the current window are the same as in the source window.

Example 2 *A simple copy to the current window from a different window*

```
50 50 translate  % location of destination in current window
200 100 scale    % size of destination
<<
  /ImageType 2
  % origin of source rectangle in mysourcewindow
  /XOrigin 50
  /YOrigin 50
  /Width 200
  /Height 100                  % size of source rectangle
  /ImageMatrix [200 0 0 100 0 0]% mapping to unit square
  /DataSource mysourcewindow   % gstate of the source window
>>   image
```

Example 3 uses the current graphics state as the source and scales the image. The effect is similar to zooming in the current window.

Example 3 *Scaling an image to mimic zooming*

```
<<
  /ImageType 2
  /XOrigin 0
  /YOrigin 0
  /Width 500
  /Height 400
  % scale and center the image
  /ImageMatrix [2 0 0 2 –50 –200] matrix invertmatrix
  /DataSource mycurrentgstate %gstate of the current window
>>   image
```

PixelCopy

The value of the **PixelCopy** key can be set to *true* to disable the **image** operator's application of color conversion, transfer functions, and halftone screens. Color conversion is often disabled—for example, when copying source pixels directly onto the output page from a previously rendered (and color-converted and halftoned) pixmap. This approach is especially useful when the source pixels do not represent a PostScript device color space.

PixelCopy is a fast form of the **image** operator for type 2 dictionaries because it involves no pixel conversion. When no scale or rotation is required between source and destination, setting **PixelCopy** to *true* can result in an even greater performance advantage. The reason for the performance boost is that the operator can use the native window system's copy function, which in some cases is assisted by hardware. However, if the window system's copy function is used and the window system is not using the same rasterization rules as the Display PostScript system, a difference of as much as one pixel in image size or location may result. In many cases, the performance gain is worth the possible error in pixelization.

PixelCopy accepts pixel sizes of 1, 2, 4, 8, 12, and 32 bits. Unlike other forms of the **image** operator, **PixelCopy** doesn't interpret the color data. The pixel sizes refer only to the number of bits used to store each pixel. So, for example, a 24-bit pixel stored in a 32-bit word is acceptable because the actual number of bits per pixel in the frame buffer is 32.

7.2 The Device Pixel Color Space

The device pixel color space applies mostly to PostScript interpreters running in an interactive display system—for example, the NeXTSTEP™ window server, where all graphic drawing to the screen is done using the Display PostScript system.

The device pixel color space allows an application to specify pixel color values directly with a display system's color rendering mechanism, bypassing the more abstract Display PostScript color rendering mechanism. This new color space greatly simplifies certain low-level tasks such as animating color maps and filling the overlapping areas of two colored objects with a third color.

This functionality can greatly simplify certain CAD applications where circuit traces are applied on top of one another and are selectively undrawn when a circuit trace is removed.

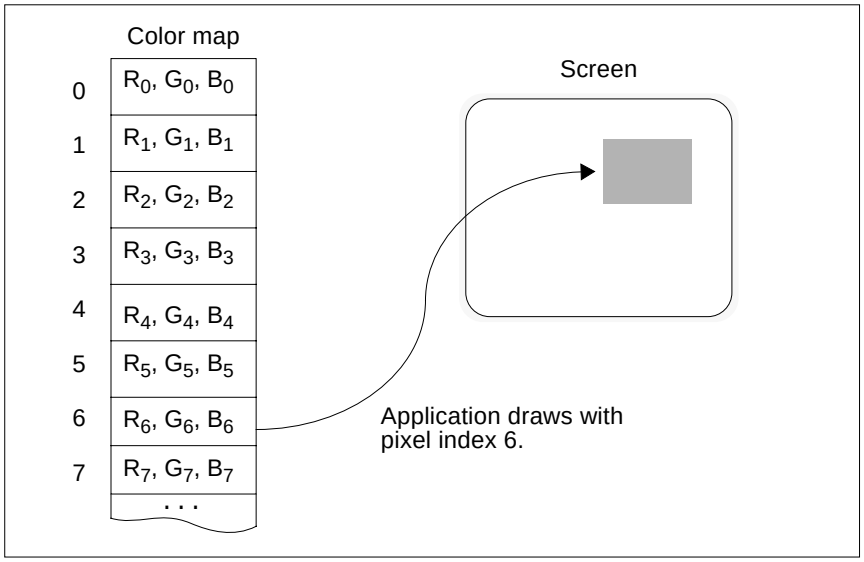
No conversion is allowed between the device pixel color space and other color spaces.

7.2.1 Drawing with a Device Pixel Color Model

A typical display system's color model designates an area of memory as a color map. The color map contains ordered triples of RGB values. Each value is referred to by a pixel index number, typically between 0 and 255, allowing an application to specify any of 256 colors.

An application draws in a designated color by storing the RGB values of the color in a color map cell and then drawing with the pixel index of that cell. For example, assume that pixel index 6 contains RGB values for green. An application could then set the pixel index to 6 and draw a green object on the screen as shown in Figure 7.3.

Figure 7.3 *Drawing with a typical display system's color mechanism*



A color map with pixel indexes ranging from 0 to 255 limits an application to 256 different colors at any one time. But because the RGB values of any pixel index can be changed, an application can draw using any color. For example, a color map could range from an index 0 value of black (R, G, B = 0, 0, 0) to an index 255 value of red (R, G, B = 1, 0, 0). In between, the map could contain 254 shades of deep red and no other colors.

7.2.2 Drawing with the Display PostScript Color Model

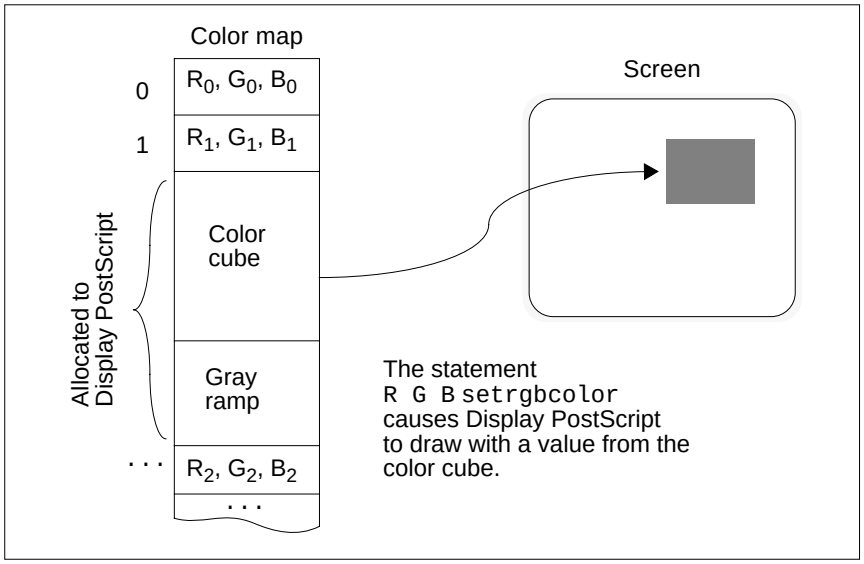
For color and grayscale rendering, the Display PostScript system normally designates a portion of the color map as a storage area for its color cube and gray ramp. The size of the color cube and gray ramp can be set by the user.

Before an application draws with the Display PostScript system, it first sets a color value with a statement such as

```
0.2 0.3 0.4 setrgbcolor
```

This statement causes the Display PostScript system to search in the color cube and find a value that matches the desired RGB value. If the color cube is small and does not contain the exact value specified, the system approximates the specified value with halftones. When the application draws an object on the screen, the object is rendered in either the specified RGB value or a halftone approximation, as shown in Figure 7.4 on page 187.

Figure 7.4 *Drawing with the Display PostScript color mechanism*



The color rendering mechanism in the Display PostScript system is fundamentally more abstract and sophisticated than that of a display system. Typically, an application allocates a cell in the color map, stores an RGB value in the cell, and refers to the cell by index number whenever it draws with that RGB value. In the Display PostScript system, the application does not manipulate the color cube after it has been set up. Instead, colors are specified with color operators such as **setrgbcolor**, **sethsbcolor**, **setcmykcolor**, or **setgray**; or with the pair of operators **setcolorspace** and **setcolor**.

7.2.3 Specifying Custom Colors

The device pixel color space applies a display system's color model to the Display PostScript system so that objects can be drawn with a specified color pixel index, bypassing the Display PostScript color rendering mechanism. The syntax for setting the color of a drawn area with the device pixel color space is:

```
[/DevicePixel bits-per-pixel] setcolorspace
pixel-index setcolor
```

After the color pixel index has been specified, drawn objects will appear in the color value of the index.

Perhaps the most obvious use of the device pixel color space is to specify a color directly, and then use the specified color to fill an object.

Rather than simply specifying a custom color, the device pixel color space should be used for color map animation and for situations in which direct control of pixel values is required, as when computing the color of overlapping areas with pixel value arithmetic. These uses of device pixel color space are discussed in the following sections.

7.2.4 Performing Color Map Animation

The directness of a display system's color drawing model is an advantage for certain low-level operations. For example, assume an application has drawn an area on the screen using the value of pixel index 6. If the application subsequently stores a different RGB value at index 6, the drawn area on the screen changes color instantly to reflect the new value—no calculation or repainting is required. In order to perform *color map animation* on the area (automatically display it in a changing array of colors), the application allocates the cell at index 6 as read-write, and then changes its RGB value.

The Display PostScript system does not directly support color map animation. To change the color of an area on the screen, the Display PostScript model requires that the area's boundary first be recalculated, and then redrawn and repainted.

Although color map animation is easy with most display systems, such systems are not adept at drawing complex shapes—areas bounded by Bézier curves, for example. In contrast, the ability to draw complex shapes is a basic capability of the Display PostScript system. The device pixel color space applies a display system's color model to the Display PostScript system so that irregular shapes can be animated with color.

For example, assume an application is running on an 8-bit color system. In order to draw an area on the screen and animate the color, the application first calls on the display system to allocate a cell in its color map. The display system returns the pixel index number of the cell that has been allocated—index number 7, let us say. The application stores the initial RGB color value into cell 7. Next, the application sets the color space to **DevicePixel** and the color to 7. It then draws with Display PostScript operators, and the drawn area is filled with the RGB value of index 7. The application then animates the color of the object by writing new values to index 7. The following outline summarizes the process:

Allocate read-write color map cell with the display system.
The display system returns index 7.
Set initial color of color map cell 7.
```
[/DevicePixel 8] setcolorspace
7 setcolor
```
Draw area with Display PostScript operators.
Animate color by writing new values into color map index 7.

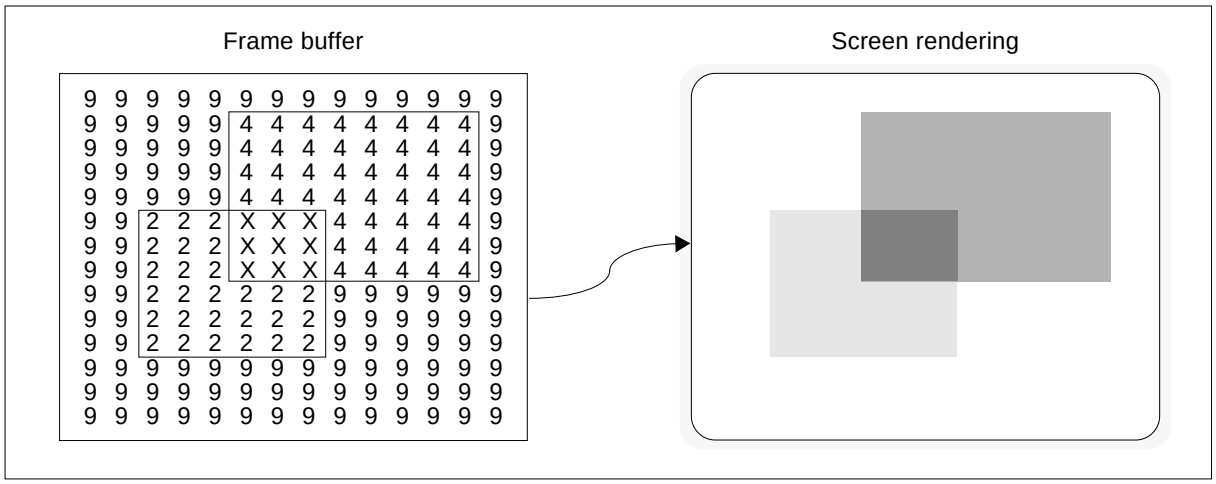
Note that the application is responsible for using the display system calls to store the desired RGB values in the color map at pixel index 7.

7.2.5 Coloring Overlapping Areas

Consider a frame buffer loaded with pixel indexes, as shown in Figure 7.5. As each index is read out of the frame buffer, system hardware dereferences it through a color map and renders the corresponding pixel with the RGB value of the index.

The frame buffer shown in Figure 7.5 holds two overlapping rectangles, one the color of pixel index 2, the other the color of pixel index 4. In a typical display system, the color of the overlapping area (indicated by X's in the frame buffer in Figure 7.5) is determined by a *drawing function*.

Figure 7.5 *Frame buffer with overlapping areas rendered on screen*



Most programs use a display system's drawing function `copy` to color overlapping areas. This function colors overlapping areas with the color of the last-drawn object (known as the *source*). Other programs use the `noop` function to preserve the previous color (known as the *destination*). Other drawing functions perform logical operations on the pixel indexes of the source and destination.

In the *X* window system, a total of sixteen *X* drawing functions perform logical operations on pixel indexes. Most of them are intended for

monochrome displays and are not useful in color environments. Their logical operations are summarized in Table 7.2.

Table 7.2 *X window system drawing functions and their logical operations*

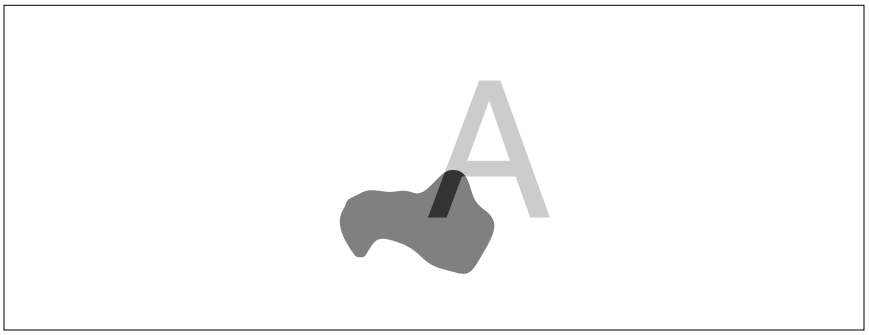
Drawing function	*Number*	*Logical operation on overlapping areas*
clear	0	Set to 0
and	1	Source AND destination
andReverse	2	Source AND NOT destination
copy	3	Source
andInverted	4	NOT source AND destination
noop	5	Destination
xor	6	Source XOR destination
or	7	Source OR destination
nor	8	NOT source AND NOT destination
equiv	9	NOT source XOR destination
invert	10	NOT destination
orReverse	11	Source OR NOT destination
copyInverted	12	NOT source
orInverted	13	NOT source OR destination
nand	14	NOT source OR NOT destination
set	15	Set to 1

The OR drawing function is of particular interest. When applied to the overlapping rectangles shown in Figure 7.5, this drawing function performs a logical OR on the bit pattern for 2 (0000 0010) and the bit pattern for 4 (0000 0100) to obtain a value of 6 (0000 0110). Using this drawing function, the overlapping area would be painted with the value stored at index 6. By storing meaningful values at 2, 4, and 6, the application can provide information to the user about the overlapping areas. For example, the application could store yellow at 2, blue at 4, and green at 6. All yellow/blue overlapping areas would then be colored green. This technique is useful for such applications as CAD/CAM programs that display overlapping deposition masks for integrated circuit design.

By extending the window system's color model to the Display PostScript system with the device pixel color space, overlapping areas of arbitrary shape (not just rectangles) can be painted with the drawing function. Consider, for

example, two screen areas painted by the Display PostScript system: one area is bounded by a Bézier curve and the other by the outline of a font character, as shown in Figure 7.6.

Figure 7.6 *Two irregular overlapping areas*



To paint the overlapping area, an application sets up the device pixel color space and the desired pixel index, and then draws the area bounded by the Bézier curve. It then sets the second color index, sets the drawing function to "`or`," and draws the letter A. The following outline summarizes the process:

```
[/DevicePixel 8] setcolorspace
2 setcolor
3 "setdrawingfunction"
```
Draw the Bézier area.
```
4 setcolor
7 "setdrawingfunction"
```
Draw the letter A.

Where `setdrawingfunction` is a window-specific PostScript custom operator to set the mode for the drawing function. See Table 7.2 for a list of drawing functions.

As a result, the intersection of the two areas is displayed automatically with the RGB value located at index 6—the logical OR of 2 and 4.

Note that the application is responsible for storing meaningful colors at proper index locations and choosing a drawing function that correctly relates them. To extend the previous example, consider an application that allows users to draw in three colors (yellow, red, and blue) on a 4-bit color machine. The application uses the OR drawing function to render pairs of overlapping colors in green, violet, and orange. When three or more different colors overlap, the overlap area is displayed in gray. The index assignments are shown in Table 7.3.

Table 7.3 *Example colors and index assignments for* OR *drawing function*

Color of object 1		*Color of object 2*		*Overlap color*	
Color	*Index*	*Color*	*Index*	*Color*	*Index*
yellow	2_{10} (0010_2)	blue	4_{10} (0100_2)	green	6_{10} (0110_2)
yellow	2_{10} (0010_2)	red	8_{10} (1000_2)	orange	10_{10} (1010_2)
blue	4_{10} (0100_2)	red	8_{10} (1000_2)	violet	12_{10} (1100_2)
yellow	2_{10} (0010_2)	violet	12_{10} (1100_2)	gray	14_{10} (1110_2)
blue	4_{10} (0100_2)	orange	10_{10} (1010_2)	gray	14_{10} (1110_2)
red	8_{10} (1000_2)	green	6_{10} (0110_2)	gray	14_{10} (1110_2)

7.2.6 Device Pixel Color Space and Layered Windows

If your application makes use of layered windows, you can use the device pixel color space to draw transparent graphics.

Many high-end graphics machines support layered windows as a performance enhancement. Layered windows are implemented with one or more bit planes separate from the main frame buffer. In a typical implementation, an 8-bit frame buffer overlays a 24-bit frame buffer.

One pixel value in the overlay frame buffer is called the transparent pixel. If a pixel in the overlay frame buffer is the transparent pixel, whatever was drawn in the underlying frame buffer shows through; if a pixel is not the transparent pixel, a normal color map lookup is performed to determine the color.

A typical use for layered windows is to overlay an annotation window on a complex image. In a flight simulator, for example, an aircraft instrument panel overlays an aerial scene. In this case, the area of the cockpit windshield is filled with the transparent pixel to allow a view of the underlying landscape. This scheme avoids potentially expensive damage to the complex three-dimensional image in the frame buffer.

An application can draw complex transparent areas by using the device pixel color space and drawing with the transparent pixel. For example, if the transparent pixel is 255, the following example draws a large, transparent A on a black field:

```
0 setgray
0 0 500 500 rectfill
[/DevicePixel 8] setcolorspace
255 setcolor
0 0 moveto
/TimesRoman 400 selectfont
(A) show
```

CHAPTER 8

Additions and Modifications to the Operators

This chapter supplements Chapter 8 in the *PostScript Language Reference Manual, Second Edition.* It details additions and modifications to the operators in the PostScript language. The majority of these operators reside in the dictionary **systemdict**. Operators that are defined elsewhere, in specific **ProcSet** resource dictionaries, are so noted.

Overview of This Chapter

addglyph *metric bitmap cid type32font* **addglyph –** 194

bind *proc* **bind** *proc* 196

cliprestore **– cliprestore –** 196

clipsave **– clipsave –** 196

composefont *key cmap array* **composefont** *font* 197

copypage **– copypage –** 197

cshow *proc string* **cshow –** 198

currentcolor **– currentcolor** $comp_1$ $comp_2$ … $comp_n$ 199

currentsmoothness **– currentsmoothness** *num* 199

currenttrapparams **– currenttrapparams** *parameterdict* 199

filter *src|tgt* $param_1$ …$param_n$ *name* **filter** *file* 199

findcolorrendering *renderingintent* **findcolorrendering** *crdname bool* 200

GetHalftoneName **– GetHalftoneName** *halftone* 200

GetPageDeviceName **– GetPageDeviceName** *devicesetup* 201

GetSubstituteCRD *renderingintent* **GetSubstituteCRD** *crdname* 201

glyphshow *name* **glyphshow –**
int **glyphshow –** 201

languagelevel **– languagelevel** *int* 202

removeall *type32font* **removeall –** 202

removeglyphs *firstcid lastcid type32font* **removeglyphs –** 202

setcolor *comp*$_1$ *comp*$_2$ *… comp*$_n$ **setcolor –** 203

setcolorscreen *redfreq redang redproc greenfreq greenang greenproc bluefreq blueang blueproc grayfreq grayang grayproc* **setcolorscreen –** 203

setcolorspace *array* **setcolorspace –**
name **setcolorspace –** 203

setcolortransfer *redproc greenproc blueproc grayproc* **setcolortransfer –** 203

sethalftone *halftone* **sethalftone –** 204

setscreen *frequency angle proc* **setscreen –**
frequency angle halftone **setscreen –** 204

setsmoothness *num* **setsmoothness –** 204

settrapparams *parameterdict* **settrapparams –** 205

settrapzone **– settrapzone –** 206

shfill *shadingdict* **shfill** 206

StartData *string int* **StartData –** 206

StartData *key int* **StartData –** 207

StartData *key int name* **StartData –** 207

widthshow *c*$_x$ *c*$_y$ *char string* **widthshow –** 208

---

addglyph *metric bitmap cid type32font* **addglyph –**

Note *The **addglyph** operator is defined in **BitmapFontInit ProcSet** rather than in the dictionary **systemdict**.*

addglyph inserts or replaces a Type 32 glyph; incrementally downloads Type 32 fonts. The **addglyph** operator installs an image string *bitmap* of a character identified by *cid* to the font cache for *type32font*. The **addglyph** operator will replace an existing character with a new image string.

The *metric* operand must be a 6- or 10-number array [*w0x w0y llx lly urx ury*] or [*w0x w0y llx lly urx ury w1x w1y vx vy*], respectively, where the elements of the array represent the following:

(*w0x, w0y*) The width (in real numbers) of the character in writing mode 0.

(*llx*, *lly*)	The lower-left corner of the character bounding box, in character space relative to the character origin; integers only. The difference (*urx* – *llx*) must equal the number of columns in the bitmap. The difference (*ury* – *lly*) must equal the number of rows in the bitmap.
(*urx*, *ury*)	The upper-right corner of the character bounding box; integers only. The difference (*urx* – *llx*) must equal the number of columns in the bitmap. The difference (*ury* – *lly*) must equal the number of rows in the bitmap.
(*w1x, w1y*)	Width of the characters in writing mode 1; real numbers.
(*vx*, *vy*)	Origin 1 relative to origin 0; real numbers. For further information, see Figure 5.6 on page 273 in the *PostScript Language Reference Manual, Second Edition*.

The *bitmap* operand consists of a string object containing the bitmap data. The bitmap representation is identical to the normal PostScript image representation for a 1-bit per pixel image. Logically, this image is painted in character space, with the (0, 0) corner of the image coinciding with (*llx*, *lly*) in character space.

The *cid* operand specifies an integer CID.

The *type32font* operand specifies the Type 32 font.

The coordinate system in which metrics and image data are interpreted is character space. When characters are shown at the CTM for which the font was designed, the complete transformation from character space to device space is the identity one-to-one. Thus, the image is treated as a device-resolution bitmap, positioned with the image space origin at (*llx*, *lly*) relative to the current point.

Errors: **stackunderflow**, **typecheck**, **rangecheck**, **limitcheck**, **invalidfont**

A **rangecheck** error occurs if *urx* is less than *llx*, *ury* is less than *lly*, or the image dimensions implied by these values are inconsistent with the length of the bitmap string.

A **limitcheck** error occurs if the glyph cannot be placed in the font cache, either because it is too large or because the cache is full.

An **invalidfont** error occurs if the specified font is not Type 32.

For further information, see Section 5.2.1, “Bitmap Fonts: Font Type 32,” on page 129.

bind *proc* **bind** *proc*

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

The **bind** operator has been changed as follows. If the value of the **IdiomRecognition** key is *true*, after completion of a **bind** operation on the argument procedure, the newly bound candidate procedure will be examined to determine if it matches any template procedure represented in the current resource category instances. All current instances in the **IdiomSet** resource category will be used. If the value of the **IdiomRecognition** key is *false*, no further action is taken and the **bind** operator behaves as described in the *PostScript Language Reference Manual, Second Edition*. (For further information on idiom recognition, see the section "IdiomSet" on page 37 and Table 10.1 on page 244.)

cliprestore **– cliprestore –**



resets the current clipping path from the one on the top of the clipping path stack and pops the clipping path stack, restoring the clipping path in effect at the time of the matching **clipsave** operation. This operator provides a way to restore only the clipping path without restoring all of the graphics state parameters associated with a **grestore** operation.

If there is no matching **clipsave** operation or if the most recent **clipsave** operation preceded the more recent unmatched **gsave** operation, the **cliprestore** operator does not pop the clipping path stack, although it does restore the clipping path from the clipping path stack associated with the **gsave** operation.

See Also: **clipsave, gsave, grestore**

clipsave **– clipsave –**



pushes a copy of the current clipping path on the clipping path stack. The saved state may be restored later by a matching **cliprestore** operation.

The clipping path is saved as part of the graphics state parameters when a **gsave** operation is performed; however the **clipsave** operator saves only the clipping path without the other graphics state parameters.

Errors: **limitcheck**

See Also: **cliprestore, grestore, gsave**

composefont *key cmap array* **composefont** *font*

This operator generates a composite font dictionary created from the *cmap* and the fonts or CIDFonts listed in the *array*. It then associates this dictionary with *key* in the **Font** resource category as if by **definefont**.

The *cmap* can be either a name (to be looked up as a key in the **CMap** resource category) or an actual CMap dictionary. The elements of *array* can be either names (to be looked up as keys in the **CIDFont** or **Font** resource categories) or actual CIDFont or font dictionaries.

composefont defines entries in the resulting composite font dictionary as follows:

CMap	*cmap*
Encoding	Identity mapping whose length is the same as **FDepVector**
FDepVector	Array of CIDFont or font dictionaries
FMapType	9
FontMatrix	Identity transformation
FontName	*key*
FontType	0
WMode	Value of **WMode** entry in *cmap*, if present, else 0

If the CMap specifies that the character space of any of the descendent fonts is to be transformed, an appropriate **makefont** is performed on each such font during the process of building the **FDepVector**. The transformation that is applied is the one specified by **beginusematrix** and **endusematrix** in the CMap. For further information, see the *Adobe CMap and CIDFont Files Specification*, Technical Note # 5014.

Note ***composefont*** *always creates a new font dictionary, regardless of whether there already exists one made from the same CMap and array of fonts. A PostScript language program should execute* ***composefont*** *prior to first use of a CID-keyed font, then invoke* ***findfont*** *each time it needs to access the resulting font.*

copypage **– copypage –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition. The following description applies to LanguageLevel 3.*



has the same behavior as the **showpage** operator, except that it does not perform an implicit **initgraphics** operation. In particular, the integer code argument to the **EndPage** page device procedure indicating the

circumstances under which it is being called will be 0 (indicating a **showpage** operation). See page 252 in the *PostScript Language Reference Manual, Second Edition*.

The consequences of this change are as follows:

- "**copypage erasepage**" will continue to work as before. Various applications have been observed to do this; their behavior will not change at all.
- **"n {copypage} repeat erasepage**," as a method to produce *n* copies of the page, will instead produce one copy followed by *n* – 1 blank pages.
- **copypage** should no longer be used as a means to implement forms. The first page will be printed correctly, showing both fixed and variable contents. Subsequent pages will show only the variable contents, the fixed contents having been erased.

The use of the **copypage** operator is discouraged.

Errors: **limitcheck**, **undefined**

See Also: **showpage, erasepage**

cshow *proc string* **cshow –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

To maintain compatibility with existing PostScript files, the **cshow** operator has been modified for composite fonts that contain CID fonts as base fonts. When the base font is a CID font, the code put on the stack for execution of *proc* is the low-order eight bits of the character code from *string*. The original code is stored in an internal variable. If the procedure does not change the current font but executes a **show** operator, the glyph is selected by using the original character code and the root composite font as the current font. For example, if the input string to the **cshow** operator is <2240>, the code on the stack when *proc* is executed would be 40 (hexadecimal). If the procedure put this value into a 1-byte string and did a **show** operation, the string <2240> would be used to look up and show the glyph from the root composite font. A **rangecheck** error would occur if *proc* tried to show a string with a byte other than 40 (hexadecimal). A **rangecheck** error is raised if a **show** operator executed by the procedure uses a value other than the code on the stack when the procedure is invoked.

Errors: **invalidaccess**, **invalidfont**, **nocurrentpoint**, **rangecheck**, **stackunderflow**, **typecheck**

currentcolor **– currentcolor** *comp*$_1$ *comp*$_2$ *… comp*$_n$

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

The **currentcolor** operator has been modified so that an arbitrary number of color components are now supported.

Errors: **stackoverflow**

See Also: **setcolor, setcolorspace**

currentsmoothness **– currentsmoothness** *num*



returns the current value, *num*, of the smoothness parameter in the graphics state, where *num* is a number from 0 to 1.

Errors: **stackoverflow**

See Also: **setsmoothness**

currenttrapparams **– currenttrapparams** *parameterdict*

Note *The **currenttrapparams** operator is defined in the **Trapping ProcSet** rather than in the dictionary **systemdict**.*



returns a copy of the current trapping parameter set. Subdictionaries and arrays are copied; strings are not copied. The returned dictionary can be modified and used as an argument to the **settrapparams** operator. It can also be stored in VM for use later in the same job, or it can be converted to a text file and saved to disk for use by future jobs. This operator can be called multiple times.

Errors: **stackoverflow**

See Also: **settrapparams, settrapzone**

filter *src|tgt param*$_1$ *…param*$_n$ *name* **filter** *file*
src|tgt dict name **filter** *file*

The **filter** operator has been modified with the introduction of the **ReusableStreamDecode** filter. See Section 3.3.7, “ReusableStreamDecode Filter,” on page 53.

findcolorrendering *renderingintent* **findcolorrendering** *crdname bool*

Note *This operator was introduced in version 2015 and is available on all Adobe PostScript 3 products.*



renderingintent is a name or string specifying the rendering intent. *crdname* is a name representing a CRD present in the **ColorRendering** resource category. If *bool* is *true*, *crdname* specifies a CRD present in the **ColorRendering** resource category that matches the desired rendering intent, device setup, and halftone combination. If *bool* is *false*, a CRD satisfying this combination exactly is not available. In this case, *crdname* specifies a substitution for the desired CRD. In either case, the CRD specified by *crdname* can be instantiated in the graphics state by using the **findresource** and **setcolorrendering** operators.

The **findcolorrendering** operator forms the name of a color rendering dictionary from the rendering intent, the device setup, and the halftone. The resulting name takes the form

renderingintent.devicesetup.halftone

where *renderingintent* is taken verbatim from the *renderingintent* operand, and *devicesetup* and *halftone* are found indirectly through procedures resident in the **ColorRendering** instance of the **ProcSet** resource category. *devicesetup* is returned by a call to the **GetPageDeviceName** procedure in the **ColorRendering ProcSet**. *halftone* is returned by a call to the **GetHalftoneName** procedure in the **ColorRendering ProcSet**.

The operator **findcolorrendering** should be called after all commands that influence either the halftone or the device setup in order to insure that all parameters that may be considered for selection of a CRD are accounted for correctly.

For further information about **findcolorrendering**, see Section 6.2.1, “CRD Selection Based on Rendering Intent,” on page 159.

GetHalftoneName **– GetHalftoneName** *halftone*

Note *The **GetHalftoneName** operator is defined in the **ColorRendering ProcSet** rather than in the dictionary **systemdict**.*

uses the optional **HalftoneName** key in the current halftone dictionary. **GetHalftoneName** always returns a name, and it may perform a variety of operations in an effort to return a meaningful name. If it is unable to return a meaningful name, it returns /none. The name selection processes for **GetHalftoneName** may be as simple as looking for the appropriate name in

the appropriate location and returning /none if it is not found, or it may be considerably more complex. For example, it could classify the current halftone in terms of angle and frequency.

GetPageDeviceName **– GetPageDeviceName** *devicesetup*

Note *The* ***GetPageDeviceName*** *operator is defined in the* ***ColorRendering ProcSet*** *rather than in the dictionary* ***systemdict****.*

always returns a name, and it may perform a variety of operations in an effort to return a meaningful name. If it is unable to return a meaningful name, it returns /none. **GetPageDeviceName** uses, as an operand to its name selection process, the **pagedevice** key **PageDeviceName**. The name selection processes for **GetPageDeviceName** may be as simple as looking for the appropriate name in the appropriate location and returning /none if it is not found, or it may be considerably more complex.

GetSubstituteCRD *renderingintent* **GetSubstituteCRD** *crdname*

Note *The* ***GetSubstituteCRD*** *operator is defined in the* ***ColorRendering ProcSet*** *rather than in the dictionary* ***systemdict****.*

renderingintent is the rendering intent passed to the **findcolorrendering** operator, and *crdname* is the name of a substitution CRD that exists in the **ColorRendering** resource category. When **GetSubstituteCRD** is called, **findcolorrendering** always returns *false*, because the desired CRD is not available. **findcolorrendering** returns the CRD returned by **GetSubstituteCRD**. If **findcolorrendering** does not call **GetSubstituteCRD**, it returns *true*. **GetSubstituteCRD** returns **DefaultColorRendering** in the event it cannot generate a meaningful CRD substitution. All PostScript interpreters beyond LanguageLevel 1 have a CRD named **DefaultColorRendering**.

glyphshow *name* **glyphshow –**
int **glyphshow –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

If the current font is a CID font, then the argument must be an integer object. The integer is used as the CID to find and show the character in the CID font. A **typecheck** error is raised if the element on the stack is an integer and the current dictionary is not a CID font or the current dictionary is a CID font and

the object on the stack is not an integer. An **invalidfont** error is raised if **glyphshow** is executed when the current dictionary is a composite font (Type 0).

Errors: **invalidaccess**, **invalidfont**, **nocurrentpoint**, **stackunderflow**, **typecheck**

languagelevel **– languagelevel** *int*

returns the level of the PostScript language. **languagelevel** returns 3 in an interpreter that supports all LanguageLevel 3 features.

removeall *type32font* **removeall –**

Note *The **removeall** operator is defined in **BitmapFontInit ProcSet** rather than in the dictionary **systemdict**.*

removes glyphs defined for the font *type32font*.

The deleted glyphs are removed from the font cache immediately. They may continue to occupy memory until all pages on which those glyphs were used have been printed.

Errors: **invalidfont**, **rangecheck, typecheck**, **stackunderflow**

An **invalidfont** error occurs if the specified font is not Type 32.

For further information, see Section 5.2.1, “Bitmap Fonts: Font Type 32,” on page 129.

removeglyphs *firstcid lastcid type32font* **removeglyphs –**

Note *The **removeglyphs** operator is defined in **BitmapFontInit ProcSet** rather than in the dictionary **systemdict**.*

removes all glyphs identified by **CID** from *firstcid* to *lastcid*, inclusive, in the font *type32font*.

The deleted glyphs are removed from the font cache immediately. They may continue to occupy memory until all pages on which those glyphs were used have been printed.

Errors: **invalidfont**, **rangecheck, typecheck**, **stackunderflow**

An **invalidfont** error occurs if the specified font is not Type 32.

A **rangecheck** error occurs if *lastcid* is less than *firstcid* or if these numbers are outside the valid range of CIDs (0 to 65535). However, no error arises from references to nonexistent glyphs.

For further information, see Section 5.2.1, “Bitmap Fonts: Font Type 32,” on page 129.

setcolor *comp*$_1$ *comp*$_2$ … *comp*$_n$ **setcolor –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

The **setcolor** operator has been modified so that an arbitrary number of color components are now supported. When **DeviceN** is the current color space, the **setcolor** operators takes a number of arguments equal to the length of the *names* array.

setcolorscreen *redfreq redang redproc greenfreq greenang greenproc bluefreq blueang blueproc grayfreq grayang grayproc* **setcolorscreen –**

The behavior of **setcolorscreen** can be altered by the **HalftoneMode** and **AccurateScreens** user parameters. See **HalftoneMode** and **AccurateScreens** in Table 10.1, “User parameters,” on page 244.

setcolorspace *array* **setcolorspace –**
name **setcolorspace –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*



The color space names have been extended to include **DeviceN**.

setcolortransfer *redproc greenproc blueproc grayproc* **setcolortransfer –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

The transfer functions set by **setcolortransfer** have no effect on any component in a device whose **ProcessColorModel** is **DeviceN**. In that case, all the components’ transfer functions must be specified as **TransferFunction** entries in the halftone dictionary supplied to **sethalftone**.

sethalftone *halftone* **sethalftone –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

The behavior of **sethalftone** can be altered by the **HalftoneMode** user parameter. See **HalftoneMode** in Table 10.1, “User parameters,” on page 244.

setscreen *frequency angle proc* **setscreen –**
frequency angle halftone **setscreen –**

The behavior of **setscreen** can be altered by the **HalftoneMode** and **AccurateScreens** user parameters. See **HalftoneMode** and **AccurateScreens** in Table 10.1, “User parameters,” on page 244.

setsmoothness *num* **setsmoothness –**



sets the smoothness parameter in the graphics state to *num*, where *num* is a number from 0 to 1. This operator controls the quality of smoothly shaded output, and thus indirectly controls the rendering performance. Smoothness is the allowable color error between shading approximated with piecewise linear interpolation and the true shading of a possibly nonlinear shading function. The error is measured for each color component, and the maximum error is used. The allowable error (or tolerance) is specified as a percentage of the range of the color component. This percentage is expressed as a value from 0 to 1. Thus, a smoothness parameter of 0.1 represents a tolerance of 10 percent in each color component.

Each device may have internal limits on the maximum and minimum tolerances attainable. For example, setting smoothness to 1 may result in an internal smoothness of 0.5 on a high-quality color device, and setting smoothness to 0 on the same device may result in an internal smoothness of 0.01 if an error of that magnitude is imperceptible on that device.

The smoothness parameter may also interact with the accuracy of color conversion. In the case of a color conversion defined by a PostScript procedure or table, the conversion function is unknown. Thus, the error may be sampled at too low a frequency, in which case, the accuracy defined by the smoothness parameter cannot be guaranteed. In most cases, however, where the conversion function is smooth and continuous, the accuracy should be within the specified tolerance.

The smoothness parameter is subject to **gsave** and **grestore** operations.

The **setsmoothness** operator is similar to the **setflat** operator. Note, however, that flatness is measured in device-dependent units of pixel width, whereas smoothness is measured as a percentage of color component range.

Errors: **stackoverflow, rangecheck**

See Also: **currentsmoothness**

settrapparams *parameterdict* **settrapparams –**

Note *The **settrapparams** operator is defined in the **Trapping ProcSet** rather than in the dictionary **systemdict**.*



sets or updates the values of the current trapping parameter set. *parameterdict* is a dictionary with the same structure and entries as the trapping parameter set and is usually a small subset of the trapping parameter set. The operator **settrapparams** combines the key-value pairs supplied to it with those in the existing trapping parameter set, replacing or adding entries as appropriate. Unrecognized entries in *parameterdict* are ignored. Changes to the current trapping parameter set do not affect already defined trapping zones. This operator can be called multiple times.

The effects of calls to **settrapparams** are cumulative. A particular entry persists through subsequent calls to **settrapparams** until overridden explicitly or until the state of the trapping parameter set is restored to some previous state by a **restore** operation. Therefore, a PostScript program can make independent calls to **settrapparams**, each requesting particular features or processing options, leaving the settings for the other features unaffected. This allows different options to be specified at different times. For example, a job containing an image may set **ImageInternalTrapping** to *true* for a zone covering the image, without changing the trap width.

This cumulative behavior does not apply to the contents of the **ColorantZoneDetails** dictionary. If a job wishes to change just one entry in the **ColorantZoneDetails** dictionary, the job must retrieve the current contents and merge them with the new value before calling the **settrapparams** operator. This enables the job to remove entries from the **ColorantZoneDetails** dictionary. Note that this is different from the cumulative behavior for **ColorantDetails** in the **setpagedevice** operator.

Errors: **stackunderflow, typecheck, limitcheck, rangecheck**

See Also: **currenttrapparams, settrapzone**

settrapzone **– settrapzone –**

Note *The **settrapzone** operator is defined in the **Trapping ProcSet** rather than in the dictionary **systemdict**.*



sets a trapping zone. The current path defines the area of the zone, and the current value of the trapping parameter set defines the trapping parameters for this zone. Subsequent changes to the current path or the current trapping parameter set do not affect this trapping zone. As with the **fill** and **stroke** operators, **settrapzone** implicitly performs a **newpath** operation after it has defined a trapping zone. The inside of the path is determined by the normal non-zero winding number rule. This operator can be called multiple times.

Errors: **limitcheck**

See Also: **currenttrapparams, settrapparams**

shfill *shadingdict* **shfill**



The **shfill** operator paints the shape and color transitions described by the **Shading** dictionary, subject to the current clipping path. The current path, if any, is unaffected. No other attributes of the graphics state are changed.

shadingdict is a **Shading** dictionary. All geometric coordinates in the dictionary are interpreted relative to current user space. All color values are interpreted relative to the **ColorSpace** attribute of the **Shading** dictionary. The **Background** attribute, if present, is ignored.

shfill should only be applied to bounded and/or geometrically defined shadings. If **shfill** is applied to an unbounded shading, the corresponding field of color will be painted across the entire current clipping region, which may be a slow process.

Note *Some shadings may read large blocks of data from PostScript streams. If the **currentfile** operator is used as the source of such data, the data should immediately follow the invocation of the **shfill** operator, just as image data follows the invocation of the **image** operator.*

Errors: **rangecheck**, **undefinedresult**

For further information, see Section 4.4, “Smooth Shading,” on page 76.

StartData *string int* **StartData –**
(*defined in the **CIDInit ProcSet***)

StartData *key int* **StartData –**
StartData *key int name* **StartData –**
(*defined in the* ***FontSetInit ProcSet*** *described in the section “FontSet Resource” on page 133*)

introduces the binary data section of a **CIDFont** or **FontSet** file and completes the construction of a **CIDFont** or **FontSet** resource instance. There are two different operators named **StartData**, one in the **CIDInit ProcSet** and one in the **FontSetInit ProcSet**; they are for defining **CIDFonts** and **FontSets**, respectively.

For either operator, the *int* operand specifies the length of the binary data section that follows. The data begins immediately after the white space character that terminates the invocation of **StartData**.

If **StartData** is invoked directly as part of a PostScript program, it consumes this data and incorporates it into the resource instance being constructed. However, if **StartData** is invoked from within a resource file being loaded by the **findresource** operator, it does not load the data into VM. Instead, it arranges for the data to be accessed from the filesystem dynamically as needed during character rendering.

For a **CIDFont**, **StartData** expects to be invoked when the dictionary stack contains the **CIDInit ProcSet** dictionary and the **CIDFont** dictionary being built. The string operand specifies the data format of the following data; it can be either Binary or Hex (Binary is strongly recommended). **StartData** then invokes the equivalent of:

```
CIDFontName /CIDFont defineresource pop
```

and removes two dictionaries from the dictionary stack.

For a **FontSet**, **StartData** expects to be invoked when the dictionary stack contains the **FontSetInit ProcSet** dictionary. It creates a **FontSet** instance dictionary from scratch, using information from the binary data (which is expected to conform to the compact font format specification). **StartData** then invokes the equivalent of:

```
key /FontSet defineresource pop
```

and removes one dictionary from the dictionary stack.

For further information, see Section 5.3.1, “CID Font Resources,” on page 136 and Section 5.2.3, “CFF and Chameleon Fonts: FontType 2 and FontType 14,” on page 132.

widthshow c_x c_y *char string* **widthshow –**

Note *The description of this operator has been modified from that given in the PostScript Language Reference Manual, Second Edition.*

paints the characters of *string* in a manner similar to the **show** operator. But while doing so, the **widthshow** operator adjusts the width of each occurrence of the character *char* by adding c_x to its *x* width and c_y to its *y* height, thus modifying the spacing between it and the next character. *char* is an integer used as a character code. When the value returned by **currentfont** is a composite font with an **FMapType** value of 9, the integer that *char* is compared with depends on the mapping defined by the CMap. If the mapping from a character code is to a CID or a glyph name, then the integer is the value of the character code. If the mapping is to another character code, then the integer value is $(f \times 256) + c$, where *f* is the font number and *c* is the character code returned by the mapping. For example, if the CMap defines the following mappings:

```
0 usefont    %Following mapping use font number 0
1 begincidchar
<8140> 633   %Maps two-byte code 16#8140 to CID 633
endcidchar

1 usefont    %Following mapping use font number 1
2 beginbfchar
<61> /a      %Maps one-byte code 16#61 to glyh name /a
<40> <A9>    %Maps one-byte code 16#40 to character code 16#A9
endbfchar
```

and if the string contained `<8140 61 40>`, then the integers compared with *char* would be `33308 (16#8140)`, `97 (16#61),` and `425 ((1 × 256) + 16#A9)` in sequence. For information on how the **widthshow** operator works with base fonts and composite fonts with other **FMapType** values, see the *PostScript Language Reference Manual, Second Edition*, on page 549.

CHAPTER 9

LanguageLevel 1/ LanguageLevel 2 Compatibility

The PostScript language is designed to be a universal standard for device-independent page descriptions, but each PostScript implementation supports features and capabilities particular to that implementation. Appendix D in the *PostScript Language Reference Manual, Second Edition,* presents guidelines for taking advantage of language extensions while maintaining compatibility with all PostScript interpreters.

Overview of This Chapter

9.1 Compatibility Operators and Keys Listed by Dictionary and by Function 211

9.1.1 By Dictionary 211

9.1.2 By Function 212

SCC Compatibility Operators 212

Page Duplexing Compatibility Operators and Keys 213

Device Compatibility Operators and Keys 213

Imagesetter Compatibility Operators and Keys 213

9.2 Operator and Key Details 214

9.3 Compatibility Operations 233

9.3.1 Error Behavior 234

9.3.2 Changing Persistent Values with Password 234

9.3.3 SCC Operations 235

Stop bits (Table 9.1) 235

Data bits (Table 9.2) 236

Flow control (Table 9.3) 236

Parity (Table 9.4) 236

Options byte to device parameters conversion (Table 9.5) 237

Positions 1 and 0 (Table 9.6) 237

9.3.4 Paper Size Operations 238

Paper size compatibility operators (in userdict) (Table 9.7) 238

9.3.5 Paper Tray Operations 239

Paper tray compatibility operators (Table 9.8) 239

LanguageLevel 1 implementations provide a collection of device control and system parameter configuration operators and procedures, most of which are defined in the dictionary **statusdict**. The contents of **statusdict** are product dependent, although an attempt has been made to maintain a consistent specification for common features. **statusdict** is the dictionary for product-specific operators and definitions.

In LanguageLevel 2 implementations, device control and system parameter configuration is accomplished through the device setup and interpreter parameters. However, to maintain compatibility with LanguageLevel 1 driver software that might depend on **statusdict** operators and keys (objects) that were often present in LanguageLevel 1 products, a collection of **statusdict** operators and keys was included in each LanguageLevel 2 implementation. Almost all of these functions are implemented as PostScript language procedures that call appropriate LanguageLevel 2 operators such as **setpagedevice**.

Adobe recommends that you do not use the **statusdict** operators and keys in PostScript language drivers beyond LanguageLevel 1 because the presence or absence of the operators and keys is product dependent. Instead, the appropriate standard PostScript operators should be used.

This chapter gives information on the compatibility operators and keys that have been added to the PostScript language since the publication of the *PostScript Language Reference Manual, Second Edition.*

- Section 9.1, “Compatibility Operators and Keys Listed by Dictionary and by Function,” summarizes the compatibility operators and keys that have been added to the PostScript language since the publication of the *PostScript Language Reference Manual, Second Edition.*

- Section 9.2, “Operator and Key Details,” on page 214 gives a detailed description of each operator or key; the operators and keys are listed alphabetically.

- Section 9.3, “Compatibility Operations,” on page 233 describes the LanguageLevel 1 compatibility objects present in LanguageLevel 2. The majority of these LanguageLevel 1 objects are operators in **statusdict**. There is a LanguageLevel 2 method of performing most LanguageLevel 1 compatibility operations. Error behavior, the use of passwords, SCC operations, paper size operations, and paper tray operations are described.

9.1 Compatibility Operators and Keys Listed by Dictionary and by Function

This section gives information on the compatibility operators and keys that have been added to the PostScript language since the publication of the *PostScript Language Reference Manual, Second Edition.* It lists the operators and keys first by dictionary and then by function. A detailed description of each operator or key is given in Section 9.2, “Operator and Key Details,” on page 214. The following symbols are used throughout the listings:

† Typically present in all releases up to and including the current PostScript implementation.

‡ Typically present in all releases up to and including the current PostScript implementation. However, in the absence of the associated feature, it performs no function aside from its documented effect on the operand stack.

ø Typically present in all releases up to and including the current PostScript imagesetter implementations.

§ Requires execution in a system administrator job.

¶ Can affect page device parameters.

Operators and keys without a symbol are associated with a particular feature and are defined only if the feature is present in the product.

9.1.1 By Dictionary

In **statusdict**:

a3tray¶
a4tray¶
accuratescreensø
appletalktype
b5tray¶
buildtime†
byteorder†
checkpassword†
checkscreenø
defaulttimeouts‡
diskonline
diskstatus
lettertray†¶
manualfeed
manualfeedtimeout
margins‡
mirrorprint
newsheet
pagecount‡
pagemarginø
pageparamsø
pagestackorder
printername†
processcolors
setduplexmode¶
sethardwareiomode‡§
setjobtimeout‡
setmargins‡§¶
setmirrorprint¶
setpageø¶
setpagemarginø¶
setpageparamsø¶
setpagestackorder §¶
setprintername‡§
setresolution¶
setsccbatch§

doprinterrors
dostartpage
dosysstart
duplexmode
emulate
firstside
hardwareiomode‡
initializedisk§
jobname†
jobtimeout†
ledgertray¶
legaltray¶
product†
ramsize
realformat†
resolution
revision†
sccbatch
sccinteractive‡
setaccuratescreensø
setdefaulttimeouts†§¶
setdoprinterrors§
setdostartpage §
setdosysstart§
setsccinteractive‡§
setsoftwareiomode‡§
settumble¶
setuserdiskpercent§
softwareiomode‡
tumble
userdiskpercent
waittimeout‡
11x17tray¶

In **userdict**:

a3¶
a4¶
a4small¶
b5¶
ledger¶
legal‡¶
letter‡¶
lettersmall¶
note¶
11x17¶

In **systemdict**:

devdismount†§
devforall†
devformat†§
devmount†§
devstatus†

9.1.2 By Function

This section lists the compatibility operators and keys by function. A detailed description of each operator or key can be found in Section 9.2, “Operator and Key Details,” on page 214.

Operators and keys are grouped in the following categories:

- SCC compatibility operators
- Page duplexing compatibility operators and keys
- Device compatibility operators and keys
- Imagesetter compatibility operators and keys

SCC Compatibility Operators

sccbatch *channel* **sccbatch** *baud options*

sccinteractive‡ *channel* **sccinteractive** *baud options*

setsccbatch§ *channel baud options* **setsccbatch** –

setsccinteractive‡§ *channel baud options* **setsccinteractive** –

Page Duplexing Compatibility Operators and Keys

duplexmode – **duplexmode** *boolean*

firstside – **firstside** *boolean*

newsheet – **newsheet** –

setduplexmode¶ *boolean* **setduplexmode** –

settumble¶ *boolean* **settumble** –

tumble – **tumble** *boolean*

Device Compatibility Operators and Keys

devdismount†§ *string* **devdismount** –

devforall† *proc scratch* **devforall** –

devformat†§ *string pages format* **devformat** –

devmount†§ *string* **devmount** *boolean*

devstatus† *string* **devstatus** *false* (if device not found)

string **devstatus** *searchable writeable hasNames mounted removable searchOrder freePages size true* (if device found)

Imagesetter Compatibility Operators and Keys

accuratescreensø – **accuratescreens** *boolean*

checkscreenø *freq angle* **checkscreen** *actualfreq actualangle moirelength*

pagemarginø – **pagemargin** *x*

pageparamsø – **pageparams** *width height margin orientation*

setaccuratescreensø *boolean* **setaccuratescreens** –

setpageø¶ *width height orientation* **setpage** –

setpagemarginø¶ *margin* **setpagemargin** –

setpageparamsø¶ *width height margin orientation* **setpageparams** –

9.2 Operator and Key Details

a3tray¶ **– a3tray –**

present only if the size [842 1191] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror**, **rangecheck**, **limitcheck**

a4tray¶ **– a4tray –**

present only if the size [595 842] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror**, **rangecheck**, **limitcheck**

accuratescreensø – **accuratescreens** *boolean*

returns the value of the user parameter **AccurateScreens**. A value of *true* means that accurate screening is enabled.

Errors: **stackoverflow**

appletalktype – **appletalktype** *string*

returns a string with the same value as the **LocalTalkType** device parameter in the %LocalTalk% parameter set and the **EtherTalkType** device parameter in the %EtherTalk% parameter set. Redefining the **appletalktype** operator will cause both the **LocalTalkType** entry and the **EtherTalkType** entry to change. Similarly, changes to the **EtherTalkType** or the **LocalTalkType** entry will change the string returned by the **appletalktype** key.

The compatibility key **appletalktype** is present only if either the %LocalTalk% or the %EtherTalk% device name is present.

Errors: **stackoverflow**

b5tray¶ **– b5tray –**

present only if the size [516 729] or [499 709] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror, rangecheck, limitcheck**

buildtime† – **buildtime** *int*

returns an integer with the same value as the system parameter **BuildTime**.

Errors: **stackoverflow**

byteorder† – **byteorder** *boolean*

returns a Boolean value with the same value as the system parameter **ByteOrder**.

Errors: **stackoverflow**

checkpassword† *int* **checkpassword** *boolean*
string **checkpassword** *boolean*

checks whether *string* or *int* (*int* is converted to a string) is a valid password for either the **SystemParamsPassword** or the **StartJobPassword** key. If valid, *true* is returned; otherwise, *false* is returned. If neither password is set, then *true* will be returned. A returned value of *true* indicates that *string* or *int* is a valid argument to the **startjob** and **exitserver** operators. There is no LanguageLevel 2 equivalent for the **checkpassword** operator.

Errors: **stackunderflow**, **typecheck**

checkscreenø *freq angle* **checkscreen** *actualfreq actualangle moirelength*

returns the actual screen frequency and angle that would be used if the **setscreen** operator were called. *moirelength* is the distance in inches where the deviation from the requested dot pattern would reach a fixed fraction of a cell size and is thus a measure of how accurate the actual screen would approximate the requested screen. Note that this operator does not affect the current screen.

Errors: **stackoverflow**

defaulttimeouts‡ – **defaulttimeouts** *job manualfeed wait*

returns the values of the system parameters **JobTimeout** and **WaitTimeout** and the page device parameter **ManualFeedTimeout** for *job*, *wait*, and *manualfeed*, respectively. The **defaulttimeouts** operator always returns three values, even if the corresponding system parameters are not present (zeros are returned in this case).

Errors: **stackoverflow**

devdismount†§ *string* **devdismount** –

sets the device parameter **Mounted** to *false* within the parameter set corresponding to the device specified by *string*. The device must be mounted before it can be dismounted. Trying to dismount a device that is not mounted will have no effect. Some devices cannot be dismounted; trying to dismount these will also have no effect.

In LanguageLevel 2, you can dismount from any save level if the **SystemParamsPassword** entry is not set. If it is set, a **devdismount** operation will raise an **invalidaccess** error unless executed within an unencapsulated system administrator job.

Errors: **invalidaccess**, **stackunderflow**, **undefinedfilename**

devforall† *proc scratch* **devforall** –

enumerates all known storage devices.

For each storage device, the **devforall** operator copies its name into the supplied *scratch* string, pushes a string object that is the substring of *scratch* that was actually used, and calls *proc*. **devforall** does not return any results of its own, but *proc* may do so.

Note *Some of the storage devices enumerated by **devforall** correspond to communication channels. These will have a **HasNames** value equal to false.*

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**, **undefined**

devformat†§ *string pages format* **devformat** –

sets the **LogicalSize** device parameter of the parameter set corresponding to the device specified by *string* to the value specified by *pages*. It then sets the **InitializeAction** entry, in the same parameter set, to the value of (*format* + 1). See the **InitializeAction** and **LogicalSize** file system device parameters for complete details.

Errors: **invalidaccess**, **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**, **undefined**, **undefinedfilename**

devmount†§ *string* **devmount** *boolean*

sets to *true* the value of **Mounted** device parameter of the parameter set corresponding to the device specified by *string*. It then returns the resulting value of **Mounted** by reading it from the same parameter set. The value *true* indicates that the device was successfully mounted or was already mounted. The value *false* indicates that the device cannot be mounted at this time.

In LanguageLevel 2, you can mount from any save level if the **SystemParamsPassword** entry is not set. If it is set, a **devmount** operation will raise an **invalidaccess** error unless executed within an unencapsulated system administrator job.

Errors: **invalidaccess**, **stackunderflow**, **undefinedfilename**

devstatus† *string* **devstatus** *false* (if device not found)

string **devstatus** *searchable writeable hasNames mounted removable searchOrder freePages size true* (if device found)

takes a disk device name (%disk%) identified by *string* from the stack. If the device name is unknown, only *false* will be left on the stack. If the device name is found, it pushes various file system attributes for the device. The attributes are *searchable*, *writeable*, *hasNames*, *mounted*, *removable*, *searchOrder*, *freePages*, and *size*.

searchable The *searchable* attribute corresponds to the **Searchable** device parameter and is a Boolean value that indicates that the device will be searched when looking for a file with no device name prefix in its name.

writeable The *writeable* attribute corresponds to the **Writeable** device parameter and indicates whether files on this device can be written.

hasNames The *hasNames* attribute corresponds to the **HasNames** device parameter and is a Boolean value that indicates whether the device supports named files.

mounted The *mounted* Boolean value corresponds to the **Mounted** device parameter and indicates whether the device is mounted.

removable The *removable* Boolean value corresponds to the **Removable** device parameter and indicates whether the media within the device can be removed.

searchOrder The *searchOrder* attribute corresponds to the **SearchOrder** device parameter and indicates the priority at which the device participates when searching for a file in operations in which no device has been specified.

freePages The *freePages* attribute corresponds to the **Free** device parameter and indicates the amount of free space (in pages).

size The *size* attribute corresponds to the **LogicalSize** device parameter and indicates the current size of the PostScript software file system (in pages).

Note *In LanguageLevel 1, a "page" had a file system specific size (typically 1024).*

Errors: **stackunderflow**

diskonline – **diskonline** *boolean*

returns *true* if and only if a writeable disk device is mounted. This is determined by searching all device parameter sets named %disk*%, where * represents zero or more additional digits or integers in the name. If the **Writeable** parameter has a value of *true* for any of the sets searched, *boolean* is set to *true*; otherwise, it is set to *false*. Note that a disk parameter set with a **Writeable** value of *true* need not have an initialized file system.

Errors: **stackoverflow**

diskstatus – **diskstatus** *free total*

returns the number of disk pages free (a page is typically 1024 characters) and the total number of pages available on all writeable disk devices. This is determined by searching all device parameter sets named %disk*% that

have a **Writeable** parameter set to *true*. The symbol ∗ represents zero or more additional digits or integers in the name. *free* is the sum of the **Free** parameters from all such parameter sets, and *total* is the sum of the **LogicalSize** parameters from all such parameter sets.

Errors: **stackoverflow**

doprinterrors – **doprinterrors** *boolean*

returns the value of the system parameter **DoPrintErrors**.

The system parameter **DoPrintErrors** must be present for the operator **doprinterrors** to be present.

Errors: **stackoverflow**

dostartpage – **dostartpage** *boolean*

returns the value of the system parameter **DoStartPage**.

The system parameter **DoStartPage** must be present for the compatibility operator **dostartpage** to be present.

Errors: **stackoverflow**

dosysstart – **dosysstart** *boolean*

returns *false* if and only if the value of the system parameter **StartupMode** is 0.

The system parameter **StartupMode** must be present for the compatibility operator **dosysstart** to be present.

Errors: **stackoverflow**

duplexmode – **duplexmode** *boolean*

returns the value of the page device parameter **Duplex**.

The page device parameter **Duplex** must be present for the compatibility operator **duplexmode** to be present.

Errors: **stackoverflow**

emulate *input-stream emulation-name* **emulate** –
or
input-stream params-dict emulation-name **emulate** –

causes the PostScript interpreter to yield control, and the emulator named by *emulation-name* to start processing. The **emulate** operator is present in **statusdict** and only in products that have one or more emulators coresident with the PostScript interpreter. The exact semantics of the emulators are product dependent and may be different in different products, even though the emulation name may be the same. (Emulators, if any, are product specific.) In most coresident emulations, the command sequence **ESC-DEL-0** can be used to make the emulator return control to the PostScript interpreter; however, the PostScript language context will generally have been lost.

The allowed values of *emulation-name* may be found in the implicit resource category **Emulator**. An illegal *emulation-name* will cause a **rangecheck** error.

A *params-dict* argument is optional. If the named emulator does not need parameters and a *params-dict* is provided, the dictionary will be ignored. If the named emulator requires parameters and no *params-dict* is provided, then product-dependent defaults will be used if possible. Currently, no emulators require parameters.

The *input-stream* is a file object that becomes the input source for the emulator. The *input-stream* specified must be appropriate to the product-dependent emulator. An illegal *input-stream* will cause an **invalidaccess** error.

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**

firstside – **firstside** *boolean*

returns *true* if the current page is a front side and *false* if the current page is a back side.

Note *This compatibility operator is sometimes found on products that do not support duplex printing. On these products,* ***firstside*** *may be used to generate output that is intended to be copied using a duplex copier.*

Errors: **stackoverflow**

hardwareiomode‡ – **hardwareiomode** *int*

returns an *int* that indicates the current communication channel for which the value of the **Enabled** parameter in the corresponding device parameter set is *true*. It will always return the channel indicated by the **CurInputDevice** entry if that channel is on and enabled and if it is one of the ones listed below. Otherwise, the smallest such *int* is returned. If none in the list is on and enabled, 0 is returned. The interpretation of *int* is:

0	%Serial%
1	%Parallel%
2	%LocalTalk%
3	%SerialB%

The %Serial%, %Parallel%, %SerialB%, or %LocalTalk% parameter set must be present for the compatibility operator **hardwareiomode** to be present.

Errors: **stackoverflow**

initializedisk§ *pages action* **initializedisk** –

initializes each writeable disk, setting the disk device parameters **LogicalSize** and **InitializeAction** to the value of *pages* and *action* + 1, respectively.

Errors: **invalidaccess**, **ioerror**, **rangecheck**, **stackunderflow**, **typecheck**

jobname† – **jobname** *string*

returns a string with the same value as the user parameter **JobName**. Redefining either the **jobname** key or the user parameter **JobName** redefines the other to the same value.

The user parameter **JobName** must be present for the compatibility key **jobname** to be present.

Errors: **stackoverflow**

jobtimeout† – **jobtimeout** *int*

returns the value of the user parameter **JobTimeout**.

Errors: **stackoverflow**

ledgertray¶ **– ledgertray –**

present only if the size [1224 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror**, **rangecheck**, **limitcheck**

legaltray¶ **– legaltray –**

present only if the size [612 1008] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror**, **rangecheck**, **limitcheck**

lettertray†¶ **– lettertray –**

present only if the size [612 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror**, **rangecheck**, **limitcheck**

manualfeed – **manualfeed** *boolean*

returns a Boolean value that works in conjunction with the page device parameter **ManualFeed** to determine whether a page is fed manually. If the value of either **manualfeed** or **ManualFeed** is *true* at the time of a **showpage** or **copypage** operation, then that page will be fed manually; otherwise, the page will not be fed manually.

The values of **ManualFeed** and **manualfeed** are determined independently. That is, setting the **manualfeed** Boolean value or setting the page device parameter **ManualFeed** does not affect the value of the other.

The **manualfeed** key is present in **statusdict** if and only if the page device parameter **ManualFeed** is defined for the product. The initial value of **manualfeed** at power-on is *false*.

Errors: **stackoverflow**

manualfeedtimeout – **manualfeedtimeout** *int*

returns an integer that works in conjunction with the page device parameter **ManualFeedTimeout** to determine the **manualfeed** time-out for any given page. By default, **manualfeedtimeout** is not defined in **statusdict** and the value of the page device parameter **ManualFeedTimeout** is used to determine the time-out value. If a job has defined **manualfeedtimeout** to be an integer value in **statusdict,** then this value will be used instead of **ManualFeedTimeout** for the time-out value.

The values of **ManualFeedTimeout** and **manualfeedtimeout** are determined independently. That is, setting the **manualfeedtimeout** integer or setting the page device parameter **ManualFeedTimeout** does not affect the value of the other.

Errors: **stackoverflow**

margins‡ – **margins** *top left*

returns the *x* and *y* components of the page device parameter **Margins** as *left* and *top*, respectively.

Errors: **stackoverflow**

mirrorprint – **mirrorprint** *boolean*

returns the value of the page device parameter **MirrorPrint**.

Errors: (none)

newsheet – **newsheet** –

If the page device **Duplex** is *true* and the current page is a back side, the **newsheet** operator causes this page to be printed as is (perhaps blank) and sets up a clean printing environment for the next page. Otherwise, executing the **newsheet** operator has no effect.

The page device parameter **Duplex** must be present for the compatibility operator **newsheet** to be present.

Errors: (none)

pagecount‡ – **pagecount** *int*

returns the value of the system parameter **PageCount**.

Errors: **stackoverflow**

pagemarginø – **pagemargin** *x*

returns the *x* value (measured in device units) of the page device parameter **PageOffset**.

Errors: **stackoverflow**

pageparamsø – **pageparams** *width height margin orientation*

Suppose that the value of the page device parameter **PageSize** is [*x y*], the value of **PageOffset** is [*X Y*], and the value of **Orientation** is *o*. Then the operator **pageparams** will return values of the form

x y <X or *Y*> 1

or

y x <X or *Y*> 0

Because the relationship between page device parameters and the **setpageparams** parameters *margin* and *orientation* are otherwise product dependent, the exact relationship between **pageparams** values and page device settings will vary from product to product.

There is no **pageparams** equivalent for page device **Orientation** values 2 and 3. Such pages will claim to have **pageparams** values with an **Orientation** of 0 or 1.

Errors: (none)

pagestackorder– **pagestackorder** *boolean*

returns the logical complement of the page device parameter **OutputFaceUp**. For example, if the value of **OutputFaceUp** is *true*, *boolean* will be *false*.

The page device parameter **OutputFaceUp** must be present for the compatibility operator **pagestackorder** to be present.

Errors: **stackoverflow**

printername† *string* **printername** *substring*

stores the value of the system parameter **PrinterName** in *string*, and returns a string object designating the *substring* actually used.

Errors: **rangecheck**, **stackunderflow**, **typecheck**

processcolors **– processcolors** *int*

returns the number of device process color components in the current page device (1 for black, 3 for RGB or CMY, or 4 for CMYK). The **statusdict** compatibility operator **processcolors** is mandatory on products that can produce more than one color; it is optional on monochrome products. Traditionally, this compatibility operator does not appear on monochrome printers. Its absence indicates a monochrome-only device (one process color).

Errors: **stackoverflow**

product† – **product** *string*

returns a string in **statusdict**. This string is initialized to the value of the string **product** in the dictionary **systemdict**.

Errors: **stackoverflow**

ramsize – **ramsize** *int*

returns the number of bytes of RAM available to the product. See the **RamSize** system parameter.

Errors: **stackoverflow**

realformat† – **realformat** *string*

returns a string with the same value as the system parameter **RealFormat**.

Errors: **stackoverflow**

resolution – **resolution** *bitsperinch*

returns the first component of the **HWResolution** array for the current output device.

Errors: **stackoverflow**

revision† – **revision** *int*

an integer with the same value as the system parameter **Revision**.

Errors: **stackoverflow**

sccbatch *channel* **sccbatch** *baud options*

returns the serial communications device parameter settings. The values are from either the %SerialB_NV% (if *channel* equals 9) or the %Serial_NV% (if *channel* equals 25) parameter set. The value of *options* is encoded as described above, and the values for data bits and parity are determined as described in Section 9.3.3, “SCC Operations,” on page 235. The values for baud, stop bits, and flow control are determined from the corresponding settings for the **Baud**, **StopBits**, and **FlowControl** entries in the %Serial% parameter set, respectively.

Note *If the **FlowControl** parameter is set to **DtrLow**, **sccbatch** will return 1 in bit positions 4, 3, and 2. If the **FlowControl** parameter is set to something other than **XonXoff**, **Dtr**, **DtrLow**, or **EtxAck**, **sccbatch** will return 0 in bit positions 4, 3, and 2.*

The %Serial_NV% or %SerialB_NV% parameter set must be present for the compatibility operator **sccbatch** to be present.

Errors: **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**

sccinteractive‡ *channel* **sccinteractive** *baud options*

pops the input argument off the stack and pushes 0 0 on the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackoverflow**, **stackunderflow**, **typecheck**

setaccuratescreens[ø] *boolean* **setaccuratescreens** –

sets the user parameter **AccurateScreens** to have the value of *boolean*.

Errors: **stackunderflow**, **invalidaccess**, **typecheck**

setdefaulttimeouts[†§¶] *job manualfeed wait* **setdefaulttimeouts** –

sets the system parameters **JobTimeout** and **WaitTimeout** to *job* and *wait*, respectively, and sets the page device parameter **ManualFeedTimeout** to *manualfeed*. The operator **setdefaulttimeouts** always takes three values, even if the corresponding system or page device parameters are not present.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setdostartpage[§] *boolean* **setdostartpage** –

sets the system parameter **DoStartPage** to the value of *boolean*.

The system parameter **DoStartPage** must be present for the compatibility operator **setdostartpage** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setdoprinterrors[§] *boolean* **setdoprinterrors** –

sets the system parameter **DoStartPage** to the value of *boolean*.

The system parameter **DoStartPage** must be present for the compatibility operator **setdostartpage** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setdosysstart[§] *boolean* **setdosysstart** –

sets the system parameter **StartupMode** according to the value of *boolean*. The system parameter **StartupMode** is set to 1 if *boolean* is *true* and set to 0 if *boolean* is *false*.

The system parameter **StartupMode** must be present for the compatibility operator **setdosysstart** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setduplexmode¶ *boolean* **setduplexmode** –

sets the page device parameter **Duplex** to *boolean*.

The page device parameter **Duplex** must be present for the compatibility operator **setduplexmode** to be present.

Errors: **stackunderflow**, **typecheck**

sethardwareiomode‡§ *int* **sethardwareiomode**

opens specified channel(s) for communications and closes all other channels. The variable *int* specifies which communication channel(s) should be opened by setting the **On** and **Enabled** parameters to *true*. All other channels will be explicitly closed by setting the **On** and **Enabled** parameters to *false*. The interpretation of *int* is as follows:

0	Open %Serial% and %SerialB%. Close all others.
1	Open %Parallel%. Close all others.
2	Open %LocalTalk% and %EtherTalk% (if both exist). Close all others.
	Open %LocalTalk% (if only %LocalTalk% exists). Close all others.
	Open %EtherTalk% (if only %EtherTalk% exists). Close all others.
3	Open %Serial% and %SerialB%. Close all others.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setjobtimeout‡ *int* **setjobtimeout** –

sets the user parameter **JobTimeout** to the value of *int*.

The user parameter **JobTimeout** must be present for the compatibility operator **setjobtimeout** to be present.

Errors: **stackunderflow**, **typecheck**

setmargins‡§¶ *top left* **setmargins** –

sets the page device **Margins** parameter to [*left top*].

The page device parameter **Margins** must be present for the compatibility operator **setmargins** to be present.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setmirrorprint¶ *boolean* **setmirrorprint** –

creates a new page device with the parameter **MirrorPrint** set to *boolean*.

Errors: **stackunderflow**, **typecheck**

setpageø¶ *width height orientation* **setpage** –

creates a new page device with the parameter **PageSize** set to the setpage values [*height width*] if the **setpage** *orientation* value is 0, and set to [*width height*] if the **setpage** *orientation* is 1.

The page device parameter **Orientation** will also be set as appropriate for the product. Because the relationship between **Orientation** and the **setpage** parameter *orientation* is product dependent, the exact relationship between **setpage** values and page device settings will vary from product to product.

Errors: **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**

setpagemarginø¶ *margin* **setpagemargin** –

creates a new page device with the parameter **PageOffset** set to [*margin* 0].

Errors: **rangecheck**, **stackunderflow**, **typecheck**

setpageparamsø¶ *width height margin orientation* **setpageparams** –

creates a new page device with the parameter **PageSize** set to **setpageparams** values [*height width*] if the **setpageparams** *orientation* value is 0, and set to [*width height*] if the **setpageparams** *orientation* is 1.

Page device parameters **PageOffset** and **Orientation** will also be set as appropriate for the product. Because the relationship between these page device parameters and the **setpageparams** parameters *margin* and *orientation* is product dependent, the exact relationship between **setpageparams** values and page device settings will vary from product to product.

Errors: **limitcheck**, **rangecheck**, **stackunderflow**, **typecheck**, **undefinedresult**

setpagestackorder§¶ *boolean* **setpagestackorder** –

sets the page device parameter **OutputFaceUp** to the logical complement of *boolean*. For example, if the value of *boolean* is *true*, **OutputFaceUp** is set to *false*.

The page device parameter **OutputFaceUp** must be present for the compatibility operator **setpagestackorder** to be present.

Errors: **invalidaccess**, **stackunderflow**, **typecheck**

setprintername‡§ *string* **setprintername** –

sets the system parameter **PrinterName** to the value of *string*.

The system parameter **PrinterName** must be present for the compatibility operator **setprintername** to be present.

Errors: **invalidaccess**, **limitcheck**, **stackunderflow**, **typecheck**

setresolution¶ *bitsperinch* **setresolution** –

creates a new page device with the parameter **HWResolution** set to [*bitsperinch bitsperinch*].

Errors: **rangecheck**, **stackunderflow**, **typecheck**

setsccbatch§ *channel baud options* **setsccbatch** –

sets the communications device parameters for serial communications. Either the %SerialB_NV% (if *channel* equals 9) or the %Serial_NV% (if *channel* equals 25) settings are affected. The following device parameters are affected by *baud* and *options*: **Baud**, **StopBits**, **DataBits**, **FlowControl**, **Parity**, and **CheckParity**. **Baud**, **StopBits**, and **FlowControl** are set according to the corresponding values for baud, stop bits, and flow control in the *options* argument. **DataBits** and **Parity** are set described in Section 9.3.3, “SCC Operations,” on page 235**. CheckParity** is set according to the new **Parity** setting:

- *true* if the setting is /Odd or /Even
- *false* if the setting is /Space or /Mark
- Not changed if the setting is /None (parity checking is not done if **Parity** is independent of the setting of **CheckParity**)

The %Serial_NV% or %SerialB_NV% device parameter set must be present for the compatibility operator **setsccbatch** to be present.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setsccinteractive‡§ *channel baud options* **setsccinteractive** –

pops the three input arguments off the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

setsoftwareiomode‡§ *int* **setsoftwareiomode** –

sets the values of the **Interpreter** parameter, and, if appropriate, **Protocol** device parameters for the current communications device parameter set, as indicated by the system parameter **CurInputDevice**. The meaning of *int* is as follows:

int	*Interpreter value*	*Protocol value*
0	**PostScript**	**Normal**
1	**ProprinterXL**	**Raw**
2	**Diablo630**	**Raw**
3	(Reserved)	
4	**HP7475A**	**Raw**

5	**LaserJetIIP**	**Raw**
	(If the LaserJet IIP emulator is present in the product)	
5	**LaserJetIII**	**Raw**
	(If the LaserJet III emulator is present in the product)	
100	**PostScript**	**Binary**

Note *A product will probably never have both the LaserJet IIP and LaserJet III emulators installed. If a product does have both emulators installed, passing a value of 5 to the* ***setsoftwareiomode*** *operator will select only LaserJet IIP.*

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

settumble¶ *boolean* **settumble** –

sets the page device parameter **Tumble** to *boolean.*

The page device parameter **Duplex** must be present for the compatibility operator **settumble** to be present.

Errors: **stackunderflow, typecheck**

setuserdiskpercent§ *int* **setuserdiskpercent** –

pops *int* off the stack. This operator is essentially a no-operation instruction.

Errors: **invalidaccess**, **rangecheck**, **stackunderflow**, **typecheck**

softwareiomode‡ – **softwareiomode** *int*

returns *int*, which indicates the interpretation mode for the current communications device as indicated by the system parameter **CurInputDevice**. See also the **setsoftwareiomode** operator.

Note *If the* ***Interpreter*** *is not one of the values that can be set using* ***setsoftwareiomode****,* the ***softwareiomode*** *operator will return –1.*

Errors: **stackoverflow**

tumble – **tumble** *boolean*

returns the value of the page device parameter **Tumble**.

The page device parameter **Duplex** must be present for the compatibility operator **tumble** to be present.

Errors: **stackoverflow**

userdiskpercent – **userdiskpercent** *int*

returns the value 0. This operator is essentially a no-operation instruction.

Errors: **stackoverflow**

waittimeout‡ – **waittimeout** *int*

returns an integer with the same value as the user parameter **WaitTimeout**. Redefining either **waittimeout** or the user parameter **WaitTimeout** redefines the other to the same value.

The user parameter **WaitTimeout** must be present for the compatibility key **waittimeout** to be present.

Errors: **stackoverflow**

11x17tray¶ **– 11x17tray –**

present only if the size [792 1224] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

Errors: **configurationerror, rangecheck, limitcheck**

9.3 Compatibility Operations

This section describes the LanguageLevel 1 compatibility objects (operators and keys) present in LanguageLevel 2. The majority of these LanguageLevel 1 objects are operators in **statusdict**. Other dictionaries may also contain compatibility objects (for example, **letter** in **userdict**). Compatibility objects need not always be operators (for example, the **waittimeout** integer in **statusdict**).

There is a LanguageLevel 2 method of performing most LanguageLevel 1 compatibility operations. For the following compatibility operators, however, there is currently no LanguageLevel 2 equivalent:

checkpassword†	**firstside**	**setsccinteractive**‡§
checkscreenø	**newsheet**	**setuserdiskpercent**§
devforall†	**sccinteractive**‡	**userdiskpercent**
emulate	**setpapertray**¶	

Discussing compatibility operators and keys in terms of LanguageLevel 2 operations not only provides the most accurate description of the compatibility operation, but it also indicates the correct LanguageLevel 2 method of carrying out the operation.

Because many of the compatibility operations originally dealt with product-specific behavior, the semantics of some operations in LanguageLevel 1 varied from one product to another. While defining compatibility operations in terms of product-independent LanguageLevel 2 operations corrects this problem, it sometimes does so at the cost of providing an imperfect emulation of the LanguageLevel 1 operation.

Some LanguageLevel 1 operations are no longer relevant for LanguageLevel 2 programs. In these cases, the compatibility operations may be implemented as no-operations that simply allow the LanguageLevel 1 programs that contain them to continue without generating errors. An example of such an operator is **setsccinteractive**.

9.3.1 Error Behavior

In general, the behavior in response to error conditions is different between the LanguageLevel 1 compatibility operation and the corresponding LanguageLevel 2 operation. This ensures that error behavior is as similar to LanguageLevel 1 error behavior as possible. For example, a LanguageLevel 1 paper tray operation such as **lettertray** may generate a **rangecheck** error, while the corresponding LanguageLevel 2 operation will generate a **configurationerror** error or will perform other actions under the control of the **Policies** entry in the page device dictionary.

9.3.2 Changing Persistent Values with Password

In LanguageLevel 1, many of the operations that changed persistent values could only be executed from jobs that had “exited the server” (this action required a password). If such an operation were executed without exiting the server, an **invalidaccess** error resulted.

In LanguageLevel 2, the notion of exiting the server has been replaced by the concept of an unencapsulated job. See Section 3.7.7 in the *PostScript Language Reference Manual, Second Edition*. An unencapsulated job is entered by executing the LanguageLevel 2 operator **startjob** or the LanguageLevel 1 operator **exitserver**. These operators require presentation of a password. The password must be equal to the value of either the **StartJobPassword** or the **SystemParamsPassword** system parameter. If the password is equal to the value of **StartJobPassword**, an ordinary unencapsulated job is started. If the password is equal to the value of **SystemParamsPassword**, a system administrator job is started. (If the **SystemParamsPassword** is a zero-length string or has never been set, every unencapsulated job is a system administrator job.)

Many compatibility operators and keys change system or device parameters. Such operators and keys use the LanguageLevel 2 **setsystemparams** or **setdevparams** operator to emulate the LanguageLevel 1 functionality. Those operators ordinarily require a **Password** parameter to be presented on each execution. This requirement is relaxed during a system administrator job, but not during an ordinary unencapsulated job. Since the compatibility operators and keys do not present a password, they can be successfully executed only during a system administrator job. Executing them during an ordinary unencapsulated job (or any encapsulated job) will cause an **invalidaccess** error.

Compatibility operators and keys that affect page device parameters save their persistent values only if they are executed from an unencapsulated job. In encapsulated jobs, the values set by these compatibility operators and keys will obey the normal save-restore rules and are not saved to persistent storage.

Note *The compatibility operators and keys are present in LanguageLevel 2 implementations for compatibility purposes only; their use in LanguageLevel 2 implementations is strongly discouraged.*

9.3.3 SCC Operations

SCC operators use a *byte options argument* (an integer parameter with values in the range 0 to 255) that holds an encoding of four SCC parameters. The four parameters are stop bits, data bits, flow control, and parity. The byte is encoded as described in Table 9.1 through Table 9.4 (bit positions 7–0, with 7 being the high bit and 0 being the low bit):

Table 9.1 *Stop bits*

Position 7	*Stop bits*
0	1 stop bit
1	2 stop bits

Table 9.2 *Data bits*

Positions 6 and 5	*Data bits*
0	Standard
1	7 bits
2	8 bits

Table 9.3 *Flow control*

Positions 4, 3, and 2	*Flow control*
0	Xon/Xoff
1	Dtr
2	Etx/Ack

Table 9.4 *Parity*

Positions 1 and 0	*Parity*
0	Space
1	Odd
2	Even
3	Mark

In LanguageLevel 1, the data bits and parity interacted nonorthogonally to produce a table of possible choices for data and parity that included many commonly desired methods of sending data. The “standard” data bits setting was present only for backward compatibility purposes with earlier versions of the SCC operators. In particular, a standard data bit setting could always be achieved with either a 7- or an 8-bit data setting. In LanguageLevel 2, there are analogous entries for the %Serial% and %SerialB% parameter sets.

The mapping between LanguageLevel 1 stop bits and flow control, and the LanguageLevel 2 %Serial% parameters **StopBits** and **FlowControl** is straightforward. It is not possible to provide such a one-to-one correspondence between the LanguageLevel 1 notion of data bits and parity and the LanguageLevel 2 %Serial% parameters **DataBits** and **Parity**. Tables 9.5 and 9.6 show the conversion between LanguageLevel 1 data bits and parity and LanguageLevel 2 **DataBits** and **Parity**. Notice that in going from **DataBits** and **Parity** to data bits and parity, standard parity is never used.

Table 9.5 *Options byte to device parameters conversion*

data bits and parity —>	***DataBits*** *and* ***Parity***
standard space	7 bits /Space
standard mark	8 bits /None
standard odd	7 bits /Odd
standard even	7 bits /Even
7 bits space	7 bits /Space
7 bits mark	7 bits /Mark
7 bits odd	7 bits /Odd
7 bits even	7 bits /Even
8 bits space	8 bits /None
8 bits mark	8 bits /None
8 bits odd	8 bits /Odd
8 bits even	8 bits /Even

Table 9.6 *Positions 1 and 0*

DataBits *and* ***Parity*** *—>*	*data bits and parity*
7 bits /None	7 bits mark
7 bits /Space	7 bits space
7 bits /Mark	7 bits mark
7 bits /Odd	7 bits odd
7 bits /Even	7 bits even
8 bits /None	8 bits mark
8 bits /Space	8 bits space
8 bits /Mark	8 bits mark
8 bits /Odd	8 bits odd
8 bits /Even	8 bits even

Tables 9.5 and 9.6 provide the best compatibility with LanguageLevel 1 behavior. In several cases, no correct choice is possible. For example, LanguageLevel 1 had no support for 7 data bits with no parity (that is, the total number of data and parity bits is 7). The LanguageLevel 2 setting of 7 bits /None is imperfectly mapped to 7 bits mark. Most serial hardware does not support 8-bit mark or space, which is why these values are never

generated in mapping from LanguageLevel 1 to LanguageLevel 2. In fact, in LanguageLevel 1, 8 bits mark and 8 bits space actually provided the equivalent of the LanguageLevel 2 8 bits /None capability.

9.3.4 Paper Size Operations

All the paper size operators are in the **userdict** dictionary. Each operator executes the **setpagedevice** operator to request a specific paper size. The only difference among these operations is the size of paper requested and the **ImagingBBox** value. The "***small**" operators (such as **lettersmall** and **a4small**) specify a non-*null* value of **ImagingBBox**; other operators specify a *null* value of **ImagingBBox**. These operators use the specified size, as indicated below, as a page device parameter **PageSize**. In addition, all these operators set the **Policies PageSize** to 7 (which guarantees that the **PageSize** established is the requested size, regardless of the actual size of the medium) and turn off the normal LanguageLevel 2 media matching mechanism. For a detailed description of **Policies PageSize** 7, see Table 4.24 on page 124. The only error that is generated is a **limitcheck** error caused by insufficient memory for the requested imaging area. In Table 9.7, default units (1/72 inch) are used as the units for the **PageSize** and **ImagingBBox**.

Table 9.7 *Paper size compatibility operators (in userdict)*

Operator	***PageSize***	***ImagingBBox***
letter‡¶	[612 792]	*null*
lettersmall¶	[612 792]	[25 25 587 767]
legal‡¶	[612 1008]	*null*
ledger¶	[1224 792]	*null*
11x17¶	[792 1224]	*null*
a4¶	[595 842]	*null*
a3¶	[842 1191]	*null*
a4small¶	[595 842]	[25 25 570 817]
b5¶	[516 729] or [499 709]	*null*
note¶	[*width height*]	[25 25 *width*-25 *height*-25]

The **note** compatibility operator will be present only if the size [*width height*] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **letter** and **lettersmall** compatibility operators will be present only if the size [612 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **legal** compatibility operator will be present only if the size [612 1008] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **a4** and **a4small** compatibility operators will be present only if the size [595 842] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **b5** compatibility operator will be present only if the size [516 729] or the size [499 709] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

9.3.5 Paper Tray Operations

All of the operators in this section are in the **statusdict** dictionary. Each operator executes the **setpagedevice** operator to request a tray containing a specific paper size. The only difference among these operations is the size of paper requested. The value of the **PageSize** parameter requested is the same as for the corresponding paper size operator discussed in the previous section, and the value of **ImagingBBox** requested is always *null*. These operators use the specified size as indicated in Table 9.8 as a page device **PageSize** parameter.

All of these operators set the value of the **Policies PageSize** entry to 0, which guarantees that a **configurationerror** error is generated if a tray containing the requested paper size is not present. The implementation of the compatibility operators convert any such **configurationerror** to a **rangecheck** error for compatibility with LanguageLevel 1 implementations. Also, a **limitcheck** error can occur because of insufficient memory for the requested imaging area.

Table 9.8 *Paper tray compatibility operators*

Operator	***PageSize***	***ImagingBBox***
lettertray †¶	[612 792]	*null*
legaltray¶	[612 1008]	*null*
ledgertray¶	[1224 792]	*null*
a3tray¶	[842 1191]	*null*
a4tray¶	[595 842]	*null*
b5tray¶	[516 729] or [499 709]	*null*
11x17tray¶	[792 1224]	*null*

The **lettertray** compatibility operator will be present only if the size [612 792] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **legaltray** compatibility operator will be present only if the size [612 1008] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **a4tray** compatibility operator will be present only if the size [595 842] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

The **b5tray** compatibility operator will be present only if the size [516 729] or the size [499 709] is an element of the **PageSize** array in some instance of the **OutputDevice** resource category.

CHAPTER 10

Additions and Modifications to the Interpreter Parameters

This chapter supplements Appendix C in the *PostScript Language Reference Manual, Second Edition.* It provides information about new and modified user, system, and device parameters. Parameters modified in response to LanguageLevel 3 are shaded .

Overview of This Chapter

10.1 Overview 243

10.2 User Parameters 243

User parameters (Table 10.1) 244

10.3 System Parameters 247

System parameters (Table 10.2) 248

10.4 Administrator Jobs and Passwords 258

10.4.1 Unencapsulated Jobs 258

10.4.2 Passwords for System and Device Parameters 259

10.5 Device Parameters 259

Device Parameter Interdependencies 260

Parameter Sets 260

Device Parameter: Type 260

The OSI (Open Systems Interconnection) layers (Figure 10.1) 261

10.5.1 Communications Parameter Sets (Type = /Communications) 262

Relationship between network communications protocols and physical communications media (Figure 10.2) 263

Setting Up Communications Parameter Sets 264

%CommName_NV%, %CommName%, and %CommName_Pending% 264

Hierarchy of %CommName_NV%, %CommName%, and %CommName_Pending% 265

Relationship among the communication parameter sets (Figure 10.3) 265

Multiple Nonvolatile Parameter Sets (%CommName_NVn%) 266

Communications parameter sets using NV values (Figure 10.4) 267

Predetermined (Hard-Wired) Parameter Values 267

Communications parameter sets using “hard-wired” values (Figure 10.5) 268
Entries Found in All Parameter Sets of Type /Communications 269
Entries in all communications parameter sets (Type = /Communications) (Table 10.3) 269
Point-to-Point Communications Parameter Sets 274
Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (Table 10.4) 274
Entries in the %Parallel%, %ParallelB%, %ParallelC%, … parameter sets (Table 10.5) 279
Entries in the %ScsiComm%, %ScsiCommB%, %ScsiCommC% … parameter sets (Table 10.6) 281
Network Communications Parameter Sets 283
Entries in the %LPR%, %LPRB%, %LPRC%, … parameter sets (Table 10.7) 286
Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets (Table 10.8) 288
Entries in the %Telnet%, %TelnetB%, %TelnetC%, … parameter sets (Table 10.9) 291
Entries in the %RemotePrinter%, %RemotePrinterB%, %RemotePrinterC%, … parameter sets (Table 10.10) 292
Entries in the %PrintServer%, %PrintServerB%, %PrintServerC%, … parameter set (Table 10.11) 293
Entries in the %LAT%, %LATB%, %LATC%, … parameter sets (Table 10.12) 295
Entries in the %LocalTalk%, %LocalTalkB%, %LocalTalkC% … parameter sets (Table 10.13) 297
Entries in the %EtherTalk%, %EtherTalkB%, %EtherTalkC%, … parameter sets (Table 10.14) 299
Entries in the %TokenTalk%, %TokenTalkB%, %TokenTalkC%, … parameter sets (Table 10.15) 301
10.5.2 Parameters Parameter Sets (Type = /Parameters) 303
Node Address 308
Form of the node address (Table 10.16) 308
Required Entries for the /Parameter Parameter Sets 308
Entries in the %SNMP%, %SNMPB%, %SNMPC%, … parameter sets (Table 10.17) 308
Entries in the %Syslog%, %SyslogB%, %SyslogC%, … parameter sets (Table 10.18) 310
Entries in the %TCP%, %TCPB%, %TCPC%, … parameter sets (Table 10.19) 311
Entries in the %UDP%, %UDPB%, %UDPC%, … parameter sets Table 10.20) 311
Entries in the %IP%, %IPB%, %IPC%, … parameter sets (Table 10.21) 312
Entries in the %SPX%, %SPXB%, %SPXC%, … parameter sets (Table 10.22) 316
Entries in the %IPX%, %IPXB%, %IPXC%, … parameter sets (Table 10.23) 316
Entries in the %EthernetPhysical%, %EthernetPhysicalB%, %EthernetPhysicalC%, … parameter sets (Table 10.24) 318
Entries in the %TokenRingPhysical%, %TokenRingPhysicalB%, %TokenRingPhysicalC%, … parameter sets (Table 10.25) 319

Entries in the %Scsi%, %ScsiB%, %ScsiC%, … parameter sets (Table 10.26) 320
Entries in the %Ide%, %IdeB%, %IdeC%, … parameter sets (Table 10.27) 322
Entries in the %Engine%, %EngineB%, %EngineC%, … parameter sets (Table 10.28) 322
Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets (Table 10.29) 324
Entries in the %Calendar%, %CalendarB%, %CalendarC%, … parameter sets (Table 10.30) 327
10.5.3 File System Parameter Sets (Type = /FileSystem) 328
Entries in the %disk0%, %disk1%, %disk2%, … parameter sets (Table 10.31) 330
Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets (Table 10.32) 334
Entries in the %ram%, %ram1%, %ram2%, … parameter sets (Table 10.33) 337
Entries in the %os%, %osB%, %osC%, … parameter sets (Table 10.34) 339

10.1 Overview

The various interpreter parameters control the operation and behavior of the PostScript interpreter. Many of these parameters have to do with allocation of memory and other resources for specific purposes. For example, there are parameters to control the maximum amount of memory used for VM, font cache, and halftone screens. Some input/output devices have parameters that control the behavior of each device individually.

A printer system is initially configured with interpreter parameter values that are appropriate for most applications. However, a PostScript program can change the interpreter parameters to favor a certain type of functionality or to adapt the product to special requirements. There are three classes of interpreter parameters: user, system, and device parameters.

For each class there is a PostScript operator to read the parameter values and an operator to set the parameter values. These six operators are **currentuserparams**, **setuserparams**, **currentsystemparams**, **setsystemparams**, **currentdevparams**, and **setdevparams**.

10.2 User Parameters

Any PostScript program can set user parameters during job execution; no password is required. The initial value of user parameters when the printer system is turned on for the first time is product dependent.

Unless otherwise specified, all user parameters are subject to the operators **save** and **restore**. (At this time, **JobTimeout** is the only parameter that does not obey the **save** and **restore** operators.) This means that if an unencapsulated job changes user parameters, these new values are the initial values for subsequent encapsulated jobs. There are exceptions to this generalization. For a system parameter whose name is the same as a user parameter, the value of the system parameter is used to initialize the corresponding user parameter at the beginning of each job. In any case, changes made to any user parameter by an encapsulated job have no effect on the initial value of user parameters for subsequent jobs.

User parameters are maintained on a per-context basis in environments that support multiple contexts.

The following user parameters are described in Table C.1 of Appendix C in the *PostScript Language Reference Manual, Second Edition*. The description of these parameters is unchanged.

MaxDictStack	**MaxLocalVM**	**MaxUPathItem**
MaxExecStack	**MaxOpStack**	**MinFontCompress**
MaxFontItem	**MaxPatternItem**	**VMReclaim**
MaxFormItem	**MaxScreenItem**	**VMThreshold**

Each user parameter is identified by a key, which is always a name object. The value of the parameter is usually an integer.

Table 10.1 describes user parameters that have been defined or amended since publication of the *PostScript Language Reference Manual, Second Edition.*

Note *In the following table, the symbol ‡ means that the key is typically present in all job server implementations (that is, printer implementations). Error listings are confined to device-specific errors.*

Table 10.1 *User parameters*

Key	*Type*	*Semantics*
AccurateScreens	boolean	Controls whether the accurate screen algorithm is used during subsequent executions of the **setscreen** and **setcolorscreen** operators. This parameter has no effect on screens established by the **sethalftone** operator. See Section 6.4.4 in the *PostScript Language Reference Manual, Second Edition*, for a description of accurate screening for the **sethalftone** operator. **Legal values**: *true*, *false* **Errors**: None

Table 10.1 *User parameters (continued)*

Key	*Type*	*Semantics*

HalftoneMode	integer	**HalftoneMode** affects the behavior of subsequent halftone setting operators such as **setscreen**, **setcolorscreen**, and **sethalftone** as explained below. It has no effect on the current halftone. Legal values for **HalftoneMode** are 0, 1, and 2. If **HalftoneMode** is set to a value other than 0, 1, or 2, the request is ignored and the current value is retained. However, since a product may choose not to support all of these values, if **HalftoneMode** is set to an unsupported (but legal) value, then the nearest supported value is substituted without indicating any error. (For example, if only 0 and 1 are supported, an attempt to set **HalftoneMode** to 2 will set it to 1, regardless of its original value.) The legal values are defined as follows:
		
		0 — This is the normal mode of operation. The behavior of **setscreen**, **setscolorscreen**, and **sethalftone** is not affected.
		
		1 — Executing **setcolorscreen**, **setscreen**, or **sethalftone** may cause these operators to ignore the halftone operand and substitute a product-specific halftone. Whether the operand halftone is actually substituted is product-specific. When the product-specific halftone is substituted, certain pages may print faster (this behavior is also product specific).
		
		2 — This is similar to the case when **HalftoneMode** is 1. In addition, if the product-specific halftone is substituted, further optimization may be employed when rendering images, resulting in further speed improvement, but at the expense of some degradation in image quality. The above additional speed optimization is disabled for rendering masked images, imagemasks, and images rotated at other than multiples of 90 degrees. (Images rotated at multiples of 90 degrees relative to the device coordinates—for example, “landscape” images—could be rendered faster with **HalftoneMode** 2 than with **HalftoneMode** 0.)
		
		Note *Since this parameter affects the behavior of subsequent halftone setting operations, to achieve the effect as specified above, a halftone setting operation (whether explicit or implicit) must be performed after **HalftoneMode** is set.*
		
		*For example, given a product that always replaces the user requested halftone with its specific halftone when **HalftoneMode** is not 0, consider the following operators:*
		`<</HalftoneMode x>> setuserparams`
		`currenthalftone sethalftone`
		
		When *x* is either 1 or 2, this **sethalftone** operation causes the product-specific halftone to be installed. If the **currenthalftone** and **sethalftone** operations were omitted, the halftone installed would remain unchanged.

Table 10.1 *User parameters* *(continued)*

Key	*Type*	*Semantics*
		Legal values: 0, 1, 2
		Errors: **typecheck**
IdiomRecognition‡	boolean	If *true*, enables procedure substitution during execution of the **bind** operator. If *false*, disables idiom recognition.
		Legal values: *true*, *false*
		Errors: **typecheck**
JobName‡	string	Establishes *string* as the name of the current job. If defined as a non-zero length string, status responses generated during the remainder of the current job will include a *job* field that reports the text of this string. The characters should be within the ASCII printable range because this information is transmitted across arbitrary communications channels and is intended for display to users.
		Legal values: Any sequence of byte values up to an implementation-dependent maximum length. The sequence should not contain the characters ";" or "]" because these characters disrupt the syntax of status messages. If the maximum length is exceeded, the string is truncated.
		Errors: **limitcheck**, **typecheck**
JobTimeout	integer	Setting the **JobTimeout** parameter to a positive value establishes this value as the current job time-out; that is, the number of seconds a job is allowed to execute before it is aborted and a PostScript **timeout** error is generated. The current value is decremented during the job, and reading it returns the number of seconds remaining before the job time-out will occur. Time spent waiting for communications and correcting device error conditions is not considered as part of the job execution time. Setting this parameter to 0 disables job time-out altogether.
		JobTimeout is not subject to the **save** and **restore** operators. It is initialized to the value of the **JobTimeout** system parameter at the beginning of each job.
		Legal values: Any positive integer or 0.
		Errors: **typecheck**
MaxSuperScreen	integer	Establishes an upper limit for the number of pixels in the supercell. The highest effective value is currently 1016. A value of 0 prevents the use of supercells. See also Section 6.3, "Halftone Dictionaries," on page 166.
		Legal values: Any positive integer.
		Errors: **typecheck**

Table 10.1 *User parameters* *(continued)*

Key	*Type*	*Semantics*
WaitTimeout ‡	integer	Indicates the current wait time-out, which is the number of seconds the interpreter waits to receive additional characters from the host before it aborts the current job by executing a PostScript **timeout** error. A value of 0 indicates an infinite time-out. This parameter is initialized to the value of the **WaitTimeout** system parameter at the beginning of each job. **Legal values**: Any positive integer or 0. **Errors**: **typecheck**

10.3 System Parameters

In general, setting system parameters requires a password.

System parameter values persist across jobs. Depending on the product, some system parameters are stored in nonvolatile memory and are persistent across restarts of the interpreter.

System parameters are global to the PostScript environment and are not maintained on a per-context basis in environments that support multiple contexts. The initial value of system parameters when the device is turned on for the first time and which parameters are stored in nonvolatile memory depends on the product implementation.

Some system parameters are *read-only*; that is, they are returned by the operator **currentsystemparams**, and any attempt to change such a system parameter using the operator **setsystemparams** has no effect. Other parameters are *write-only*; that is, they can be set by **setsystemparams** but are not returned by **currentsystemparams**.

Each system parameter is identified by a key, which is always a name object. The following system parameters are described in the *PostScript Language Reference Manual, Second Edition*; the description of these parameters is unchanged:

ByteOrder	**CurScreenStorage**	**MaxOutlineCache**
CurDisplayList	**CurUPathCache**	**MaxPatternCache**
CurFontCache	**DoPrintErrors**	**MaxScreenStorage**
CurFormCache	**MaxDisplayList**	**MaxUPathCache**
CurOutlineCache	**MaxFontCache**	**RealFormat**
CurPatternCache	**MaxFormCache**	

Table 10.2 describes system parameters that have been defined or amended since publication of the *PostScript Language Reference Manual, Second Edition*.

Note *In the following table, the symbol ‡ means that this key is typically present in all job server implementations (that is, printer implementations). Error listings are confined to device-specific errors.*

Table 10.2 *System parameters*

Key	*Type*	*Semantics*
BuildTime	integer	(*Read-only*) A time stamp identifying a specific build of the PostScript interpreter. The values returned by the **BuildTime** parameter on two different products need not be comparable. In general, **BuildTime** should only be interpreted in conjunction with a manufacturer's product documentation. **Legal values**: Any integer. **Errors**: None
CompressImageSource	boolean	When *true*, a compression filter will be applied to the image source data where such compression benefits the product in terms of memory use or performance. **Legal values**: *true*, *false* **Errors**: **typecheck**
CurBufferType	name	*(Read-only*) Indicates how the raster memory is used. The legal values are /Band and /Hybrid. /Band indicates that the system will render to bands (groups of scan lines), regardless of the amount of RAM available. /Hybrid indicates that if the **MaxRasterMemory** parameter is large enough to contain a full-page frame buffer, the interpreter will render into the full-page buffer; otherwise, it will render to bands. This parameter is typically found on imagesetters. **Legal values**: /Band, /Hybrid **Errors**: None
CurInputDevice	string	(*Read-only*) Indicates the name of the communications device corresponding to the current input file for the currently executing PostScript program. The string that is returned corresponds to the communications device parameter set whose values are normally stored in RAM—for example, (%Serial%). For further information on communications devices, see Section 10.5.1, "Communications Parameter Sets (Type = /Communications)." **Legal values**: A string containing a communications device name. **Errors**: None

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
CurOutputDevice	string	(*Read-only*) Indicates the name of the communications device corresponding to the current output file for the currently executing PostScript job. The string that is returned corresponds to the communications device parameter set whose values are normally stored in RAM—for example, (%Serial%). For further information on communications devices, see Section 10.5.1, "Communications Parameter Sets (Type = /Communications)." **Legal values**: A string containing a communications device name. **Errors**: None
CurSourceList‡	integer	(*Read-only*) Indicates the number of bytes currently occupied by source lists. A source list holds the internal data representation for sampled image source data and uncached character pixel arrays. **Legal values**: Any positive integer or 0. **Errors**: None
CurStoredFontCache	integer	(*Read-only*) Indicates the number of bytes that the storage device font cache currently occupies. **Legal values**: Any positive integer. **Errors**: None
CurStoredScreenCache	integer	(*Read-only*) This parameter indicates the number of bytes currently used for screen files on the storage device that includes the currently active screens. **Legal values**: Any positive integer or 0. **Errors**: None
DoPrintErrors	boolean	Indicates whether the built-in error handler for the product is enabled. All PostScript printer systems have an error handler to catch errors that are generated by programs. See Section 3.10.2 in the *PostScript Language Reference Manual, Second Edition*, for further information. Some printer systems also have a built-in error handler that can be enabled to print the error and stack contents on the current partial page, much like that described in Appendix A of *PostScript Language Program Design*. The

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
		system parameter **DoPrintErrors** determines whether this error printing is enabled. This system parameter is present only in printer systems that have such a built-in error handler. Any printer system that supports LaserJet 4 emulation will have one and can be controlled either from this system parameter or from the PJL commands `@PJL [SET | DEFAULT] LPARM: POSTSCRIPT PRTPSERRS = [ON | OFF]` **Legal values**: *true*, *false* **Errors**: **typecheck**
DoStartPage	boolean	Indicates whether the start page should print during system initialization. The start page prints if the value of **DoStartPage** is *true* during system initialization. **Legal values**: *true*, *false* **Errors**: **typecheck**
EnvironmentSave	boolean	In systems with multiple PDLs, the **EnvironmentSave** parameter controls whether the system can reclaim memory belonging to dormant PDLs when the system runs out of memory. If the value of **EnvironmentSave** is *true*, all permanent objects belonging to all PDLs persist across PDL switches. If the value of **EnvironmentSave** is *false*, all memory belonging to dormant PDLs can potentially be reclaimed when the system runs out of memory. Setting **EnvironmentSave** to *true* at low memory configurations could make the system essentially unusable. In low memory configurations, this parameter should always be *false*. When the memory installed in the system is above the product-defined limit, this parameter can be set by the user. When the value of **EnvironmentSave** is changed, the new value is effective immediately (the system does not have to be rebooted). Default value: product specific **Legal values**: *true*, *false* **Errors**: **typecheck**
FactoryDefaults	boolean	Usually *false*. Setting this value to *true* and immediately turning off the printer system causes all nonvolatile parameters to revert to factory default values at the next power-on. The job that sets the value of the **FactoryDefaults** parameter to *true* must be the last job executed before power-off; otherwise, the request is ignored. This requirement reduces the chance of malicious jobs resetting the device to factory defaults.

Table 10.2 *System parameters* *(continued)*

Key	*Type*	*Semantics*
		A password is not required in the dictionary passed to the **setsystemparams** operator if **FactoryDefaults** is the only entry in the dictionary or if the only other entry is **PassWord**. This allows the factory default values to be reestablished even if the system parameters password has been corrupted.
	Note	*Passwords can be reset using this operation.*
		The set of parameters required to reset to factory default value using this action is product dependent. In most products, the **PageCount** parameter is not reset.
		See Section 10.4.2, “Passwords for System and Device Parameters,” on page 259.
		Legal values: *true, false*
		Errors: **typecheck**
FatalErrorAddress‡	integer	A fatal system software error causes a PostScript output device to stop execution and, in most products, to restart the PostScript interpreter. Before execution is stopped, the address at which the error occurs is stored in the **FatalErrorAddress** parameter and is transmitted to the host over the communications channel. A non-zero value of **FatalErrorAddress** indicates that a fatal system error has occurred earlier. On some products, if this value is non-zero during system initialization, the address is printed on the start page or possibly on a separate page.
		Legal values: Any integer.
		Errors: None
FontResourceDir‡	string	Controls the location of external fonts. Fonts are resources in LanguageLevel 2. The **Font** resource category implementation concatenates the value of **FontResourceDir** and the font name to get the external location of the font. For example, if the value of **FontResourceDir** were (Resource/Font/), then the **Times-Roman** resource of the **Font** category would be in (Resource/Font/Times-Roman).
		This parameter is provided separately from the **GenericResourceDir** system parameter to allow backward compatibility with applications that expect fonts to be located under (fonts/). In such a case, the value of **FontResourceDir** should be set to (fonts/).

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
	Note	*Applications and users should access external fonts only through the resource operators or the **findfont** operator or, if the fonts need to be accessed as files, through the **ResourceFileName** procedure. See Section 3.9 in the PostScript Language Reference Manual, Second Edition. The above parameter should be used only to control the location of external fonts by the resource management mechanism.* **Legal values**: Any string of non-null characters. **Errors**: **limitcheck**, **typecheck**
GenericResourceDir‡ and **GenericResourcePathSep**‡	string	Control the location of external resources for the **Generic** resource category and all categories based upon it (for example, **Category**, **Encoding**, **Form**, **Pattern**, **ProcSet**, **ColorSpace**, **Halftone**, and **ColorRendering**). The **Generic** category implementation concatenates the value of **GenericResourceDir**, the category name, the value of **GenericResourcePathSep**, and the resource name to get the external location of the resource. If the values of **GenericResourceDir** and **GenericResourcePathSep**, for example, were (Resource/) and (/), respectively, then the **AdobeLogo** resource of the **Pattern** category would be in Resource/Pattern/AdobeLogo. The value of **GenericResourceDir** should be an absolute path—that is, a path beginning at the root of the storage device. It must contain any trailing path separator. It should include a storage device (for example, %os%) only if a single device is to be considered, or it should omit the device if all searchable devices are to be considered. If a device is dedicated to generically managed resources (for example, %GenericResource%) that may access resources through a network server or along a search path, then **GenericResourceDir** should be set to that device. Resource files are expected to be in subdirectories with names that correspond to category names. The resource file name should be the same as the name of the resource it defines. In the above example, the file named Resource/Pattern/AdobeLogo should contain a PostScript program that, when run, will define the **AdobeLogo** instance in the **Pattern** resource category.
	Note	*Applications and users should access external resources only through the resource operators or, if the external resources must be accessed as files, through the **ResourceFileName** procedure. See Section 3.9 in the PostScript Language Reference Manual, Second Edition. The above parameters should be used only to control the location of external resources by the resource management mechanism.* For products with no external resources (and presumably no file systems), the value of **GenericResourceDir** should be set to (%null). This mechanism can also be used by site administrators to temporarily disable access to external resources.

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
		Legal values: Any string of non-null characters.
		Errors: **limitcheck**, **typecheck**
InstalledRam	integer	(*Read-only*) Indicates, in bytes, the total amount of installed RAM in the system. (**InstalledRam** should not be confused with **RamSize**, which is the amount of RAM available to the page description language.)
		Legal values: Any positive integer.
		Errors: None
JobTimeout	integer	Indicates the value, in seconds, to which the user parameter **JobTimeout** is initialized at the beginning of each job. The minimum value for **JobTimeout** is 15; setting a number between 1 and 14 will result in 15 being set. (If smaller values were allowed, this might prevent a subsequent job from successfully setting the **JobTimeout** parameter to another value.) A negative value will be ignored and the previous setting of **JobTimeout** used. A value of 0 indicates that the time-out is infinite.
		Legal values: 0 or any integer greater than or equal to 15.
		Errors: **typecheck**
LicenseID	string	Contains the Adobe-assigned license identifier. Its value is unique to each product.
		Legal values: Any string of non-null characters.
		Errors: **limitcheck**, **typecheck**
MaxDisplayAndSourceList	integer	Limits the amount of memory that can be used by the display list and source list. The value of **MaxDisplayAndSourceList** should be greater than the value of either of the two related parameters, **MaxDisplayList** and **MaxSourceList**.
		Legal values: Any integer.
		Errors: None
MaxHWRenderingBuffer	integer	Indicates the amount of memory, in bytes, to reserve for use by hardware rendering devices, such as PixelBurst or ColorBurst, to store display list data. The memory is permanently allocated during system initialization. If the value being set is outside of the legal range, **MaxHWRenderingBuffer** is set to the nearest acceptable value. The minimum value meets the requirements of the rendering device, and the maximum value is an amount that will not jeopardize execution of a PostScript job.
		Legal values: Product dependent. Any positive integer, typically 8192 or greater.
		Errors: None

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
MaxImageBuffer‡	integer	Indicates the maximum number of bytes that can be allocated for a single image buffer. An image buffer holds an internal data representation for sampled image source data. The parameter may be rounded by the interpreter if a requested value is out of range. **Legal values**: Any integer. **Errors**: **typecheck**
MaxPermanentVM	integer	Defines the upper limit, in bytes, of the amount of permanent VM that can be downloaded when the value of **EnvironmentSave** is *true*. The upper limit of permanent VM is the VM at save level zero defined by unencapsulated PostScript jobs. This limit is not enforced if the value of **EnvironmentSave** is *false*. If the value of **EnvironmentSave** is *true*, any attempt to download more permanent VM than is defined by **MaxPermanentVM** will generate a **VMerror**. If a user attempts to set the value of **MaxPermanentVM** to less than the current permanent VM plus a small threshold value, **MaxPermanentVM** will default to the smallest allowable value. Whenever the value of **EnvironmentSave** is reset to *true*, if **MaxPermanentVM** is set to a value lower than the smallest allowable value, **MaxPermanentVM** will default to the minimum allowable value. **MaxPermanentVM** is present in the system parameter set only if **EnvironmentSave** is defined. **Legal values**: Any integer. **Errors**: None
MaxRasterMemory	integer	Indicates the largest amount of memory, in bytes, that may be allocated to the frame buffer. This parameter may be used to limit the amount of raster memory; unused raster memory is available for use as VM. Thus, the **MaxRasterMemory** parameter allows the user to trade off raster memory allocation (which allows larger page sizes and higher resolutions) against VM (which allows more downloaded fonts and the production of more complex pages). The **MaxRasterMemory** parameter is consulted only during system initialization; any changes to the value of the parameter will not take effect until then. **Legal values**: Product dependent. **Errors**: **typecheck**

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
MaxSourceList‡	integer	Indicates the maximum number of bytes that can be allocated for source lists. A source list holds internal data representation for sampled image source data and uncached character pixel arrays. This parameter may be rounded by the interpreter if a requested value is out of range. **Legal values**: Any positive integer. **Errors**: **typecheck**
MaxStoredFontCache	integer	Defines the maximum number of bytes that the storage device font cache can occupy on the chosen storage device (such as the disk). Setting **MaxStoredFontCache** to 0 turns off stored caching. Setting **MaxStoredFontCache** to −1 (or to a positive value too large) sets the number of bytes that the font cache can occupy to the logical size of the storage device. If the logical size of the storage device is not known, an implementation-dependent value is used. **Legal values**: −1, 0, or any positive integer. **Errors**: **typecheck**
MaxStoredScreenCache	integer	Defines the maximum number of bytes that the storage device screen cache can occupy on the chosen storage device. Setting the value of **MaxStoredScreenCache** to 0 turns off stored caching. Setting the value of **MaxStoredScreenCache** to a negative value (or to a positive value too large) sets the number of bytes that the screen cache can occupy to the logical size of the storage device. If the logical size of the storage device is not known, an implementation-dependent value is used. **Legal values**: Any integer. **Errors**: **typecheck**
MinBandBuffers	integer	Used to specify the minimum number of band buffers (groups of scan lines) to be allocated from memory set aside as raster memory. The default value depends on the product and the amount of memory installed. This parameter is typically found on imagesetters, where the default is 2 for configurations with 32 MB of memory or less; otherwise it is 3. **Legal values**: Any positive integer. **Errors**: None

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
PageCount	integer	(*Read-only*) Indicates the number of pages that have successfully been processed since manufacture. The **PageCount** parameter is incremented when the interpreter finishes executing each page. The page count is incremented at these times by the value of the current copy count. If one or more pages are not actually printed for any reason, including manual feed time-out and job abort, the value of **PageCount** is not decreased accordingly. In most products, the value of **PageCount** is not reset at a user request to return to factory default values. However, the value of **PageCount** may be reset if the nonvolatile memory in which it is stored has been corrupted.
	Note	*In releases prior to Adobe PostScript software, version 2014, the **PageCount** parameter is incremented when a page completes printing, rather than during execution of the **showpage** or **copypage** operators.*
		Legal values: Any positive integer or 0.
		Errors: None
PrinterName	string	Establishes *string* as the current name of the device. If the device is on a network, this name might be used by the system as part of a name identifier for the device considered as a node on the network. The **PrinterName** parameter is usually printed on the start page, and thus it should consist of printable characters, although this is not required. Setting this parameter to a zero-length string causes **PrinterName** to be set to the value of the **product** string in the **systemdict** dictionary.
		Legal values: Any string of 32 or fewer non-null characters.
		Errors: **limitcheck**, **typecheck**
RamSize	integer	(*Read-only*) Indicates, in bytes, the amount of installed RAM available to the PDL. In some cases, this value might be less than the total amount of installed RAM in the product. For example, the system diagnostics might have determined that certain banks of RAM are defective, so it would consider them unavailable.
		Legal values: Any positive integer.
		Errors: None
Revision	integer	(*Read-only*) Designates the current revision level of the product in which the PostScript interpreter is running. Each product has its own numbering system for revisions, independent of those of any other product. The value is identical to the value of the integer **revision** in the **systemdict** dictionary.
		Legal values: Any integer.
		Errors: None

Table 10.2 *System parameters* *(continued)*

Key	*Type*	*Semantics*
SourceListBypass	boolean	Controls an optimization in the handling of image data. When image data has been stored in a format appropriate for final rendering, a product may delay reading the data until it is ready to render the image. Properties such as color space, storage method, and data interleaving determine whether the image data is eligible for the optimization. Format requirements are product specific. A value of *true* allows the interpreter to delay reading the data. A value of *false* disables the optimization. **Legal values**: *true*, *false* **Errors**: **invalidaccess**, **stackunderflow**, **typecheck**
StartJobPassword‡	string	If a program starts an unencapsulated job using the **startjob** or **exitserver** operator, and if the password it presents to that operator is the value of **StartJobPassword**, then the subsequent unencapsulated job will need to present a password equal to the value of **SystemParamsPassword** each time **setsystemparams**, **setdevparams**, or other system administrator operations are invoked. See Section 10.4.1, “Unencapsulated Jobs,” on page 258 and Section 10.4.2, “Passwords for System and Device Parameters,” on page 259. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **limitcheck, typecheck**
StartupMode	integer	Controls whether the system start file (Sys/Start) or some other start-up procedure should be executed during system initialization. The Sys/Start file executes if the value of **StartupMode** is 1 during system initialization. If the **StartupMode** value is 0, no special start-up procedures are run during system initialization. Other values of **StartupMode** can occur in specific products and result in a product-dependent start-up procedure execution. **Legal values**: Product dependent, but restricted to values between 0 and 255. **Errors**: **typecheck**

Table 10.2 *System parameters (continued)*

Key	*Type*	*Semantics*
SystemParamsPassword‡	string	If a program starts an unencapsulated job using the **startjob** or **exitserver** operator, and if the password it presents to that operator is the value of **SystemParamsPassword**, then the subsequent unencapsulated job is permitted to invoke **setsystemparams**, **setdevparams**, or other system administrator operations without presenting a password each time. This extends to LanguageLevel 1 compatibility operators that change system parameters but provide no means to present a password. See Section 10.4.1, "Unencapsulated Jobs," on page 258 and Section 10.4.2, "Passwords for System and Device Parameters," on page 259. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **limitcheck**, **typecheck**
ValidNV	boolean	(*Read-only*) Indicates whether nonvolatile memory is currently used to store persistent parameters. During system initialization, if nonvolatile memory is corrupt, factory defaults are reestablished. If further testing indicates that nonvolatile memory is defective, it will not be used, and **ValidNV** is *false*; otherwise, **ValidNV** is *true*. In many products, if nonvolatile memory is defective, it is emulated in RAM. The operating behavior is the same, except that persistent parameter values are lost when the printer system is powered off or restarted and factory defaults are used. **Legal values**: *true*, *false* **Errors**: None
WaitTimeout‡	integer	Indicates the value, in seconds, to which the user parameter **WaitTimeout** is initialized at the beginning of each job. A value of 0 indicates that the time-out is infinite. Any attempt to set the system parameter **WaitTimeout** to a negative value is ignored, and the previous setting of **WaitTimeout** is used. **Legal values:** Any positive integer or 0. **Errors:** **typecheck**

10.4 Administrator Jobs and Passwords

10.4.1 Unencapsulated Jobs

An unencapsulated job is entered by executing the LanguageLevel 2 operator **startjob** or the LanguageLevel 1 operator **exitserver**. These operators require a password to be presented. The password must be equal to the value of either the **StartJobPassword** or the **SystemParamsPassword** system parameter. If the password is equal to the value of **StartJobPassword**, an ordinary unencapsulated job is started. See Section 3.7.7 in the *PostScript Language Reference Manual, Second Edition*. If the password is equal to the value of

SystemParamsPassword, a system administrator job is started. (If the **SystemParamsPassword** is a zero-length string or has never been set, every unencapsulated job is a system administrator job.)

During a system administrator job, **setsystemparams** and **setdevparams** may be invoked without presenting a password each time. Also, LanguageLevel 1 compatibility operators that change system and device parameters may be executed; see Section 9.3.2, “Changing Persistent Values with Password,” on page 234.

10.4.2 Passwords for System and Device Parameters

The system parameters **StartJobPassword** and **SystemParamsPassword** are explained in Section C.3.1 in the *PostScript Language Reference Manual, Second Edition*. Section C.4 makes the statement, “**setdevparams** is very similar to **setsystemparams**; the same restrictions apply.” This statement needs to be clarified. When device parameters are set, most but not all will require a password equal to **SystemParamsPassword**. Also, there is one system parameter that does not require a password. The exceptions to the rules are as follows:

- The **FactoryDefaults** system parameter does not require a password if **FactoryDefaults** is the only entry in the dictionary passed to the **setsystemparams** operator. If the only other key in the dictionary is the password, it is ignored. This is necessary so that if the value of **SystemParamsPassword** has been forgotten, there will still be a way to reset it. See the description of the **FactoryDefaults** parameter in Table 10.2 on page 248.

- The device parameters **Interpreter** and **Protocol** in device parameter sets of type /Communications do not require a password if one or both are the only entries in the dictionary passed to the **setdevparams** operator. If the only additional key in the dictionary is the password, it is ignored. See **Interpreter** described in Table 10.3 on page 269 and **Protocol** described in Table 10.4 on page 274.

10.5 Device Parameters

Device parameters are set using the operator **setdevparams** and are read using the operator **currentdevparams**. Device parameters are similar to system parameters in that they require a password (if a value is set for **SystemParamsPassword**), are global to the PostScript environment, and persist across jobs. As with system parameters, some device parameters may be stored persistently in nonvolatile memory.

Device Parameter Interdependencies

Device parameters differ from both system and user parameters in that they are interdependent; that is, the legality of a value for a given parameter may depend on the value of another parameter. For example, the serial communications device parameter set contains an **Interpreter** parameter and a **Protocol** parameter. The **Interpreter** parameter determines which page description language is to be used for an incoming job on that channel. The **Protocol** parameter determines the communications protocol used to send and receive data. **Interpreter** can be set to /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, or /ProprinterXL. **Protocol** can be set to /Binary, /Normal, /Raw, or /TBCP. However, the serial channel cannot be configured to have **Protocol** set to /Raw and **Interpreter** set to /PostScript. This would be an illegal combination of device parameters. This condition is termed a *configuration error.* A PostScript error (**configurationerror**) occurs if the **setdevparams** operator attempts to establish an illegal configuration.

Most configuration dependencies are between parameters in the same device parameter set. However, there is a dependency among all communications devices that requires at least one of the communications channels to be **On** and **Enabled**.

There might also be cases where certain device parameter sets have interdependencies. For example, if both LocalTalk and a serial channel share the same hardware port on a printer system, both cannot be **On** at the same time. If one channel is already **On** and the other is turned **On**, the first is turned off and disabled.

Parameter Sets

Device parameters are grouped into sets, where the set of parameters describes the configuration of a physical or logical communications channel, storage device, hardware device, or software entity, such as a language emulator. For example, the %Serial% parameter set contains the parameters for the serial device. Each named parameter set known to the **currentdevparams** and **setdevparams** operators corresponds to an instance of the **IODevice** resource category. Even if two products have the same named device, their respective parameter sets might differ because the hardware support for that device may differ on the two products.

Device Parameter: Type

There are four general categories of device parameter sets. Each device parameter set must have a key-value pair that indicates its type. The key is **Type** and the value can be /Communications, /Emulator, /FileSystem, or /Parameters. The **Type** parameter is *read-only.*

- **Type** = /Communications. These parameter sets (described in Section 10.5.1, “Communications Parameter Sets (Type = /Communications),” on page 262) are associated with the application layer of the open systems interconnection (OSI) reference model shown in Figure 10.1 (see “Network Communications Parameter Sets” on page 283) or with point-to-point connections that need not be layered or need not conform to the OSI model (see “Point-to-Point Communications Parameter Sets” on page 274).

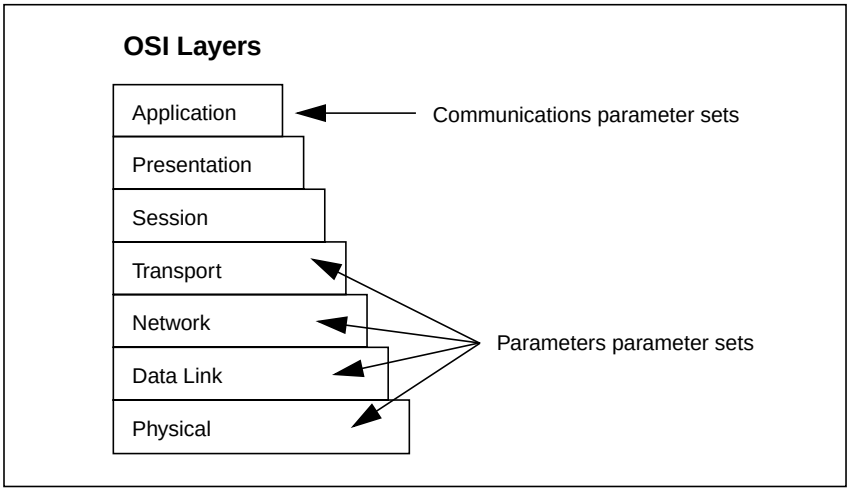
- **Type** = /Parameters. These parameter sets correspond to the transport, network, data link, and physical layers of the OSI reference model (shown in Figure 10.1), as well as to miscellaneous devices such as engine and calendar. See Section 10.5.2, “Parameters Parameter Sets (Type = /Parameters),” on page 303.

- **Type** = /FileSystem. These parameter sets describe storage devices. See Section 10.5.3, “File System Parameter Sets (Type = /FileSystem),” on page 328.

- **Type** = /Emulator. These parameter sets allow PostScript printer systems to emulate other printer systems. See Appendix B, “Emulators,” on page 347.

Note *Not all of the device parameters listed in Tables 3.4 to 3.35 will be present in every printer system product. Note also that error listings are confined to device-specific errors.*

Figure 10.1 *The OSI (Open Systems Interconnection) layers*



10.5.1 Communications Parameter Sets (Type = /Communications)

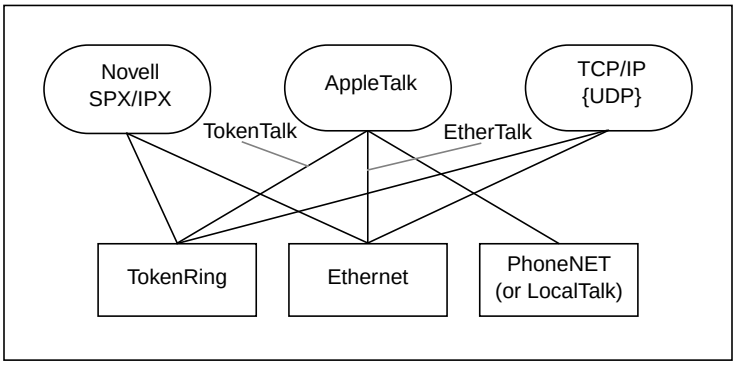
Adobe has defined various device parameter sets (or dictionaries of key-value pairs) to aid system administrators in setting up and maintaining network printer systems. These parameter sets are based on the layers within the OSI model (shown in Figure 10.1), even though many of the protocols predate the OSI reference model. The PostScript interpreter or emulator receives its jobs from the application layer of OSI, and only the application layer of OSI has parameter sets of **Type** /Communications. The device parameter sets that correspond to the transport, network, data link, and physical layers of the OSI model are of **Type** /Parameters. See Section 10.5.2, “Parameters Parameter Sets (Type = /Parameters),” on page 303.

Note *The /Communications parameter sets are described in the following sections: “Entries Found in All Parameter Sets of Type /Communications” on page 269, “Point-to-Point Communications Parameter Sets” on page 274, and “Network Communications Parameter Sets” on page 283.*

A raster output device can have various physical communications channels and can use many different protocols over these channels. For point-to-point connections, the choices of physical hardware include serial, unidirectional and/or bidirectional parallel, and SCSI bus. For network communications hardware, the choices include Ethernet, TokenRing, and LocalTalk®. Network adaptors, such as line multiplexers, parallel-to-Ethernet adaptors, and SCSI-to-Ethernet adaptors, allow raster output devices to connect to networks to which they do not have a physical connection. For point-to-point communications, there are no underlying device parameters specified by /Parameter parameter device sets; that is, only the /Communications device parameter set is required. The network parameter sets described in this section, however, apply only to raster output devices that are true peers of the network and have a direct connection to it. These parameter sets support the popular protocol stacks, such as TCP/IP, AppleTalk®, and Novell® NetWare® (SPX®/IPX®). See “Point-to-Point Communications Parameter Sets” on page 274 and “Network Communications Parameter Sets” on page 283 for a description of the related parameter sets.

In order to reside directly on networks, raster output devices must recognize various standard communications protocols. The choices of protocols and the physical hardware media on which they reside are limited, as shown in Figure 10.2. The connectors in this diagram indicate that the network protocols can be used with the physical medium indicated to deliver and receive messages.

Figure 10.2 *Relationship between network communications protocols and physical communications media*



- *Novell SPX/IPX*. The Novel NetWare communications protocols were originally derived from the XNS (Xerox Network System) protocols. For example, the Novell IPX (Internetwork Packet Exchange) is virtually identical to the Xerox Network Layer Protocol called IDP (Internetwork datagram protocol). The Novell transport protocol is called SPX (Sequenced Packet Exchange).

  For a discussion of Novell networks, see Malamud, Carl: *Analyzing Novell® Networks*; Van Nostrand Reinhold: New York, 1992, page 81.

- *AppleTalk*. AppleTalk is the name Apple Computer, Inc. uses for its networking software that is built into every Macintosh computer. The data link protocol LocalTalk Link Access Protocol (LLAP) is used for communicating over LocalTalk or PhoneNET networks. EtherTalk Link Access Protocol (ELAP) and TokenTalk Link Access Protocol (TLAP) are the data link protocols for communication over Ethernet and TokenRing, respectively. AppleTalk over Ethernet is called EtherTalk, and AppleTalk over TokenRing is called TokenTalk.

  See *Computer Networks* for a description of TokenRing and Ethernet (Tanenbaum, Andrew S., *Computer Networks*; Prentice-Hall, Inc.: New Jersey, 1981). See *Inside AppleTalk* for a description of AppleTalk (Sidhu, Andrews, and Oppenheimer, *Inside AppleTalk®, Second Edition*; Addison-Wesley Publishing Company, Inc.: Menlo Park, California, 1990).

- *TCP/IP*. TCP/IP is a network architecture sponsored by DARPA and has been adopted by a large number of vendors. TCP (Transmission Control Protocol) is a transport layer protocol. IP (Internet Protocol) is the network layer protocol used by TCP. UDP (User Datagram Protocol) is a connectionless protocol. It is also dependent on IP for routing to the destination but does not ensure that the destination receives the packets.

Setting Up Communications Parameter Sets

The communications parameters for a product can be set up in different ways and may involve the use of front panels, hardware switches, and PostScript operators and procedures. This section describes a generic model for setting communications parameters that works across a variety of products and enables PostScript spoolers and utilities to use the same model when reading and writing communications device parameters.

A raster output device typically has several hardware ports for communications. For example, a printer system might have a parallel port and two serial ports named channel A and channel B. The parallel port is associated with the parameter set named %Parallel%. Serial channel A, which is wired to a 25-pin RS-232A connector, is associated with the parameter set named %Serial%. Serial channel B, which is wired to either an 8- or a 9-pin connector, is associated with the parameter set named %SerialB% or with the parameter set named %LocalTalk%. In this example, two parameter sets are associated with the same hardware port.

Note *When there are multiple instances of a communications parameter set, the naming convention is %CommName%, %CommNameB%, %CommNameC%, and so on.*

%CommName_NV%, %CommName%, and %CommName_Pending%

For any given communications device, there are three parameter sets. For example, if the name of the device is %CommName%, the three associated parameter sets are %CommName_NV%, %CommName%, and %CommName_Pending%. These three parameter sets have the following general characteristics:

- %CommName_NV% values usually are stored in nonvolatile memory.

  In products with limited nonvolatile memory, only some of the %CommName_NV% values may actually be saved to nonvolatile memory; in products with sufficient nonvolatile memory, typically all writeable %CommName_NV% values are saved to nonvolatile memory. PostScript utility programs need be concerned with these differences. If the intent is to affect persistent values, use %CommName_NV%. The implementation will do the best it can given the amount of nonvolatile memory available in the product.

- %CommName% values usually are stored in RAM and do not persist when the printer system is powered off.

- %CommName_Pending% is a *read-only* parameter set whose values are used to configure the communications hardware and software at the beginning of the next file. This parameter set reflects either the current

values of some writeable parameter set, such as %CommName%, or some predetermined values selected via a switch or front panel. How the system computes the values in %CommName_Pending% is described below.

In our printer system example above, the following parameter sets might be available:

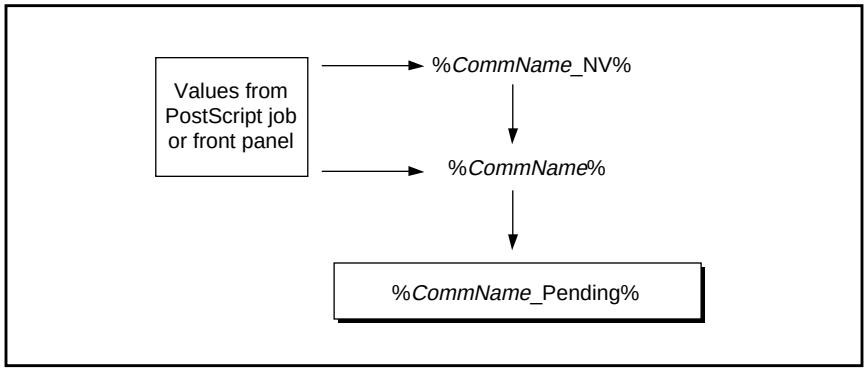
%Parallel_NV%	%Parallel%	%Parallel_Pending%
%Serial_NV%	%Serial%	%Serial_Pending%
%SerialB_NV%	%SerialB%	%SerialB_Pending%
%LocalTalk_NV%	%LocalTalk%	%LocalTalk_Pending%

Hierarchy of %CommName_NV%, %CommName%, and %CommName_Pending%

All products have the three parameter sets %CommName_NV%, %CommName%, and %CommName_Pending% to provide a consistent model that is product independent. This section describes the hierarchical relationship among the parameter sets, starting with a simple subset of the model and progressing to more complex situations.

Note *On some products, the parameter sets %CommName_NV%, %CommName%, and %CommName_Pending% may not be distinct from one other.*

Figure 10.3 *Relationship among the communication parameter sets*



In Figure 10.3, values written to %CommName% are written through to %CommName_Pending%, and values written to %CommName_NV% are written through to %CommName% and then to %CommName_Pending%. Beyond this, there are several variables:

- The product may have a front panel. The values set by the user at the front panel are written to %CommName% or to %CommName_NV% (if the values are to persist across restarts and power cycles). Some products store to only one of these sets.

- The product may have switches through which it can be directed to use either %CommName_NV% parameter sets or built-in (hard-wired) values.

  Generally, products do not have both a front panel and switches.

- PostScript programs (usually spoolers or utilities) may write parameter values to %CommName% or %CommName_NV% (usually the former) at any time. This is true whether the output device has a front panel or switches.

In Figure 10.3, the %CommName% parameter set, which is in RAM and does not persist when the printer system is powered off, is used in many cases (but not all) to update the %CommName_Pending% set. Thus, on many products, such as those with a front panel but no switches, the %CommName% and %CommName_Pending% sets always have the same values and appear redundant.

The %CommName_NV% set usually stores the parameters in nonvolatile storage. In the simple case in Figure 10.3, writing to %CommName_NV% writes through to %CommName%, which in turn writes through to %CommName_Pending%.

In general, a spooler or utility almost always should write to %CommName%. It should write to %CommName_NV% only if parameters are to persist when the printer system is powered off.

A front panel usually writes to %CommName_NV% to change the power-on parameters, although the front panel also can write to %CommName%.

Multiple Nonvolatile Parameter Sets (%CommName_NV*n*%)

Complicating this picture, it is possible to have more than one nonvolatile parameter set. Such sets are named %CommName_NV%, %CommName_NV2%, %CommName_NV3%, and so on. As with a single nonvolatile set, these parameter sets obtain their values by being written to by a PostScript spooler or utility.

Figure 10.4 *Communications parameter sets using NV values*

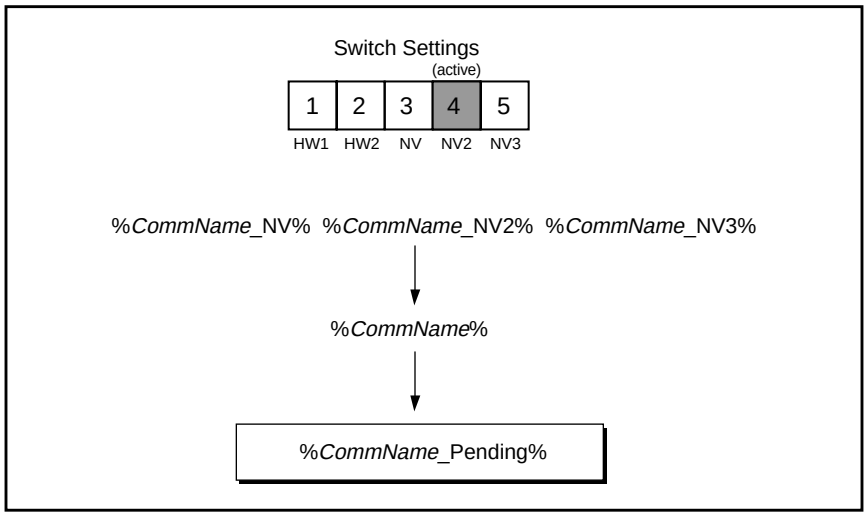
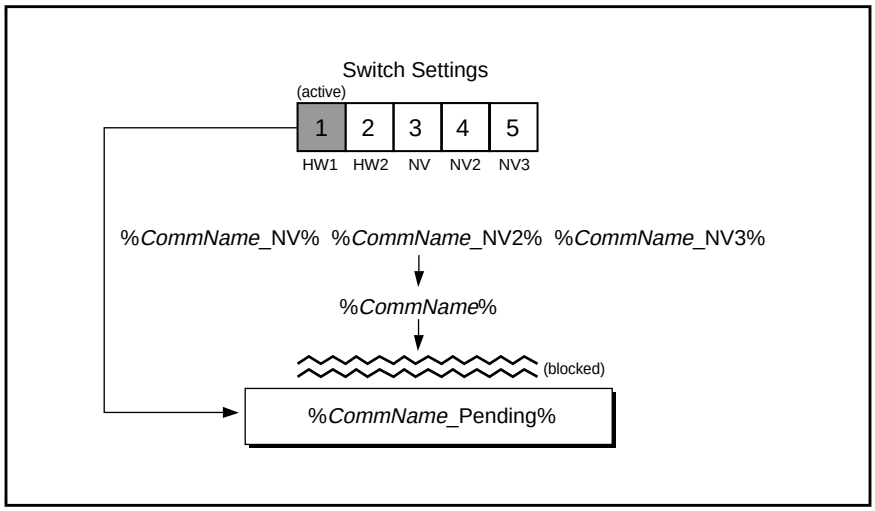


Figure 10.4 shows a situation in which there are three nonvolatile sets. Only one of these sets can be active at any given time. In this figure, the active set is %CommName_NV2%, as indicated by the switch setting. When the switch is set to this position, or when the product is restarted or powered on with the switch in this position, the values in %CommName_NV2% are written through to %CommName% and to %CommName_Pending%. While the setting %CommName_NV2% is active, a PostScript job can write to any of the nonvolatile parameter sets, but only values written to %CommName_NV2% would migrate to %CommName% and %CommName_Pending%. Changing the switch to the position corresponding to %CommName_NV3% would cause %CommName_NV3% values to become the active values in %CommName% and %CommName_Pending%.

Predetermined (Hard-Wired) Parameter Values

In addition to the switch settings that indicate which nonvolatile parameter set should be used, other switch settings can affect this hierarchy of parameter sets and cause a predetermined set of communications parameters to be written directly to %CommName_Pending%. This situation is shown in Figure 10.5.

Figure 10.5 *Communications parameter sets using "hard-wired" values*



In Figure 10.5, switch positions 1 and 2 designate two "hard-wired" parameter sets. When the switch is set to position 1, for example, PostScript programs may still write to one of the %CommName_NV% sets or to %CommName%, but not to %CommName_Pending% unless the switch is reset to one of positions 3 through 5.

This example explains the existence of the %CommName_Pending% set as separate from the %CommName% set: that is, it allows absolute determination of the communications parameters that will be used, no matter what other activity occurs.

Note *Reading the %CommName_NV% set or the %CommName% set gives no information about the parameters being used for the current job or the next job, but simply returns the values last written to these sets. Reading %CommName_Pending% returns the values to be used for the next job. Determining the parameters of the current job is of little interest. Either the job is a page description, in which case it should not be accessing device parameters at all, or the job is a utility that is interested in either determining or affecting the settings for future jobs. If the device parameters are used as described above, utilities can be written without concern for exactly which parameters are stored in nonvolatile memory and without concern for whether a utility job, front panel, or switch is used to establish communications parameters.*

As described in the previous section, a spooler or utility almost always should write to %CommName%. It should write to %CommName_NV% only if parameters are to persist across restarts and power cycles.

Changes to parameters sets of **Type** /Communications take effect after the current file (containing one or more PostScript jobs) is fully processed by the interpreter and before reading from the next file.

A file that specifies communications changes will complete before a transition to new settings of the %CommName_Pending% set occurs. The user must ensure that no additional data will be transferred from host to printer system on this communications channel until after the transition to new settings is complete.

Note *There is a distinction between "end of job" and "end of file." A file can contain multiple jobs. For further information, see Section 3.7.7 in the PostScript Language Reference Manual, Second Edition.*

Entries Found in All Parameter Sets of Type /Communications

Table 10.3 describes the entries found in all device parameter sets of **Type** /Communications.

Table 10.3 *Entries in all communications parameter sets (Type = /Communications)*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	Selects how the output channel is managed after each job finishes executing. The printer system does not wait for the pages of the job to finish printing but immediately starts executing the next job. The **DelayedOutputClose** parameter is set independently for each communications channel. When the value of **DelayedOutputClose** is *true*: • An EOF is not sent until all pages of a job have been printed. On network channels, the connection remains open until the job finishes printing. • If a job produces output and if there are preceding jobs that have not finished printing that are using the same output channel, the output will not be sent until those jobs have completed printing and the EOFs for them have been sent. • Spontaneous messages, such as printer system error messages, are sent to the channel if it is either the output channel for the job executing or the output channel for jobs that have finished executing but have not finished printing. When the value of **DelayedOutputClose** is *false*:

Table 10.3 *Entries in all communications parameter sets (Type = /Communications) (continued)*

Key	*Type*	*Semantics*
		• An EOF is sent as soon as a job finishes executing in the interpreter. On network channels, the connection may be closed when the job finishes executing, even though pages produced by the job might not have finished printing. • Output generated by a job can be transmitted without delay, even if previous jobs are using the same output channel and have not finished printing (the EOF for those jobs will have already been sent). • Spontaneous messages, such as printer system error messages, are sent to the channel only if it is the output channel for the job executing, even if it is the output channel for previous jobs that have not finished printing. The **DelayedOutputClose** parameter setting for a job source is controlled by the parameter sets for the output channel of that job source. Thus, if the serial B channel is used for the output of jobs received on the parallel port, then the **DelayedOutputClose** value in the %SerialB% parameter set applies to jobs received on both the serial B and parallel ports. The **DelayedOutputClose** parameter does not appear in a communications parameter set if the channel has no output or if all messages generated asynchronously from the interpreter are directed to a logically separate channel. *Note* *In versions prior to 2014, upon completion of each job, the interpreter waits for all pages of the job to be printed before sending an EOF or closing a connection and before starting to execute another job. Thus, any output associated with the job, including printer system error messages, is always sent before the next job begins and before the EOF is sent or a connection closed.* **Legal values**: *true, false* **Errors**: None
Enabled	boolean	Designates whether data arriving on the communications channel represented by the parameter set should be considered as a job to be scheduled for execution by the PostScript interpreter or an emulator. If the value of **Enabled** is *true*, arriving data is scheduled as an executable job. If the value of **Enabled** is *false*, the data will not be scheduled as an executable job, but the channel can be used directly by a job for reading and writing data. A **configurationerror** is generated if setting **Enabled** would produce either of the following situations: • Trying to set **On** to *false* and **Enabled** to *true* within the same parameter set • When trying to turn off **Enabled** in one communications device parameter set results in all channels having **Enabled** set to *false*

Table 10.3 *Entries in all communications parameter sets (Type = /Communications) (continued)*

Key	*Type*	*Semantics*
		Legal values: *true, false*
		Errors: **configurationerror, typecheck**
HasNames	boolean	*(Read-only)* Indicates whether the communications channel represented by the parameter set supports named files. **HasNames** is always *false* in device parameter sets of **Type** /Communications.
		Legal values: *false*
		Errors: None
Interpreter	name	Designates which interpreter or emulator is to be used to interpret the next incoming job arriving on this communications channel. This parameter is used only if the value of **Enabled** is *true* and the value of **PrinterControl** is /PSPrinter. For certain communications channels there is a relationship between the **Interpreter** and the **Protocol** parameters that can result in a **configurationerror**. See the **Protocol** entry in Table 10.4 for further details. Either **Interpreter** or **Protocol** or both can be set without a password if no other parameters are specified in the execution of the **setdevparams** operator. The **Interpreter** value /AutoSelect is described below. For information on the other legal values, see Appendix B, “Emulators,” on page 347. /AutoSelect: Provides automatic and seamless switching between the available interpreters and emulators based on the input data stream. The value of **Interpreter** should be set to /AutoSelect on channels that connect to hosts that alternately send PostScript jobs, raw PCL® (LaserJet IIP or LaserJet III), and “printscreen” jobs (in the IBM® PC-compatible environment). It can be used on any communications channel. When used for a given communications channel, it is important that the underlying communications protocol is one that preserves all incoming data. In particular, for a serial or parallel channel, this implies that **Protocol** is set to /Raw, /Binary, or /TBCP. For serial and parallel communications channels, the following is true: • /AutoSelect detects interpreter boundaries and job boundaries if the value of **Protocol** is set to /TBCP or /Binary. • /AutoSelect detects interpreter boundaries, job boundaries, and protocol boundaries and automatically selects the protocol if the value of **Protocol** is set to /Raw. This is the recommended setting for **Protocol** when using /AutoSelect. When /AutoSelect detects that a PostScript job is being received and **Protocol** is /Raw, only Normal and TBCP protocols can be recognized (e.g., Binary is not supported).

Table 10.3 *Entries in all communications parameter sets (Type = /Communications) (continued)*

Key	*Type*	*Semantics*
		• When **Interpreter** is set to /AutoSelect, the value of **Protocol** must be /Binary, /Raw, or /TBCP; otherwise, a **configurationerror** is generated. For other communications channels that are binary in nature, the following is true: • /PCL is used only in printer systems that emulate a LaserJet 4 or later LaserJet printers. For emulations of earlier LaserJets, the values /LaserJetII and /LaserJetIII are used. • /AutoSelect detects interpreter boundaries and job boundaries. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	Designates whether the communications channel is turned on and able to receive and send data. If the parameter is *true*, data transmitted to the channel by a host is buffered and flow control protocols are applied. Data sent to the channel when this parameter is *false* is lost. A **configurationerror** is generated if setting the **On** parameter would produce a situation in which **On** is *false* and **Enabled** is *true* in the same parameter set. If two communications devices share the same physical port and setting the **On** parameter produces a situation in which both channels had **On** set to *true*, the one that was originally **On** is turned off and disabled, and the new one is turned **On**. If **On** is *true* and **Enabled** is *false*, the channel is not considered as a source of jobs to be scheduled, but the channel can be used by a PostScript job to send and receive data by means of the file operators. During power up, if it is determined that all installed communications channels are currently off, it is up to the product to perform its own unique recovery strategy. For example, the product could search for an installed communications channel and force it on, even if this was not the state preserved in nonvolatile memory. An alternative would be to inform the user via the operator control panel that the product cannot be initialized until the problem is rectified. **Legal values**: *true, false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	Used to select or indicate how a host queries and controls the printer system for the communications channel associated with this parameter set. If the **PrinterControl** parameter is set to /PSPrinter, the following statements are true:

Table 10.3 *Entries in all communications parameter sets (Type = /Communications) (continued)*

Key	*Type*	*Semantics*
		• The **Interpreter** parameter selects the page description language. • Printer system error messages are sent in the usual Adobe fashion (on channels processing jobs, and in `%%[ . . . ]%%` format). • PJL commands are not recognized unless the **Interpreter** parameter is set to /AutoSelect or /PCL. If the **Interpreter** parameter is /AutoSelect, `ENTER LANGUAGE` and `UEL` are handled and other PJL commands are identified as PJL commands and discarded. If **Interpreter** is set to /PCL, `UEL` is handled. If the **PrinterControl** parameter is set to /PJL, the following statements are true: • PJL controls language selection. • Printer system error messages are in PJL format and are enabled or disabled by PJL commands. Whether a job is being processed on a channel does not affect whether messages are sent. • The "PJL current environment" is used on each invocation of a page description language to set up the initial state. The PJL language is described in the Hewlett-Packard *Printer Job Language Reference Manual* (Manual Part No. 5961-0998). The edition of this manual published in May, 1993 includes the PJL support on LaserJet 4si (4si MX) printers. Earlier editions describe the PJL on LaserJet 4 (and 4M) printers, but not the LaserJet 4si printers. For example, when **PrinterControl** is set to /PJL, the behavior is that of LaserJet 4 communications. When **PrinterControl** is set to /PSPrinter, it supports PDLs of LaserJet4, but not all identical communications behaviors. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror**, **rangecheck**, **typecheck**

Point-to-Point Communications Parameter Sets

Additional required entries for the serial, parallel, and SCSI device parameter sets are summarized below. Details of the required entries are given in the referenced tables.

%Serial%
(Table 10.3 and Table 10.4 on page 274)

Baud	**FlowControl**	**PrinterControl**
CheckParity	**HasNames**	**Protocol**
DataBits	**Interpreter**	**StopBits**
DelayedOutputClose	**On**	**Type**
Enabled	**Parity**	

%Parallel%
(Table 10.3 and Table 10.5 on page 279)

DelayedOutputClose	**Interpreter**	**OutputDevice**
Enabled	**nAckPulseWidth**	**PrinterControl**
Handshake	**nStrobeExpectedPulseWidth**	**Protocol**
HasNames	**On**	**Type**

%ScsiComm%
(Table 10.3 and Table 10.6 on page 281)

Bus	**Filtering**	**On**
DelayedOutputClose	**HasNames**	**PrinterControl**
Enabled	**Interpreter**	**Type**

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets*

Key	*Type*	*Semantics*
Baud	integer	Designates the baud rate on the underlying serial hardware. Normally this value can be set to any non-negative number; it will not be rounded. The underlying serial hardware will, however, round the baud rate to the nearest achievable value. Hardware rounding will not be reflected in the value of the parameter when it is read. On some products this parameter might be restricted to a small number of legal values. **Legal values**: Product dependent. **Errors**: **rangecheck**, **typecheck**
CheckParity	boolean	Designates whether parity checking is done by the device on incoming data. This parameter is ignored if the value of **Parity** is /None. If the value of **CheckParity** is *true* and a parity error occurs, a PostScript **ioerror** results. If the value of **CheckParity** is *false*, no parity checking occurs.

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: *true, false*
		Errors: **typecheck**
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true, false*
		Errors: None
DataBits	integer	Designates the number of data bits per byte communicated over the channel. If the value of this parameter is 7, the high bit of a received byte of data is set to 0. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits, and stop bits.
		Legal values: 7, 8
		Errors: **rangecheck**, **typecheck**
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true, false*
		Errors: **configurationerror**, **typecheck**
FlowControl	name	Designates the serial flow control method used between the host and the device.
	Note	*Not all serial channels support all flow control modes.*
		The legal values of **FlowControl** are as follows:
		/Dtr: DTR (data terminal ready) and DSR (data set ready) hardware signals are used by the printing device and the host, respectively, to indicate to the other when data may be transmitted. A high value for the signal indicates that data may be transmitted; a low value indicates that data should not be transmitted.
		/DtrLow: As for /Dtr, except the active sense of the signals is reversed. A low signal indicates that data may be transmitted, a high signal indicates that data should not be transmitted.
		/EtxAck: Two characters, ETX (end-of-text) and ACK (acknowledgment), are reserved for flow control usage. The protocol is symmetrical for printing device and host. Each sender knows an agreed upon maximum number of characters that the other side can receive. A sender may send up to this number of characters followed by an ETX. The sender may send more data only when it has received an ACK from the receiver on the other side.

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		/RobustXonXoff: Operates similarly to the /XonXoff protocol, except that periodically (typically every second) the interpreter will send the host an Xon if it is able to receive data. /XonXoff: Used by PCs. Two characters, XON and XOFF, are reserved for flow control usage. For all **Protocol** parameter settings except /Raw, the protocol is symmetric for printing device and host. If one side wishes the other to stop sending data, it sends an /XOFF. When it is ready to receive data again, it sends an /XON. When the **On** parameter is set to *false*, the interpreter sends an /XOFF to the host just before turning off the channel. If **Protocol** is set to /Raw, /XON and /XOFF sent from the host to the printer system are treated as data and not reserved as flow control characters. /XON and /XOFF sent from the printer system to the host are to be treated as flow control characters. /XonXoff2: Used by UNIX systems. This protocol operates similarly to the /XonXoff protocol, except that when the **On** parameter is set to *false*, the interpreter sends an /XON to the host. This allows the host to unload any data it wishes to send. The interpreter simply drops the data. **Legal values**: /Dtr, /DtrLow, /EtxAck, /RobustXonXoff, /XonXoff, /XonXoff2 **Errors**: **rangecheck**, **typecheck**
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values**: *false* **Errors**: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL Errors: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Parity	name	Designates the parity to be used between the host and the device. If **Parity** is /Space or /Mark, the parity bit should always be 0 or 1, respectively. If **Parity** is /None, neither the host nor the device should send a parity bit. If **Parity** is /Even, even parity is used. If **Parity** is /Odd, odd parity is used. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits, and stop bits. Most serial devices do not support 8-bit data with either space or mark parity, although setting the parameters in this manner does not generate a **configurationerror**. The results of this configuration, however, are unpredictable. **Legal values**: /Even, /Mark, /None, /Odd, /Space **Errors**: **rangecheck**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
Protocol	name	Indicates the communications protocol that is used. /Binary: In this mode, an encoding scheme allows the full range of 8-bit values to be transmitted as data while also providing for certain communications functions, such as end-of-file, software flow control, abort job, status query, and so on. This protocol is suitable for use with any language (for example, the PostScript language or a printer emulation). However, it is obsolete and has been superseded by /TBCP. /Normal: In this mode, certain control characters, such as end-of-file, software flow control, abort job, and status query, are reserved as communications functions. These codes cannot be carried as data. This protocol is suitable for use only when sending ASCII-encoded PostScript jobs; it is unsuitable for PostScript jobs containing binary data or any printer emulation jobs. /Raw: In this mode, all characters are treated as data; there are no reserved characters, and one of the communications functions is available. Normally, this protocol is suitable for use only with printer emulation, not with the PostScript interpreter. However, in products that support an **Interpreter** value of /AutoSelect, protocol processing is handled by the AutoSelect facility; therefore, the value of **Protocol** should be /Raw when **Interpreter** is set to /AutoSelect.

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		/TBCP: In this mode, an encoding scheme allows the full range of 8-bit values to be transmitted as data, while also providing for certain communications functions, such as end-of-file, software flow control, abort job, status query, and so on. It also provides explicit begin-protocol and end-protocol sequences that permit the receiver to switch automatically between /Normal and /TBCP mode processing. This protocol is suitable for use with any language (for example, the PostScript interpreter or a printer emulation). A **configurationerror** is generated if setting either the **Protocol** or the **PrinterControl** parameter would result in the following combination: • **Protocol** with a value of /Normal and **PrinterControl** with a value of /PJL A **configurationerror** is also generated if setting **Protocol** or **Interpreter** would produce one of the following situations when **Enabled** is *true* and **PrinterControl** is /PSPrinter • **Protocol** with a value of /Raw and **Interpreter** with a value of /PostScript • **Protocol** with a value of /Normal and **Interpreter** with a value other than /PostScript • **Protocol** with a value of /Normal and **Interpreter** with a value of /AutoSelect That is, PostScript jobs cannot be executed over a channel using the /Raw protocol, and emulators cannot be executed over a channel using the /Normal protocol. Likewise, when automatic selection of interpreters and emulators is chosen, the /Normal protocol cannot be used. Either the **Protocol** or **Interpreter** parameter or both can be set without a password if no other parameters are specified in the execution of the **setdevparams** operator. **Legal values**: /Binary, /Normal, /Raw, /TBCP **Errors**: **configurationerror**, **rangecheck**, **typecheck**
StopBits	integer	Designates the number of stop bits that are transmitted by the serial hardware. The hardware will always be able to receive data transmitted with one or two stop bits. The total number of bits for each byte transmitted or received is the sum of the number of start bits (always 1), data bits, parity bits, and stop bits. **Legal values**: 1, 2 **Errors**: **rangecheck**, **typecheck**

Table 10.4 *Entries in the %Serial%, %SerialB%, %SerialC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Table 10.5 *Entries in the %Parallel%, %ParallelB%, %ParallelC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269. **Legal values**: *true, false* **Errors**: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269. **Legal values**: *true, false* **Errors**: **configurationerror**, **typecheck**
Handshake	integer	Indicates the hardware/software signal interface that is to be used for communications across the parallel interface. If this key is not present, the default is unidirectional parallel. 0 Unidirectional communications commonly used by PCs and PC-compatibles 1 Bidirectional communications, as specified by version 0.6 of the Hewlett-Packard Boise specification 2 IEEE-1284 Draft Specification 1.00 3 Unidirectional, Ack in Busy 4 Unidirectional, Ack after Busy (Japanese implementation) 5 Unidirectional, Ack while Busy 6 IEEE-1284 Draft Specification 2.00, Ack in Busy 7 IEEE-1284 Draft Specification 2.00, Ack after Busy 8 IEEE-1284 Draft Specification 2.00, Ack while Busy Values 1 and 2 are obsolete. Value 1 was superseded by value 2, which was superseded by values 6, 7, and 8.

Table 10.5 *Entries in the %Parallel%, %ParallelB%, %ParallelC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Setting the **OutputDevice** parameter to %Parallel% will generate a **configurationerror** when the **Handshake** parameter is set to one of the unidirectional values (0, 3, 4, 5). Conversely, if the **OutputDevice** parameter is set to %Parallel%, then setting the **Handshake** parameter to one of the unidirectional values will generate a **configurationerror**.
	Note	*"The IEEE-1284 Draft Specification 2.00" refers to the document "Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers," IEEE P1284 D2.00, September 10, 1993.*
		Legal values: 0–8
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
nAckPulseWidth	integer	*nAck* is a signal that originates from the peripheral to the host. This signal is in the form of a pulse and its meaning depends on which mode of operation is being used or which state the peripheral device driver is currently in. The pulse width is controlled by the peripheral support circuitry or device driver. This parameter allows the *nAck* pulse width to be examined or changed. The value is the number of nanoseconds for the duration of the pulse (rounded to nearest value that can be achieved).
		Legal values: Normally an integer in the range 500–10000.
		Errors: **rangecheck**, **typecheck**
nStrobeExpectedPulseWidth	integer	*nStrobe* is a signal that originates from the host to the peripheral. This signal is in the form of a pulse, and its meaning depends on which mode of operation is being used or in which state the peripheral device driver is currently. The pulse width may vary from host to host. This parameter allows the expected duration of the *nStrobe* pulse width to be examined or changed. The value is the number of nanoseconds for the duration of the pulse (rounded to the nearest value that can be achieved).
		Legal values: Normally an integer in the range 750–500000.
		Errors: **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.

Table 10.5 *Entries in the %Parallel%, %ParallelB%, %ParallelC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
OutputDevice	string	Specifies which communications device is to be used for %stdout and %stderr. If the value of **OutputDevice** is the empty string, %stdout and %stderr information is routed to the default back channel specified for the device. Setting the **OutputDevice** parameter to %Parallel% will generate a **configurationerror** when the **Handshake** parameter is set to one of the unidirectional values (0, 3, 4, 5). Conversely, if the **OutputDevice** parameter is set to %Parallel%, then setting the **Handshake** parameter to one of the unidirectional values will generate a **configurationerror**. **Legal values**: %Serial%, %SerialB%, %SerialC%, and so forth; %Parallel%, %ParallelB%, %ParallelC%, and so forth; or the empty string. **Errors**: **rangecheck**, **configurationerror**
PrinterControl	name	For the definition of **PrinterControl**, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**
Protocol	name	For the definition of **Protocol**, see Table 10.4 on page 274. **Legal values**: /Binary, /Normal, /Raw, /TBCP **Errors**: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Table 10.6 *Entries in the %ScsiComm%, %ScsiCommB%, %ScsiCommC%, … parameter sets*

Key	*Type*	*Semantics*
Bus	string	(*Read-only*) Designates which SCSI bus device parameter set is associated with this %ScsiComm% channel. **Legal values**: %Scsi%, %ScsiB%, %ScsiC%, and so forth. **Errors**: None

Table 10.6 *Entries in the %ScsiComm%, %ScsiCommB%, %ScsiCommC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
DelayedOutputClose		
	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Filtering	name	Indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language.
		Legal values: /InterpreterBased, /None
		Errors: **configurationerror**, **rangecheck**, **typecheck**
		/InterpreterBased:
		Filters the input stream as necessary to conform to the language. For example, the data stream may have been sent to the printer system encoded as a tagged binary communication protocol (TBCP) PostScript job and must be decoded to a normal PostScript job before it is passed to the interpreter. See **Protocol** in Table 10.4 on page 274 for a description of /TBCP.
		/None: Passes the data unchanged to the interpreter.
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**

Table 10.6 *Entries in the %ScsiComm%, %ScsiCommB%, %ScsiCommC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Network Communications Parameter Sets

This section describes the device parameter sets that correspond to the OSI application layer protocol. This is the layer to which the PostScript interpreter (or language emulator) attaches for the purpose of receiving jobs and sending data back to the host. These are all of type /Communications. The defined parameter sets %LPR%, %AppSocket%, and %Telnet% allow for the use of the TCP/IP protocol over Ethernet; the parameter sets %RemotePrinter% and %PrintServer% are associated with the Novell NetWare application layer. This section also describes the device parameter sets for the AppleTalk networking software (EtherTalk, TokenTalk, and LocalTalk), which do not split out the physical layer.

- %LPR%. The UNIX command `lpr` sends a printer job to a printing system. TCP port 515 is used for LPR. Because LPR (or `lpr` daemon) is by definition unidirectional, any %stdout or %stderr information is transmitted by means of the Syslog facility described in Table 10.18 on page 310. The LPR service depends on the TCP/IP protocol. See Table 10.7 on page 286.

- %AppSocket%. AppSocket was created to support TranScript™ software from Adobe Systems, Inc. It provides a more robust interface than LPR because it utilizes bidirectional communications directly. The AppSocket protocol can be used by drivers other than TranScript and for transmitting data other than PostScript jobs. See Table 10.8 on page 288.

- %Telnet%. Telnet gives network users interactive access to and exclusive use of the PostScript interpreter (or emulator). Port 23 is used for Telnet, which is a TCP/IP network service. See Table 10.9 on page 291.

- %RemotePrinter%. The Novell remote printer application is managed by a Novell print server and takes print jobs downloaded from a print server. A remote printer system may be shared by many print servers. See Table 10.10 on page 292.

- %PrintServer%. The Novell print server application communicates with Novell file servers to download print jobs from the print queues. A print server may communicate with multiple file servers and access multiple print queues. See Table 10.11 on page 293.

- %LAT%. LAT protocol is used primarily for communication between hosts and terminal servers. Often, these terminal servers have a printing device attached, making it possible to print from a given set of hosts to a particular set of printer systems on a local area network. See Table 10.12 on page 295.

- %LocalTalk%. LocalTalk is the name of AppleTalk running over LocalTalk or PhoneNET networks. See Table 10.13 on page 297.

- %EtherTalk%. EtherTalk is the name of AppleTalk running over Ethernet networks. See Table 10.14 on page 299.

- %TokenTalk%. TokenTalk is the name of AppleTalk running over TokenRing networks. See Table 10.15 on page 301.

%LPR%

(Table 10.3 and Table 10.7 on page 286)

Enabled	**On**	**SendWindowSize**
Filtering	**PrinterControl**	**Type**
HasNames	**PrintHost**	
Interpreter	**ReceiveWindowSize**	

%AppSocket%

(Table 10.3 and Table 10.8 on page 288)

ControlPortNumber	**HasNames**	**ReceiveWindowSize**
DataPortNumber	**Interpreter**	**SendWindowSize**
DelayedOutputClose	**On**	**StatusPortNumber**
Enabled	**PrinterControl**	**Type**
Filtering	**PrintHost**	

%Telnet%

(Table 10.3 and Table 10.9 on page 291)

DelayedOutputClose	**Interpreter**	**Type**
Enabled	**On**	
HasNames	**PrinterControl**	

%RemotePrinter%

(Table 10.3 and Table 10.10 on page 292)

DelayedOutputClose	**HasNames**	**PrinterControl**
Enabled	**Interpreter**	**Type**
Filtering	**On**	

%PrintServer%

(Table 10.3 and Table 10.11 on page 293)

DelayedOutputClose	**Interpreter**	**PrinterControl**
Enabled	**LoginPassword**	**Type**
Filtering	**On**	
HasNames	**PreferredServer**	

%LAT%

(Table 10.3 and Table 10.12 on page 295)

DelayedOutputClose	**Interpreter**	**PrinterControl**
Enabled	**KeepaliveTimer**	**RetransmitTimer**
Filtering	**MulticastTimer**	**RetransmitLimit**
Groups	**On**	**Type**
HasNames	**Physical**	

%LocalTalk%

(Table 10.3 and Table 10.13 on page 297)

DelayedOutputClose	**Interpreter**	**PrinterControl**
Enabled	**LocalTalkType**	**Type**
Filtering	**NodeID**	
HasNames	**On**	

%EtherTalk%

(Table 10.3 and Table 10.14 on page 299)

DelayedOutputClose	**EtherTalkZone**	**NodeID**
Enabled	**Filtering**	**On**
EthernetAddress	**HasNames**	**PrinterControl**
EtherTalkType	**Interpreter**	**Type**

%TokenTalk%

(Table 10.3 and Table 10.15 on page 301)

Address	**HasNames**	**TokenTalkType**
Bridging	**Interpreter**	**Type**
DelayedOutputClose	**NodeID**	**Zone**
Enabled	**On**	
Filtering	**PrinterControl**	

.

Table 10.7 *Entries in the %LPR%, %LPRB%, %LPRC%, … parameter sets*

Key	*Type*	*Semantics*
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
Filtering	name	Indicates whether the input stream needs further filtering before the data can be correctly interpreted as a page description language. **Legal values**: /InterpreterBased, /None **Errors**: **configurationerror**, **rangecheck**, **typecheck** /InterpreterBased: Filters the input stream as necessary to conform to the language. For example, the data stream may have been sent to the printer system encoded as a tagged binary communication protocol (TBCP) PostScript job and must be decoded to a normal PostScript job before it is passed to the interpreter. See **Protocol** in Table 10.4 on page 274 for a description of /TBCP. /None: Passes the data unchanged to the interpreter.
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values:** *false* **Errors**: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269.

Table 10.7 *Entries in the %LPR%, %LPRB%, %LPRC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: /PSPrinter, /PJL
		Errors: **configurationerror, rangecheck, typecheck**
PrintHost	string	Lists at most two IP mask/address pairs where the mask is applied to the given IP address to specify which hosts are allowed to make LPR connections. The slash is used as a delimiter between the two subfields. If two address pairs are specified, they are separated by a space delimiter. The mask, which has the same syntax as the IP address, is optional. If an address is specified with no corresponding mask, a mask of 255.255.255.255 is assumed. For example, a mask/address pair in which the mask is not specified is: 138.46.24.37 Two IP mask/address pairs will have the format: 255.255.255.0/138.46.24.37 255.255.255.0/138.46.24.38 **Legal values**: An empty string or a string of 63 or fewer non-null characters that specifies up to two IP mask/addresses separated by the ASCII blank character. An IP address can be of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255. IP addresses cannot be set to illegal values; for example, trying to use an IP address equal to 0.0.0.0, 127.0.0.0, 255.255.255.255, *N.N.N.*255 or other illegal values will result in a **rangecheck** error. **Errors**: **typecheck**, **limitcheck**, **rangecheck**
ReceiveWindowSize	integer	Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter value in order to accommodate the host's expectations. The receive window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%. **Legal values**: An integer in the range 1024–65535 **Errors**: **typecheck**, **rangecheck**

Table 10.7 *Entries in the %LPR%, %LPRB%, %LPRC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
SendWindowSize	integer	Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter value in order to accommodate the host's expectations. The send window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%. **Legal values**: An integer in the range 1024–65535. **Errors**: **typecheck**, **rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Table 10.8 *Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets*

Key	*Type*	*Semantics*
ControlPortNumber	integer	Denotes a port used by the unit for handshaking between the host and the unit while setting up a session. A session with the printer system prevents other hosts from being able to interrupt the printer system to run other jobs. Communication is via TCP, not UDP. Suggested default value: 9101 **Legal values**: A positive integer representing a port number not reserved by any of the standard services. **Errors**: **typecheck**, **rangecheck**, **configurationerror**
	Warning	*An error is not raised when a port number previously reserved for some other purpose is specified.*
DataPortNumber	integer	Denotes a bidirectional port for transmission of printer language jobs. The suggested default value is 9100. Users are free to use another port number to avoid a conflict if another unit on the network is already using 9100. **Legal values**: A positive integer representing a port number not reserved by any of the standard services. **Errors**: **typecheck**, **rangecheck**, **configurationerror**

Table 10.8 *Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
	Warning	*An error is not raised when a port number previously reserved for some other purpose is specified.*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286. **Legal values**: /InterpreterBased, /None **Errors**: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values**: *false* **Errors**: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**

Table 10.8 *Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
PrintHost	string	Lists at most two IP mask/address pairs where the mask is applied to the given IP address to specify which hosts are allowed to make LPR connections. The slash is used as a delimiter between the two subfields. If two address pairs are specified, they are separated by a space delimiter. The mask, which has the same syntax as the IP address, is optional. If an address is specified with no corresponding mask, a mask of 255.255.255.255 is assumed. A mask/address pair in which the mask is not specified, for example, would be: 138.46.24.37 Two IP mask/address pairs would have the format: 255.255.255.0/138.46.24.37 255.255.255.0/138.46.24.38 **Legal values**: An empty string or a string of 63 or fewer non-null characters that specifies up to two IP mask/addresses separated by the ASCII blank character. An IP address can be of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255. IP addresses cannot be set to illegal values; for example, trying to use an IP address equal to 0.0.0.0, 127.0.0.0, 255.255.255.255, *N.N.N.*255 or other illegal values will result in a **rangecheck** error. **Errors**: **typecheck**, **limitcheck**, **rangecheck**
ReceiveWindowSize	integer	Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter value in order to accommodate the host's expectations. The receive window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%. **Legal values**: An integer in the range 1024–65535. **Errors**: **typecheck**, **rangecheck**

Table 10.8 *Entries in the %AppSocket%, %AppSocketB%, %AppSocketC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
SendWindowSize	integer	Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter value in order to accommodate the host's expectations. The send window size specified here overrides any request for this parameter in the associated sets of type /Parameters, for example, %TCP%. **Legal values**: An integer in the range 1024–65535. **Errors**: **typecheck**, **rangecheck**
StatusPortNumber	integer	Denotes a port used by the unit for the purpose of sending status information back to the host. When using TCP/IP, communication is via UDP, not the TCP transport layer. The suggested default value is 9101. Users may use another port number to avoid a conflict with another unit on the network already using 9101. **Legal values**: A positive integer representing a port number not reserved by any of the standard services. **Errors**: **typecheck**, **rangecheck**, **configurationerror**
	Warning	*An error is not raised when a port number is specified that has been previously reserved for some other purpose.*
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Table 10.9 *Entries in the %Telnet%, %TelnetB%, %TelnetC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269. **Legal values**: *true, false* **Errors**: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.

Table 10.9 *Entries in the %Telnet%, %TelnetB%, %TelnetC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269.
		Legal values: /PSPrinter, /PJL
		Errors: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Communications
		Errors: None

Table 10.10 *Entries in the %RemotePrinter%, %RemotePrinterB%, %RemotePrinterC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**

Table 10.10 *Entries in the %RemotePrinter%, %RemotePrinterB%, %RemotePrinterC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286.
		Legal values: /InterpreterBased, /None
		Errors: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.
		Legal values: *true, false*
		Errors: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269.
		Legal values: /PSPrinter, /PJL
		Errors: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Communications
		Errors: None

Table 10.11 *Entries in the %PrintServer%, %PrintServerB%, %PrintServerC%, … parameter set*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true, false*
		Errors: None

Table 10.11 *Entries in the %PrintServer%, %PrintServerB%, %PrintServerC%, … parameter set (continued)*

Key	*Type*	*Semantics*
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286.
		Legal values: /InterpreterBased, /None
		Errors: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
LoginPassword	string	Specifies the password that the print server application uses to gain access to the job queue. Setting this parameter to an empty string indicates that no password has been specified. The value of this parameter returned by the **currentdevparams** operator is the string (INVALID), regardless of what the password is set to. Attempts to set the **LoginPassword** to the string (INVALID) will be ignored.
		Legal values: A string of up to 32 characters.
		Errors: **limitcheck**, **typecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
PreferredServer	string	(*Read-write*) Name of the fileserver that the **PrintServer** attempts to attach to in order to service queues. The validity of the **PreferredServer** name is not checked; the value is passed directly to the nearest fileserver for routing request. Novell 3.x fileserver names are limited to 47 characters; spaces and the characters " * + , \ / | ; : = < > ? [ ] are illegal. Novel 4.x limits the parameter length to 64 characters; spaces are legal but not desirable. Novell 4.x converts spaces to underscores.
	Note	*Strings that contain spaces may not work on Novell NetWare 3.x.*

Table 10.11 *Entries in the %PrintServer%, %PrintServerB%, %PrintServerC%, … parameter set (continued)*

Key	*Type*	*Semantics*
		Legal values: A string of up to 64 characters. The default is an empty string.
		Errors: **typecheck**, **limitcheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269.
		Legal values: /PSPrinter, /PJL
		Errors: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Communications
		Errors: None

Table 10.12 *Entries in the %LAT%, %LATB%, %LATC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286.
		Legal values: /InterpreterBased, /None
		Errors: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
Groups	string	Defines the groups allowed to access a device (printer device). A group is defined as an integer value in the range 0–255. Designate multiple groups in a string using a space as a delimiter; designate a range of groups using a hyphen. For example, (3-8 200 145-160). Default value: 0 (gives all groups access)

Table 10.12 *Entries in the %LAT%, %LATB%, %LATC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: 0 to 255
		Errors: **rangecheck**
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269.
		Legal values: *false*
		Errors: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269.
		Legal values: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL
		Errors: **configurationerror**, **rangecheck**, **typecheck**
KeepaliveTimer	integer	Specifies the interval at which the circuit layer exchanges “keepalive” messages to maintain circuits when no session traffic is present.
		Legal values: 0 to 180 seconds
		Errors: **rangecheck**
MulticastTimer	integer	Specifies the interval at which directory service advertisements (SAs) are multicast to advertise the printer’s service.
		Legal values: 0 to 180 seconds
		Errors: **rangecheck**
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Physical	string	(*Read-only*) Specifies the physical layer over which LAT is accessed. The string is set to a device parameter set corresponding to a physical communications medium, such as the string (%EthernetPhysical%). The Physical parameter can associate one network layer parameter with one and only one physical layer parameter set.
		Legal values: A string of 32 or fewer non-null characters that specifies a physical layer.
		Errors: None
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269.
		Legal values: /PSPrinter, /PJL
		Errors: **configurationerror, rangecheck, typecheck**

Table 10.12 *Entries in the %LAT%, %LATB%, %LATC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
RetransmitTimer	integer	Specifies the interval at which the circuit layer retransmits unacknowledged messages. **Legal values**: 1 to 10 seconds **Errors**: **rangecheck**
RetransmitLimit	integer	Specifies the number of retransmissions that will be attempted before a circuit is declared dead. **Legal values**: 4 to 120 **Errors**: **rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None

Table 10.13 *Entries in the %LocalTalk%, %LocalTalkB%, %LocalTalkC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286. **Legal values**: /InterpreterBased, /None **Errors**: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values**: *false* **Errors**: None

Table 10.13 *Entries in the %LocalTalk%, %LocalTalkB%, %LocalTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
LocalTalkType	string	Represents the *type* component of the AppleTalk entity name. The entity consists of the three pieces—*zone*, *type*, and *object*, each of which is a string of 32 or fewer non-null characters. The *object* component is set to the value of the **PrinterName** system parameter, and the *zone* component is set to the wildcard character (asterisk). If the printer system also supports EtherTalk and/or TokenTalk communications, setting the **LocalTalkType** string will set the **EtherTalkType** and/or **TokenTalkType** parameter to the same value. The **appletalktype** compatibility operator will reflect a change to the **LocalTalkType** parameter. Therefore, getting the **LocalTalkType** parameter will always yield the same value as getting the **EtherTalkType** and/or the **TokenTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **limitcheck**, **typecheck**
NodeID	integer	(*Read-only*) Represents the local network address of the device. Legal addresses are 0 or values from 128 to 254. If the value of **NodeID** is 0, this indicates that the address has not been established. The value is used as an address hint when first establishing addresses as part of the AppleTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the AppleTalk device. **Legal values**: 0 or any integer from 128–254. **Errors**: None
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.

Table 10.13 *Entries in the %LocalTalk%, %LocalTalkB%, %LocalTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: /Communications
		Errors: None

Table 10.14 *Entries in the %EtherTalk%, %EtherTalkB%, %EtherTalkC%, … parameter sets*

Key	*Type*	*Semantics*
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
EthernetAddress	string	(*Read-only*) A unique string that represents the Ethernet address of the printer system. The string is of the form (*xx:xx:xx:xx:xx:xx*), where each *x* represents a digit in hexadecimal. **Legal values**: A string of 17 characters representing a legal Ethernet address. **Errors**: None
EtherTalkType	string	Represents the *type* component of the EtherTalk *entity name*. The *entity name* consists of three pieces—*zone*, *type*, and *object*, each of which is a string of 32 or fewer non-null characters. The *object* component is set to the value of the **PrinterName** system parameter. The *zone* component is set to the printer system zone name. If the printer system also supports LocalTalk and/or TokenTalk communications, setting the **EtherTalkType** string will set the **LocalTalkType** and/or **TokenTalkType** parameter to the same value. The **appletalktype** compatibility operator will reflect a change to the **EtherTalkType** parameter. Therefore, getting the **EtherTalkType** parameter will always yield the same value as getting the **LocalTalkType** and/or the **TokenTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **limitcheck**, **typecheck**

Table 10.14 *Entries in the %EtherTalk%, %EtherTalkB%, %EtherTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
EtherTalkZone	string	Represents the *zone* portion of the EtherTalk *entity name*. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **limitcheck**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286. **Legal values**: /InterpreterBased, /None **Errors**: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values:** *false* **Errors**: None
Interpreter	name	For the definition of **Interpreter**, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
NodeID	integer	(*Read-only*) Represents the local network address of the device. Legal addresses are values between 1 to 254. If the value of **NodeID** is 0, this indicates that the address has not been established. The value is used as an address hint when first establishing addresses as part of the AppleTalk protocol. As such, the parameter might not represent the actual address until that portion of the protocol is complete during initialization of the AppleTalk device. **Legal values**: 0 or any integer between 0 and 254. **Errors**: None
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.

Table 10.14 *Entries in the %EtherTalk%, %EtherTalkB%, %EtherTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: /Communications
		Errors: None

Table 10.15 *Entries in the %TokenTalk%, %TokenTalkB%, %TokenTalkC%, … parameter sets*

Key	*Type*	*Semantics*
Address	string	(*Read-only*) A unique string that represents the TokenRing address of the unit. The string is of the form (*xx*:*xx*:*xx*:*xx*:*xx*:*xx*), where each *x* represents a digit in hexadecimal.
		Legal values: A string of 17 characters representing a legal address.
		Errors: None
Bridging	name	Bridging on the TokenRing can be done in several different ways. Setting this parameter to /Transparent implies a transparent bridging, where the entire "universe" is one large single ring structure and all identities are unique. Setting this parameter to /SourceRoute implies that routing is done by specifying an explicit path, including the ring identification, RIF. Setting this parameter to the default value /Adaptive implies that the software will automatically recognize the routing style and respond in kind (either as a one-time determination or when processing each connection).
		Legal values: /Transparent, /SourceRoute, /Adaptive
		Errors: **configurationerror**, **typecheck**
DelayedOutputClose	boolean	For the definition of the **DelayedOutputClose** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: None
Enabled	boolean	For the definition of the **Enabled** parameter, see Table 10.3 on page 269.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Filtering	name	For the definition of the **Filtering** parameter, see Table 10.7 on page 286.
		Legal values: /InterpreterBased, /None
		Errors: **configurationerror**, **rangecheck**, **typecheck**
	Warning	*In a complete AppleTalk/Macintosh environment, the **Filtering** parameter should be set to /None, or you will encounter communications problems.*

Table 10.15 *Entries in the %TokenTalk%, %TokenTalkB%, %TokenTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
HasNames	boolean	For the definition of the **HasNames** parameter, see Table 10.3 on page 269. **Legal values**: *false* **Errors**: None
Interpreter	name	For the definition of the **Interpreter** parameter, see Table 10.3 on page 269. **Legal values**: /PostScript, /AutoSelect, /Diablo630, /EpsonFX850, /HP7475A, /LaserJetIII, /LaserJetIIP, /PCL, /ProprinterXL **Errors**: **configurationerror**, **rangecheck**, **typecheck**
NodeID	integer	(*Read-only*) For the definition of the **NodeID** parameter, see Table 10.14 on page 299. **Legal values**: 0 or any integer from 1 to 254. **Errors**: None
On	boolean	For the definition of the **On** parameter, see Table 10.3 on page 269. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
PrinterControl	name	For the definition of the **PrinterControl** parameter, see Table 10.3 on page 269. **Legal values**: /PSPrinter, /PJL **Errors**: **configurationerror, rangecheck, typecheck**
TokenTalkType	string	Represents the *type* component of the TokenTalk *entity name*. The *entity name* consists of three pieces—*zone*, *type*, and *object*, each of which is a string of 32 or fewer non-null characters. The *object* component is set to the value of the **PrinterName** system parameter. The *zone* component is set to the printer system zone name. If the printer system also supports LocalTalk and/or EtherTalk communications, setting the **TokenTalkType** string will set the **LocalTalkType** and/or **EtherTalkType** parameter to the same value. The **appletalktype** compatibility operator will reflect a change to the **TokenTalkType** parameter. Therefore, getting the **TokenTalkType** parameter will always yield the same value as getting the **LocalTalkType** and/or the **EtherTalkType** parameter and will match what is returned by the **appletalktype** compatibility operator. **Legal values**: Any string of 32 or fewer non-null characters. **Errors**: **typecheck**

Table 10.15 *Entries in the %TokenTalk%, %TokenTalkB%, %TokenTalkC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Communications **Errors**: None
Zone	string	Represents the *zone* component of the TokenTalk *entity name*. **Legal values:** Any string of 32 or fewer non-null characters. **Errors**: **typecheck**

10.5.2 Parameters Parameter Sets (Type = /Parameters)

This section describes the parameter sets of **Type** /Parameters, which correspond to the transport, network, data link, and physical layers of OSI (Figure 10.1 on page 261), as well as miscellaneous devices such as engine and calendar.

Certain parameter sets are associated with the lower layers of the OSI model, which do not connect directly to a PostScript interpreter. (The PostScript interpreter or emulator receives its jobs from the application layer.) The parameter sets associated with implementations of the network layer must reference a parameter set associated with an instance of the data link or physical layer (in other words, a distinct interface to the network). For example, if there is only one network layer implementation, but that implementation is connected to *n* network interfaces, then for each network layer parameter set there is a unique data link and physical parameter set named within it. A parameter set for the network layer can be viewed as a distinct binding of the network address and the network interface.

Certain parameter sets of **Type** /Parameters are used to control various network services, such as SNMP and the logging facility syslog (Syslog). Other parameter sets configure the SCSI bus, the IDE bus, the engine, and the console.

The following device parameter sets control the SNMP and Syslog network services:

- %SNMP%. SNMP allows the system administrator to query for information about the unit. The information that can be queried is driven by a database called a MIB. %SNMP% is not a communications port for PostScript jobs, thus the parameter set is of **Type** /Parameters. The parameters listed in Table 10.17 on page 308 are only those parameters

that need to be accessible from the PostScript language as opposed to the MIB(s) the device may have. These are the only parameters that are changeable from an environment separate from SNMP (the network side). The rules governing when changes take effect for each parameter within this parameter set are described in Table 10.17 on page 308.

For further information about SNMP, see "A Simple Network Management Protocol," Case, Fedor, Schoffstall, and Davin, Request for Comments 1157, DDN Network Information Center, SRI International, May 1990.

- %Syslog%. Syslog is a logging facility that sends log messages back to a UNIX host. Most of the messages contain network-specific information, but may include any other pertinent information the unit wishes to convey. Communication is via the UDP transport layer. Changes to this parameter set do not take effect until the unit is reinitialized. See Table 10.18 on page 310.

The following device parameter sets correspond to the lower protocol levels of the OSI application layer protocol model:

- %TCP%. TCP is the transport layer responsible for reliable data transfer by guaranteeing message delivery and reception. It is a connection-oriented protocol. Any packet that is lost will be retransmitted. Changes to parameters in the TCP parameter set do not take effect until the unit is reinitialized. See Table 10.19 on page 311.

- %UDP%. UDP is a connectionless (or datagram) protocol used in the TCP/IP networking suite. When using UDP with a peer host, there is no need for "handshaking" prior to communication. UDP packets are sent without any guarantee of delivery and may arrive at the destination in any order. See Table 10.20 on page 311.

- %IP%. IP is the network layer responsible for routing messages to their destinations. This layer decides which physical interface is to send outgoing messages and which transport layer is to receive incoming messages. Changes to parameters in the IP set do not take effect until the unit is reinitialized. See Table 10.21 on page 312.

- %SPX%. SPX is the Novell NetWare connection-oriented protocol. When using SPX to communicate with a peer host, there is "handshaking" before the connection is ready. The delivery of SPX packets is expected to be acknowledged to guarantee delivery, and packets arrive in sequence at the destination. Unlike TCP, it does not provide a sliding window functionality for flow control. See Table 10.22 on page 316.

- %IPX%. IPX is the Novell NetWare connectionless (or datagram) protocol. When using IPX with a peer host, there is no need for “handshaking” prior to communication. IPX packets are sent without any guarantee of delivery and may arrive at the destination in any order. NetWare broadcasting uses IPX. See Table 10.23 on page 316.

The following device parameter sets correspond to the lowest protocol layers of the OSI application layer model:

- %EthernetPhysical%. This parameter set corresponds to a physical Ethernet connector and its associated hardware and the data link layer software that handles events from this device. Changes to parameters in this set do not take effect until the unit is reinitialized. See Table 10.24 on page 318.

- %TokenRingPhysical%. This parameter set corresponds to a physical TokenRing connector and its associated hardware and the data link layer software that handles events from this device. Changes to parameters in this set do not take effect until the unit is reinitialized. See Table 10.25 on page 319.

The following parameter sets are used to configure the SCSI and IDE buses, the engine, the console, and the calendar:

- %Scsi%. This parameter set is used to configure the SCSI bus. It is always present in a printer system that has a SCSI bus, even if no devices are present on the SCSI bus. If more than one SCSI bus is present, the first is called %Scsi%, the second %ScsiB%, the third %ScsiC%, and so on. Changes to SCSI parameters do not take effect until the next time the system is initialized. See Table 10.26 on page 320.

- %Ide%. This parameter set is used to configure the IDE bus. It is always present in a printer system that has an IDE bus, even if no devices are present on the IDE bus. If more than one IDE bus is present, the first is called %Ide%, the second %IdeB%, the third %IdeC%, and so on. Changes to IDE parameters do not take effect until the next time the system is initialized. See Table 10.27 on page 322.

- %Engine%. This parameter set is used to configure the engine. See Table 10.28 on page 322.

- %Console%. This parameter set provides a means of setting and controlling characteristics of the operator console of an output device that includes the PostScript language. The keys currently defined in this set provide an extensible way of selecting the natural language (for example, English or Japanese) in which information will be displayed on the operator console. See Table 10.29 on page 324.

- %Calendar%. A printer has a battery-powered time-of-day clock. This clock must be set initially, and then twice a year to follow daylight savings time. The string (%Calendar%) identifies the calendar device, and the entries in the dictionary describe the local date and time.

Required entries for the parameter sets of **Type** /Parameter are summarized below. Details of the required entries are given in the referenced tables.

%SNMP%
(Table 10.17 on page 308)

PrivateHost	**SysLocation**	**TrapHost**
SysContact	**SysName**	**Type**

%Syslog%
(Table 10.18 on page 310)

LogHost	**LogPriority**	**Type**

%TCP%
(Table 10.19 on page 311)

On	**ReceiveWindowSize**	**SendWindowSize**
Type		

%UDP%
(Table 10.20 on page 311)

CheckSum	**On**	**Type**

%IP%
(Table 10.21 on page 312)

BroadcastAddress	**IPAdressDynamic**	**Physical**
GatewayAddress	**NetworkMask**	**TransmitEncapsulation**
IPAddress	**On**	**Type**

%SPX%
(Table 10.22 on page 316)

On	**ReceiveWindowSize**	**Type**

%IPX%
(Table 10.21 on page 312)

CheckSum	**On**	**Type**
HopCount	**Physical**	
NetworkAddress	**TransmitEncapsulation**	

%EthernetPhysical%
(Table 10.24 on page 318)

ConnectorType	**Name**	**Type**
EthernetAddress	**On**	

%TokenRingPhysical%
(Table 10.25 on page 319)

Address	**Name**	**Type**
Bridging	**On**	
ConnectorType	**Speed**	

%Scsi%
(Table 10.26 on page 320)

BootDelay	**InitiatorId**	**TargetId**
CheckParity	**Poll**	**Type**

%Ide%
(Table 10.27 on page 322)

BootDelay	**Poll**	**Type**

%Engine%
(Table 10.28 on page 322)

BSizeStandard	**DarknessCyan**	**PageCount**
Darkness	**DarknessMagenta**	**TimeToStandby**
DarknessBlack	**DarknessYellow**	**Type**

%Console%
(Table 10.29 on page 324)

CharSet	**Language**	**Type**
Country		

%Calendar%
(Table 10.30 on page 327)

Day	**Month**	**Type**
Hour	**Running**	**Year**
Minute	**Second**	

Node Address

A *node address* is a unique address for a node on some network. The form of the node address depends on the protocol being used to communicate with the node. Table 10.16 lists the various node address forms.

Table 10.16 *Form of the node address*

Protocol	*Node address forms*	*Description*
TCP/IP	*N.N.N.N*	Each *N* is a decimal number in the range 0–255.
Novell SPX/IPX	*XXXXXXXX:xxxxxxxxxxxx*	Each *X* and *x* is a hexadecimal digit in the range 0–F (upper or lower case). *XXXXXXXX* is the network part of the address, and *xxxxxxxxxxxx* is the *Media Access Control* part known as the *Novell Node Number.*
AppleTalk DDP	*N.N.n*	Each *N* and *n* is a decimal number in the range 0–25. *N.N* represents the network part of the address, and *n* represents the node ID.

Required Entries for the /Parameter Parameter Sets

Table 10.17 *Entries in the %SNMP%, %SNMPB%, %SNMPC%, … parameter sets*

Key	*Type*	*Semantics*
PrivateHost	string	A single node address (Table 10.16 on page 308) per protocol of a host that is able to set those SNMP variables that can be written. An empty string indicates that no host has access. An empty string is the usual default value, thus the unit will need to have this parameter explicitly set via the operator **setdevparams** prior to using SNMP. **Legal values**: An empty string or a string (of 49 or fewer non-null characters) that specifies up to one node address per protocol separated by the ASCII blank character. **Errors**: **typecheck, rangecheck, limitcheck**
SysContact	string	Traditionally the name and either the phone number or the address of the person responsible for the unit. Changes to this parameter take effect immediately. **Legal values**: A string of 32 or fewer non-null characters. **Errors**: **typecheck, limitcheck**
SysLocation	string	Location of the raster output device unit. Changes to this parameter take effect immediately. **Legal values**: A string of 32 or fewer non-null characters. **Errors**: **typecheck, limitcheck**

Table 10.17 *Entries in the %SNMP%, %SNMPB%, %SNMPC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
SysName	string	Name of the raster output device unit (expected by SNMP). Changes to this parameter take effect immediately. **Legal values**: A string of 32 or fewer non-null characters. **Errors**: **typecheck, limitcheck**
TrapHost	string	A list of one or more (*node-address*/*community*) pairs for each protocol with a host that is able to receive traps. See Table 10.16 on page 308 for the syntax of a node address. A slash is used as a delimiter between the *node-address* and the *community* string. The ASCII blank is used to separate each pair in the list. The *community* string portion is case insensitive. An empty string indicates that no traps are being sent to the host. Here are some example *community* strings: • *public* • *proxy* • *private* • *regional* • *core* For example, the value (130.248.224.46/public) is an IP address for a trap host node in a public community. The empty string is the usual default value, thus the unit will need to have this parameter explicitly set via the operator **setdevparams** prior to using the trap host facility. **Legal values**: An empty string or a string that specifies one or more (*node-address*/*community*) pairs separated by the ASCII blank character. **Errors**: **typecheck, rangecheck, limitcheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Parameters **Errors**: None

Table 10.18 *Entries in the %Syslog%, %SyslogB%, %SyslogC%, … parameter sets*

Key	*Type*	*Semantics*
LogHost	string	Contains an IP address for a host that receives Syslog messages from the unit. An empty string indicates that no Syslog messages are to be sent by the unit. A null string implies that Syslog messages are disabled. **Legal values**: An empty string or a string (of 15 or fewer non-null characters) that specifies a legal IP address. An IP address is of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255. Trying to use an IP address equal to 0.0.0.0, 255.255.255.255 or *N.N.N*.255 will result in a **rangecheck** error. **Errors**: **typecheck, limitcheck, rangecheck**
LogPriority	integer	Designates which logging messages are to be sent on to the Syslog host. All logging messages associated with the specified **LogPriority** and those of higher priority (smaller numbers are higher priority) are sent. The following is a list, from the BSD UNIX and SunOS™ convention, of priorities and their corresponding meanings: 0 Unit is no longer usable. 1 Messages indicating that immediate action is needed on the part of a system administrator. 2 Critical error messages. 3 Error messages. 4 Warning messages. 5 Normal but significant conditions. 6 Informational messages. 7 Debugging messages. **Legal values**: An integer in the range 0–7. **Errors**: **typecheck, rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Parameters **Errors**: None

Table 10.19 *Entries in the %TCP%, %TCPB%, %TCPC%, … parameter sets*

Key	*Type*	*Semantics*
On	boolean	A value of *true* means that the TCP protocol is activated at boot time. Otherwise it is off. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
ReceiveWindowSize	integer	Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter value in order to accommodate the host's expectations. **Legal values**: An integer in the range 1024–65535. **Errors**: **typecheck, rangecheck**
SendWindowSize	integer	Specifying the send window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter states in order to accommodate the host's expectations. **Legal values**: An integer in the range 1024–65535. **Errors**: **typecheck, rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values:** /Parameters **Errors**: None

Table 10.20 *Entries in the %UDP%, %UDPB%, %UDPC%, … parameter sets*

Key	*Type*	*Semantics*
Checksum	boolean	Specifies whether checksum values will be inserted in outgoing packets formed by the software. The default should be *true*. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
On	boolean	A value of *true* means that the UDP protocol is activated at boot time. Otherwise it is off.

Table 10.20 *Entries in the %UDP%, %UDPB%, %UDPC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Parameters
		Errors: None

Table 10.21 *Entries in the %IP%, %IPB%, %IPC%, … parameter sets*

Key	*Type*	*Semantics*
BroadcastAddress		
	string	Broadcast address mask used when broadcasting messages to the local network. The **BroadcastAddress** parameter reflects the current broadcast address mask in use by the unit. In order to "set" the **BroadcastAddress** explicitly and have it take effect the next time the unit is initialized, you must have the **IPAddressDynamic** parameter set to *false* when issuing the **setdevparams** operator.
	Note	*If the value of **BroadcastAddress** is not legal with respect to the **IPAddress** and **NetworkMask** parameters, it is changed to a value that is legal without warning the user. For example, suppose the **IPAddress** is 134.14.15.16 and the **BroadcastAddress** is 134.14.255.255. If the user changes **IPAddress** to 134.15.15.16 without explicitly changing the **BroadcastAddress**, the **BroadcastAddress** is automatically changed to 134.15.255.255.*
		Legal values: A string of 15 or fewer non-null characters that specifies a legal **BroadcastAddress**. A **BroadcastAddress** is of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255.
		Errors: **typecheck, rangecheck, limitcheck**
GatewayAddress		
	string	Contains the *(destination-address/gateway-address)* pairs to other networks. An empty string specifies that dynamic routing, if available in the product, is enabled. In accordance with the route command, the value of **GatewayAddress** is defined here as a *destination-address* and a *gateway-address*. A slash is used as a delimiter between the two fields. Multiple address pairs can be specified and are separated by a space delimiter. The number of *(destination-address/gateway-address)* pairs is implementation dependent.

Table 10.21 *Entries in the %IP%, %IPB%, %IPC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		The *destination-address* specifies the network address. A valid network address varies according to the class. A class A network address requires the first field be non-zero. The others may be zero (for a net with no subnets), or contain subnet address information. A class B network address requires the first two fields to be non-zero. A class C network address requires the first three fields to be non-zero. A network address of 0.0.0.0 is a special case used by default if no previous network address matches the desired target IP address. If multiple entries have the same address, then the earlier entry will be ignored. The value of **GatewayAddress** reflects the current (*destination address/gateway-address*) pairs to other networks in use by the unit. The **GatewayAddress** parameter may be "set" explicitly at any time. It will take effect only upon unit initialization, and then only if the **IPAddressDynamic** parameter is set to *false* using the **setdevparams** operator. If the **IPAddressDynamic** parameter is set to *true* at unit initialization time, the **GatewayAddress** parameter will not take effect. Routing information is gathered via dynamic routing using a RIP (Routing information protocol) request to the network. The default should be the empty string, implying dynamic routing. **Legal values:** An empty string or a string that specifies one or more legal (*destination-address/gateway address*) pairs. The maximum length of the string is $32n - 1$, where n is the number of (*destination-address/gateway-address*) pairs. The value n is product specific. These addresses are IP addresses of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255. Loopback addresses (127.*N.N.N*) and broadcast addresses (*N.N.N*.255) are illegal for either the destination or the gateway part of the pair. **Errors**: **typecheck, rangecheck, limitcheck**
IPAddress	string	A unique string that represents the IP address of the unit. The IP address is mapped directly to the lowest physical address by which the unit is known (for example, **EthernetAddress** if **Physical** is %EthernetPhysical%). The value of **IPAddress** reflects the current IP address in use by the unit. In order to "set" the **IPAddress** explicitly and have it take effect the next time the unit is initialized, you must have **IPAddressDynamic** set to *false* when issuing the **setdevparams** operator. The default is an empty string, which implies that the IP protocol layer is not active.

Table 10.21 *Entries in the %IP%, %IPB%, %IPC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
	Note	*Whenever the value of* ***IPAddressDynamic*** *is true,* the ***currentdevparams*** *operator will return a value for the parameter* ***IPAddress*** *that has been determined by a BOOTP or RARP sequence during boot up. Changing the* ***IPAddress*** *parameter to some other value via the* ***setdevparams*** *operator has the effect of changing the user-explicit value, which is only used if* ***IPAddressDynamic*** *is false. The* ***currentdevparams*** *operator will return the user-explicit value of* ***IPAddress*** *only when the value of* ***IPAddressDynamic*** *is false.* **Legal values:** An empty string or a string (of 15 or fewer non-null characters) that specifies a legal IP address. An IP address is of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255. Trying to set an IP address equal to 0.0.0.0, 255.255.255.255, 127.*N.N.N*, *N.N.N*.0, *N.N.N*.255 or any address whose first field is in the range 224–255 will result in a **rangecheck** error. **Errors**: **typecheck, rangecheck, limitcheck**
IPAddressDynamic	boolean	A value of *true* indicates that the **IPAddress** is obtained by a BOOTP or RARP sequence during boot up. A value of *false* means that the **IPAddress** must be explicitly set by a PostScript job via the **setdevparams** operator in order for connections to be made on the local network. The default value is usually *false*. **Legal values**: *true*, *false* **Errors**: **typecheck**
NetworkMask	string	Indicates which fields of the **IPAddress** parameter designate the network portion of the IP address and which designate the node portion. For example, the value 255.255.255.0 is a **NetworkMask** for a class B network with subnets. The **NetworkMask** value is used to determine if a certain IP address is on the same network as the unit. The value of **NetworkMask** will reflect the current network mask in use by the unit. In order to "set" the **NetworkMask** explicitly and have it take effect the next time the unit is initialized, you must have **IPAddressDynamic** set to *false* when issuing the **setdevparams** operator. If **IPAddress** is set to *true*, the **GatewayAddress** parameter should be set to appropriately legal values since dynamic routing is not always reliable when **IPAddress** is received via RARP.

Table 10.21 *Entries in the %IP%, %IPB%, %IPC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
	Note	*If the value of **NetworkMask** is not legal with respect to the **IPAddress**, the value of the **NetworkMask** parameter is changed to a value that is legal without warning the user. For example, if a class B **IPAddress** is given with a class A network mask, the **NetworkMask** will be changed to the default class B network mask. The default class A network mask is 255.0.0.0. The default class B network mask is 255.255.0.0. The default class C network mask is 255.255.255.0. No subnets are accounted for in these default network masks.*
		Legal values: A string of 15 or fewer non-null characters that specifies a legal IP mask. IP masks are of the form *N.N.N.N*, where each *N* is a decimal number in the range 0–255.
		Errors: **limitcheck, typecheck**
On	boolean	A value of *true* means that the IP protocol is activated at boot time. Otherwise it is off.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Physical	string	(*Read-only*) Specifies the physical layer over which IP is accessed. The string designates a device parameter set corresponding to a physical communications medium, such as the string %EthernetPhysical%. A network layer parameter set can be associated with one and only one physical layer parameter set by the **Physical** parameter.
		Legal values: A string of 32 or fewer non-null characters that specifies a physical layer.
		Errors: None
TransmitEncapsulation	name	Specifies the transmit encapsulation type. /SNAP indicates either an 802.2 header or an 802.5 header with a SNAP header. /DIX indicates Ethernet II headers. The default value should be /DIX for Ethernet. The default value is (and can reasonably only be) /SNAP for TokenRing.
		Legal values: /SNAP, /DIX
		Errors: **typecheck**
	Note	*These new values have been introduced to eliminate dependencies on the type of connection used (Ethernet or TokenRing). Legal values in the 2016 Supplement were /802.3-2-SNAP, /DIX, and /802.5-2-SNAP.*
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Parameters
		Errors: None

Table 10.22 *Entries in the %SPX%, %SPXB%, %SPXC%, … parameter sets*

Key	*Type*	*Semantics*
On	boolean	A value of *true* means that the SPX protocol is activated at boot time. Otherwise it is off. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
ReceiveWindowSize	integer	Specifying the receive window size is a means of tuning the code for optimal throughput. This setting is enacted at boot time, when memory is allocated for use by the network communications software. The actual window size is established when the connection is opened and may be smaller than this parameter in order to accommodate the host's expectations. **Legal values**: An integer in the range 1024–59392 **Errors**: **typecheck, rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values:** /Parameters **Errors**: None

Table 10.23 *Entries in the %IPX%, %IPXB%, %IPXC%, … parameter sets*

Key	*Type*	*Semantics*
Checksum	boolean	Specifies whether checksum values will be inserted in outgoing packets formed by the software. The default should be *true*. **Legal values**: *true*, *false* **Errors**: None
HopCount	integer	(*Read-write*) Specifies the maximum number of routers the print server will go through in trying to attach to fileservers while looking for queues to service. The preferred value is the smallest value needed to reach all of the printer system's servers. A count of 15 is defined as trying to reach all reachable servers; a count of 16 is defined as unreachable. If the **PreferredServer** parameter is set, the **HopCount** parameter is ignored unless the **PreferredServer** value is unreachable. A negative value or a value larger than 15 will default to 15. **Legal values**: 0 to 15 **Errors**: **typecheck**

Table 10.23 *Entries in the %IPX%, %IPXB%, %IPXC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
NetworkAddress	string	(*Read-only*) Identifies the network in which the unit is located. The concatenation of the **NetworkAddress** value and the Novell Node Number will uniquely identify the unit on the network. The Novell Node Number is derived from the MAC address of the networking media. For Ethernet, the Novell Node Number is the **EthernetAddress** parameter of the %EthernetPhysical% set. The **NetworkAddress** is obtained from the Novell file server on the local net upon booting the printer system. **Legal values**: An empty string or a string (of 8 or fewer non-null characters) that specifies a legal Novell network address. A Novell network address is of the form *XXXXXXXX*, where each *X* represents a digit in hexadecimal in the range 0–F. **Errors**: None
On	boolean	A value of *true* means that the IPX protocol is activated at boot time. Otherwise it is off. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
Physical	string	(*Read-only*) Specifies the physical layer over which IPX is accessed. The string designates a device parameter set corresponding to a physical communications medium, such as the string %EthernetPhysical%. A network layer parameter set can be associated with one and only one physical layer parameter set by the **Physical** parameter. **Legal values**: A string of 32 or fewer non-null characters that specifies a physical layer. **Errors**: None
TransmitEncapsulation	name	Specifies the transmit encapsulation type. /NO_SNAP indicates either an 802.2 or 802.5 header without a SNAP header. /SNAP indicates either an 802.2 or 802.5 header with a SNAP header. The default value should be /802.3 when the value of **Physical** is %EthernetPhysical% and /NO_SNAP when the value of **Physical** is %TokenRingPhysical%. /DIX and /802.3 are applicable only to Ethernet. The default value is /802.3. /DIX indicates Ethernet Version II. /Adaptive indicates that, by the nature of the interaction between host and printer system, an encapsulation format to use in responses to the host can be derived at boot time. The value of this parameter is checked solely for legality; it is not checked for applicability. The relationships between the values of **TransmitEncapsulation**, **Physical**, /EthernetPhysical, and /TokenRingPhysical are as follows:

Table 10.23 *Entries in the %IPX%, %IPXB%, %IPXC%, … parameter sets (continued)*

Key	Type	Semantics

Novell FrameType	TransmitEncapsulation	Physical
Ethernet_802.3	/802.3	/EthernetPhysical
Ethernet_802.2	/NO_SNAP	/EthernetPhysical
TOKEN_RING	/NO_SNAP	/TokenRingPhysical
Ethernet_SNAP	/SNAP	/EthernetPhysical
TOKEN_RING_SNAP	/SNAP	/TokenRingPhysical
Ethernet_II	/DIX	/EthernetPhysical
Any	/Adaptive	/EthernetPhysical or /TokenRingPhysical

Legal values: /802.3, /NO_SNAP, /SNAP, /DIX, /Adaptive

Errors: **typecheck**

Note *These new values have been introduced to eliminate dependencies on the type of connection used. Legal values in the 2016 Supplement were /802.3-2, /802.3-X, /802.3-2-SNAP, /802.5-2-SNAP.*

Type name (*Read-only*) Designates the general category of parameters in the parameter set.

Legal values: /Parameters

Errors: None

Table 10.24 *Entries in the %EthernetPhysical%, %EthernetPhysicalB%, %EthernetPhysicalC%, … parameter sets*

Key	Type	Semantics

ConnectorType name (*Read-only*) Indicates which Ethernet connector type is being used.

Legal values: /RJ45, /BNC, /AUI, /AAUI

Errors: None

EthernetAddress

string *(Read-only)* Returns a unique string that represents the Ethernet address of the unit. The string is of the form (*XX*:*XX*:*XX*:*XX*:*XX*:*XX*), where each *X* represents a digit in hexadecimal.

Note *When Novell is used, the Ethernet address is also known as the Novell Node Number.*

Legal values: A string of 17 characters representing a legal Ethernet address.

Errors: None

Name string *(Read-only)* Specifies the mnemonic name, such as (le0) for “Lance chip interface unit 0” or (so0) for “Sonic chip interface unit 0,” for the Ethernet interface used.

Table 10.24 *Entries in the %EthernetPhysical%, %EthernetPhysicalB%, %EthernetPhysicalC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: Any string of 16 or fewer non-null characters.
		Errors: None
On	boolean	A value of *true* means that the Ethernet channel is enabled at boot time. Otherwise it is off.
		Legal values: *true*, *false*
		Errors: **configurationerror**, **typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Parameters
		Errors: None

Table 10.25 *Entries in the %TokenRingPhysical%, %TokenRingPhysicalB%, %TokenRingPhysicalC%, … parameter sets*

Key	*Type*	*Semantics*
Address	string	*(Read-only)* Returns a unique string that represents the Ethernet address of the unit. The string is of the form (*XX*:*XX*:*XX*:*XX*:*XX*:*XX*), where each *X* represents a digit in hexadecimal.
		Legal values: A string of 17 characters representing a legal TokenRing address.
		Errors: None
Bridging	name	Bridging on the TokenRing can be done in several different ways. When this parameter is set to /Transparent, this implies a transparent bridging, where the entire "universe" is one large single ring structure and all identities are unique. When set to /SourceRoute, routing is done by specifying an explicit path, including the ring identification RIF. When set to /Adaptive, the software will automatically recognize the routing style and respond in kind (either as a one-time determination or when each connection is processed).
		Legal values: /Transparent, /SourceRoute, /Adaptive
		Errors: **configurationerror, typecheck**
ConnectorType	name	*(Read-only)* Indicates which TokenRing connector type is being used.
		Legal values: /RJ45, /DB9, /MAU
		Errors: None

Table 10.25 *Entries in the %TokenRingPhysical%, %TokenRingPhysicalB%, %TokenRingPhysicalC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Name	string	*(Read-only)* Specifies the mnemonic name, such as (le0) for "Lance chip interface unit 0" or (so0) for "Sonic chip interface unit 0," for the TokenRing interface used. **Legal values**: Any string of 16 or fewer non-null characters. **Errors**: None
On	boolean	A value of *true* means that the TokenRing channel is enabled at boot time. Otherwise it is off. **Legal values**: *true*, *false* **Errors**: **configurationerror**, **typecheck**
Speed	integer	Indicates the speed at which the ring is operated, in megabits per second. **Legal values**: 4, 16 **Errors**: **configurationerror**, **typecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Parameters **Errors**: None

Table 10.26 *Entries in the %Scsi%, %ScsiB%, %ScsiC%, … parameter sets*

Key	*Type*	*Semantics*
BootDelay[a]	integer	Indicates how long the disk I/O driver should wait (in seconds) during system initialization for the disk to spin up, before determining that a disk is not present or not responding. A value of 0 means that there is no waiting for the disk to spin up. Set this parameter in accordance with the characteristics of the disk attached to the printer system. **Legal values**: 0 or any positive integer**.** **Errors**: None
CheckParity	boolean	Indicates if parity on the SCSI bus is to be checked. The default value is usually *true*. *Warning* *Setting **CheckParity** to true on products that do not support parity checking would be unwise.* **Legal values**: *true*, *false* **Errors**: None

Table 10.26 *Entries in the %Scsi%, %ScsiB%, %ScsiC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
InitiatorId[a]	integer	Address on the SCSI bus used by the printer system when it serves as initiator. The default value is usually 7. **Legal values**: Any integer in the range 0– 7. **Errors**: **configurationerror**
Poll[a]	integer	A bit-encoded specification indicating which addresses on the SCSI bus should be polled by the printer system when it looks for disks during system initialization. For example, a 1 in bit 0 means poll for %disk0%. Any bits in this mask that correspond to addresses that are used as the printer system's **InitiatorId** or **TargetId** address, as the **InitiatorId** address for other hosts on the bus, or as the **TargetId** address of peripherals belonging to other hosts on the bus should be set to 0 (meaning "do not poll"). If the bit is set to poll the address corresponding to the printer system's **InitiatorId** or **TargetId** address, a **configurationerror** is generated. If the bit is set to poll an address that shouldn't be polled, anomalies may occur on the bus. The value of **Poll** is expressed as an integer bit mask in the range 0–254 (never 255 since all bits cannot be on—one bit must be reserved for the **InitiatorId** address). The default value is usually 127 (7F in hexadecimal). **Legal values**: An integer bit mask in the range 0–254. **Errors**: **configurationerror**
TargetId[b]	integer	The SCSI bus address reserved by the printer system for use as the %ScsiComm% communications channel. This address may be the same as the **InitiatorId** address. **Legal values**: Any integer in the range 0–7. **Errors**: **configurationerror**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Parameters **Errors**: None

a. Present only when disks are present on the bus.
b. Present only when ScsiComm is used on the bus

Table 10.27 *Entries in the %Ide%, %IdeB%, %IdeC%, … parameter sets*

Key	*Type*	*Semantics*
BootDelay[a]	integer	Indicates how long the disk I/O driver should wait (in seconds) during system initialization for the disk to spin up, before determining that a disk is not present or not responding. A value of 0 means that there is no waiting for the disk to spin up. Set this parameter in accordance with the characteristics of the disk attached to the printer system. **Legal values**: 0 or any positive integer. **Errors**: None
Poll[a]	integer	A bit-encoded specification indicating which addresses on the SCSI bus should be polled by the printer system when it looks for disks during system initialization. For example, a 1 in bit 0 means poll for %disk0%. The value of **Poll** is expressed as an integer bit mask in the range 0–3. **Legal values**: An integer bit mask in the range 0–3. **Errors**: **configurationerror**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values**: /Parameters **Errors**: None

a. Present only when disks are present on the bus.

Table 10.28 *Entries in the %Engine%, %EngineB%, %EngineC%, … parameter sets*

Key	*Type*	*Semantics*
BSizeStandard	name	Assists the engine in determining the physical dimensions of the paper when B4 or B5 paper is selected. There are two choices for the value of **BSizeStandard**: /ISO: ISO is the international body that defines the "metric" paper sizes (A4, A3, B5, B4, B3, and so forth). These are the paper sizes used in Europe and much of the rest of the world. The table below lists the dimensions for the B4 and B5 paper sizes as defined by ISO: *Paper Size* — *Dimensions* B4 — 250 x 353 mm or 709 x 1001 default units. B5 — 176 x 250 mm or 499 x 709 default units. /JIS: The Japanese Industrial Standards Committee is the national body that specifies standards for use in the country of Japan. Japan also uses the standard "A" paper sizes. However, they use a slightly different definition of the "B" paper sizes. The table below lists the dimensions for the B4 and B5 paper sizes for JIS:

Table 10.28 *Entries in the %Engine%, %EngineB%, %EngineC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Paper Size *Dimensions* B4 257 x 364 mm or 729 x 1032 default units. B5 182 x 257 mm or 516 x 729 default units.
	Note	*In the above tables, a "default unit" denotes 1/72 of an inch.*
		Legal values*:* /ISO, /JIS
		Errors*:* **rangecheck**
Darkness	real	Controls the overall lightness or darkness of the rendered page on a monochrome device. This parameter does not affect the frame buffer, nor does it have any computational overhead. Legal values are real numbers from 0.0 through 1.0. A value of 0.0 means minimum darkness; 1.0 means maximum darkness. This option is provided in some products whose marking hardware allows software control of colorant application.
		Legal values*:* Real numbers in the range 0.0–1.0.
		Errors*:* **rangecheck**
DarknessBlack	real	Controls the overall lightness or darkness of the black color on a rendered page produced on a device with multiple toner stations. See **Darkness** for a complete description.
		Legal values*:* Real numbers in the range 0.0–1.0.
		Errors*:* **rangecheck**
DarknessCyan	real	Controls the overall lightness or darkness of the cyan color on a rendered page. See **Darkness** for a complete description.
		Legal values*:* Real numbers in the range 0.0–1.0.
		Errors*:* **rangecheck**
DarknessMagenta	real	Controls the overall lightness or darkness of the magenta color on a rendered page. See **Darkness** for a complete description.
		Legal values*:* Real numbers in the range 0.0–1.0.
		Errors*:* **rangecheck**
DarknessYellow	real	Controls the overall lightness or darkness of the yellow color on a rendered page. See **Darkness** for a complete description.
		Legal values*:* Real numbers in the range 0.0–1.0.
		Errors*:* **rangecheck**
PageCount	integer	A count of all pages fed by the engine. The count includes all of the pages successfully printed as well as the pages that were jammed or spoiled. The value of **PageCount** is determined by querying the engine.

Table 10.28 *Entries in the %Engine%, %EngineB%, %EngineC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: Any positive integer or 0.
		Errors: **typecheck**
TimeToStandby	integer	After the specified number of minutes, the engine will go into a "standby" mode, in which it stops trying to keep itself ready to print a page; that is, it stops keeping its fuser hot. The next time the controller sends a feed or prefeed command, the engine will enter the "warming up" state until it is ready to print. The range of acceptable values for **TimeToStandby** are product specific. An illegal value is rounded to the nearest allowed value. Specifying a value of 0 for this parameter has the effect of never letting the printer system enter the "standby" mode.
		Legal values: Product-specific. Typically an integer in the range 0–720.
		Errors: **rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.
		Legal values: /Parameters
		Errors: None

Table 10.29 *Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets*

Key	*Type*	*Semantics*
CharSet	name	Specifies the name of a character set. This may be either the name of a standard character set or a vendor-specific character set. If it is a standard character set, the name will designate the standard (and where applicable, the variant within the standard), for example, ISO-646-ISV (for ASCII). If it is vendor specific, then the name should designate the vendor and the identification of the character set used by that vendor, for example, IBM-Codepage-550. Because the same character set may be known by several names (for example, ASCII and ISO-646-ISV), aliases are allowed for character set names; that is, the same character set may be designated by more than one name.
		Legal values:
		/ASCII — This is the same as ISO-646-IRV except for the "$."
		/ISO-646-ISV — This is ASCII with a currency symbol instead of "$."
		/ISO-8859-1 — This is the ISO 8-bit Latin-1 character set.
		/Adobe-Japan1-0 — This is the CID-keyed Japanese character collection.
		Errors: **rangecheck**

Table 10.29 *Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets (continued)*

Key	*Type*	*Semantics*	
Country	name	Indirectly specifies the dialect of the language by referring to the country in which the dialect is used. The country is indicated by a two-character country code from the ISO 3166 Standard (these codes represent the names of countries).	
		Legal values:	(source — ISO 3166)
		Value	*Meaning*
		/AR	Argentina
		/AU	Australia
		/BE	Belgium
		/BO	Bolivia
		/BR	Brazil
		/CA	Canada
		/CL	Chile
		/CN	Peoples Republic of China
		/TW	Taiwan
		/CO	Columbia
		/CS	Czechoslovakia (probably obsolete)
		/DK	Denmark
		/EC	Ecuador
		/FI	Finland
		/FR	France
		/DE	Germany
		/GR	Greece
		/HU	Hungary
		/IN	India
		/ID	Indonesia
		/IE	Ireland
		/IL	Israel
		/IT	Italy
		/JP	Japan
		/LU	Luxembourg

Table 10.29 *Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets (continued)*

Key	*Type*	*Semantics*	
		/MX	Mexico
		/NL	Netherlands (Holland)
		/NZ	New Zealand
		/NO	Norway
		/PK	Pakistan
		/PA	Panama
		/PY	Paraguay
		/PE	Peru
		/PH	Philippines
		/PL	Poland
		/PT	Portugal
		/SA	Saudi Arabia
		/ZA	South Africa
		/ES	Spain
		/SE	Sweden
		/GB	United Kingdom (England, Wales, Scotland, N. Ireland)
		/US	United States
		/SU	Soviet Union (probably obsolete)
		/VE	Venezuela
		Errors:	**typecheck**
Language	name	Specifies a two-character language code from the ISO 639 Standard Code for the representation of names of languages.	
		Legal values:	(source — ISO 639)
		Value	*Meaning*
		/CS	Czech
		/DA	Danish
		/DE	German
		/EL	Greek
		/EN	English
		/FI	Finnish

Table 10.29 *Entries in the %Console%, %ConsoleB%, %ConsoleC%, … parameter sets (continued)*

Key	*Type*	*Semantics*	
		/FR	French
		/GA	Irish
		/HU	Hungarian
		/IT	Italian
		/IW	Hebrew
		/JA	Japanese
		/KO	Korean
		/NL	Dutch
		/NO	Norwegian
		/PL	Polish
		/PT	Portuguese
		/RU	Russian
		/SK	Slovak
		/SV	Swedish
		/ZH	Chinese
		Errors:	**rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set.	
		Legal values:	/Parameters
		Errors:	None

Table 10.30 *Entries in the %Calendar%, %CalendarB%, %CalendarC%, … parameter sets*

Key	*Type*	*Semantics*	
Day	integer	Represents the day of the month.	
		Legal values:	An integer in the range 1–31.
		Errors:	**rangecheck**
Hour	integer	Represents the hour.	
		Legal values:	An integer in the range 0–23.
		Errors:	**rangecheck**

Table 10.30 *Entries in the %Calendar%, %CalendarB%, %CalendarC%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Minute	integer	Represents the minute. **Legal values**: An integer in the range 0–59. **Errors**: **rangecheck**
Month	integer	Represents the month. **Legal values**: An integer in the range 1–12. **Errors**: **rangecheck**
Running	boolean	Turns the clock off and on. When turning the clock on (setting the value to *true*), the time elements should also be set at the same time in order to avoid a **rangecheck** error. The clock must be on in order to set the time. If the clock is turned off (to preserve battery power) or is assumed to be inaccurate, the time returned is January 1, 1980 00:00:00. **Legal values**: *true, false* **Errors**: **rangecheck**
Second	integer	Represents the second. **Legal values**: An integer in the range 0–59. **Errors**: **rangecheck**
Type	name	(*Read-only*) Designates the general category of parameters in the parameter set. **Legal values:** /Parameters **Errors**: None
Year	integer	Represents the year. The value of this parameter returned by the **currentdevparams** operator has special significance. If it is in the range 1980–2079, then it represents the year. If it is 0, then the clock is either turned off (to preserve battery power) or assumed to be inaccurate. **Legal values**: An integer in the range 1980–2079. **Errors**: **rangecheck**

10.5.3 File System Parameter Sets (Type = /FileSystem)

There are four commonly used parameter sets of **Type** /FileSystem; they are %disk%, %cartridge% or %rom%, and %ram%. The parameter set %rom% is used to denote a cartridge device that is nonremovable and nonwriteable. %ram% is used to denote a file system that is writeable and stored in some form of RAM. The memory used by %ram% competes with other uses of

memory, such as PostScript VM, and explicit use of %ram% to store files competes with other uses of %ram%, such as reusable stream filters. As with other devices, %ram% is not necessarily present in all interpreters. Display PostScript products and UNIX-based products have a separate device parameter set, %os%. This parameter set is not present in printer systems. Most of the %os% parameter set values are constant, read only; a minimal set of parameters is provided primarily for consistency with other types of file systems.

Multiple instances of the /FileSystem parameter sets use digit suffixes. Digit suffixes are generally used to distinguish between multiple instances of storage devices, and only a disk device generally uses the suffix 0. Letter suffixes are generally used only to distinguish between multiple instances of nonstorage devices.

Required entries in these parameter sets are summarized below; detailed descriptions of the entries are given in the referenced tables.

%disk%
(Table 10.31 on page 330)

BlockSize	**Interleave**	**Removable**
Bus	**LogicalSize**	**Searchable**
Free	**Mounted**	**SearchOrder**
HasNames	**PhysicalSize**	**Type**
InitializeAction	**PrepareAction**	**Writeable**

%cartridge% or %rom%
(Table 10.32 on page 334)

BlockSize	**InitializeAction**	**Searchable**
CartridgeID	**LogicalSize**	**SearchOrder**
CartridgeType	**Mounted**	**Type**
Free	**PhysicalSize**	**Writeable**
HasNames	**Removable**	

%ram%
(Table 10.33 on page 337)

BlockSize	**LogicalSize**	**Searchable**
Free	**Mounted**	**SearchOrder**
HasNames	**PhysicalSize**	**Type**
InitializeAction	**Removable**	**Writeable**

%os%
(Table 10.34 on page 339)

BlockSize	**LogicalSize**	**SearchOrder**
Free	**Mounted**	**Type**
HasNames	**Removable**	**Writeable**
InitializeAction	**Searchable**	

Note *In the following tables, "read-only" refers to access by language operators (for example, **setdevparams** and **currentdevparams**). The value of a read-only parameter can change, but not as the result of invoking the **setdevparams** operator. Changes to parameters of type /FileSystem are effective immediately.*

Also in the following tables, a page is defined as a unit of storage whose size is file system dependent.

Table 10.31 *Entries in the %disk0%, %disk1%, %disk2%, … parameter sets*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only)* Indicates the disk or cartridge formatting size of a page (for the logical and physical size of the media). The formatting size of a page for a cartridge is 1 byte per block. The formatting size of a page for a disk using the Adobe file system is 1024 bytes per block. **Legal values:** A positive integer, typically 1024. **Errors:** None
Bus	string	(*Read-only*) With the Adobe storage device implementation, the value of **Bus** will indicate the name of the bus on which this disk resides. This string can be used as input to the **setdevparams** or **currentdevparams** operators to get bus parameters. **Legal values:** %Scsi%, %ScsiB%, %ScsiC%, %Ide%, %IdeB%, %IdeC%, etc. **Errors:** None
Free	integer	*(Read-only)* Indicates the amount of free space available on the media for the device in pages, where the page size is indicated by the parameter **BlockSize**. This parameter is valid only if the device is mounted (that is, the **Mounted** parameter is set to *true*). A value of 0 indicates that either the device is not mounted or the media is completely full. **Legal values:** 0 or any positive integer. **Errors:** None
HasNames	boolean	*(Read-only)* Indicates whether the device represented by the parameter set supports named files. If *false*, the device is not mounted. **Legal values:** *true*, *false* **Errors:** None
InitializeAction	integer	Specifies an action for initializing the device. The following are valid values for disks: 0 — Indicates no action. The value returned when the parameter is read.

Table 10.31 *Entries in the %disk0%, %disk1%, %disk2%, … parameter sets (continued)*

Key	*Type*	*Semantics*	
		1	Indicates that the current file system (if any) is to be deleted and a new one of size **LogicalSize** created (the media is assumed to have been formatted already). The device must first be mounted; otherwise, an **ioerror** will result. For further information, see the description of the **LogicalSize** parameter.
		2	Reformats the entire media before creating a new file system of size **LogicalSize**. The **Interleave** parameter also plays a role in how the media is to be formatted. See the description for the **Interleave** below for details.
		3 or more	Has the same effect as the value 2 and also carries out product-dependent actions, which typically consist of reformatting the disk and running integrity tests before creating the file system. Some devices can have additional parameters that serve as arguments to **InitializeAction**.
		Legal values:	Any positive integer or 0.
		Errors:	**ioerror**
Interleave	integer	Arranges logically contiguous sectors on the disk in a way that is most efficient for the system using that disk. This parameter is used only when the media is being formatted. See **InitializeAction** above. For example, assume there are 16 sectors going around a single track on a disk. If the first sector has a logical number of 1, the second 2, the third 3, and so on, the value of **Interleave** is 1 (1-to-1 interleaving). In this case, the system must be very fast in order to be able to take data from the disk, one sector immediately after another. If the system fails to consume the first sector in time for the second sector, the system has to wait an entire revolution of the disk to get the next sector. This can result in very poor performance. If the first sector has a logical number of 1, the third has a logical number of 2, the fifth has a logical number of 3, and so on, the system will need to be able to consume the current sector while the head skips over a sector in time for the next logical sector. In this case, the value of **Interleave** is 2 (2-to-1 interleave). The sectors in between are used for higher logical numbers, and it takes a minimum of two revolutions to get the data for an entire track off the disk. In this example, the second physical sector on the disk would be between logical sectors 1 and 2, and it would be logical sector 9. Similarly, with an **Interleave** value of 3 (3-to-1 interleaving), the first sector has a logical number of 1, the fourth has a logical number of 2, and so on.	

Table 10.31 *Entries in the %disk0%, %disk1%, %disk2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Normally, **Interleave** should be set to a value that allows the software to use the information during the time between sectors, but not waste any time. It is difficult to determine what the proper value is, and the value is highly dependent on the job accessing the disk. For drives that provide buffering for a full track of data, 1-to-1 interleaving is almost always most efficient. **Legal values:** Any positive integer. The legality of the value is disk dependent. **Errors:** **ioerror**
LogicalSize	integer	When set, specifies the size of the file system to be created and used as an argument to the action carried out by **InitializeAction**. If **LogicalSize** is 0, **InitializeAction** uses a default size that is normally the size of the entire media within the device. For further information, see the description of the **InitializeAction** parameter. When queried, this parameter indicates the current size of the file system on the device in pages, where the page size is indicated by the parameter **BlockSize**. A value of 0 indicates that the device is not mounted. If **LogicalSize** is set with a certain value and then the device is reformatted, a query of **LogicalSize** should return the value that was set. However, if the parameter is queried at any time before the media within the device is reformatted, it may return the current size rather than the value that was set. **Legal values:** Any positive integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**. **Errors:** **rangecheck, typecheck**
Mounted	boolean	If set to *true*, the system attempts to mount the device. If set to *false*, the system attempts to dismount the device. Mounting a device makes it known to the system and makes it at least readable, depending on the nature of the device. A device will not mount successfully if it does not contain a valid file system. When queried, the return value indicates whether the device is currently mounted. Obtain the result of an attempted mount by querying **Mounted** immediately after it is set. The **Mounted** parameter raises a **configurationerror** if it is set to *true* and mounting fails; it also raises a **configurationerror** if it is set to *false* and dismounting fails. **Legal values:** *true, false* **Errors:** **configurationerror, typecheck, ioerror**

Table 10.31 *Entries in the %disk0%, %disk1%, %disk2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
PhysicalSize	integer	*(Read-only)* Indicates the size of the media in pages, where the page size is indicated by the parameter **BlockSize**. This parameter is valid only when the device is mounted (that is, **Mounted** is set to *true*). A value of 0 indicates that the device is not mounted. **Legal values:** Any positive integer or 0. **Errors:** None
PrepareAction	integer	Specifies an action to prepare the underlying file system for a specific purpose. Valid values are: 0 — Indicates no action is to be performed (no-op). 1 — Causes a product-specific action to load system files. In one case, these files support older versions of Adobe Japanese typefaces. On a writeable file system, these system files enable older versions of the Japanese Font Downloader utility to work correctly. *Note* *If the values of **InitializeAction** and **PrepareAction** are set in the same invocation of the **setdevparams** operator, the actions performed by **InitializeAction** precede those performed by **PrepareAction**.* **Legal values:** 0, 1 **Errors:** **rangecheck, ioerror**
Removable	boolean	*(Read-only)* Indicates whether the device supports removable media. Depending on how the removable media device operates, setting the value of **Mounted** to *false* will either eject the media or allow its removal. When the media has been removed, it cannot be mounted again until it is re-inserted. **Legal values:** *true*, *false* **Errors:** None
Searchable	boolean	Indicates whether the device participates in searches in file system operations that have specified a file name without specifying a device. See Section 3.8.2 in the *PostScript Language Reference Manual, Second Edition*, for further information. *Note* *Devices that support removable media (on some products) will initially have the value of **Searchable** set to false. The value of **Searchable** must be explicitly set to true to have the media be searched.* **Legal values:** *true*, *false* **Errors:** None

Table 10.31 *Entries in the %disk0%, %disk1%, %disk2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
SearchOrder	integer	Indicates the priority at which the device participates when searching for a file in operations where no device has been specified. The lower the integer, the higher the priority. This parameter is ignored if the **Searchable** parameter is *false*. **Legal values:** Any positive integer or 0. **Errors:** None
Type	name	*(Read-only)* Designates the general category of parameters in a device parameter set. Must have a value of /FileSystem. **Legal values:** /FileSystem **Errors:** None
Writeable	boolean	*(Writeable, but only during a mount)* Indicates whether the files on the device can be open for write access. This parameter can be set to *true* or *false* only during a mount (that is, when the value of **Mounted** is being set to *true* in a call to the **setdevparams** operator) and only if the media is not write protected. If the media is already write protected, this parameter is a constant equal to *false*. When the device is not mounted, this parameter indicates whether or not the drive will support writeable media. **Legal values:** *true*, *false* **Errors:** None

Table 10.32 *Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets*

Key	*Type*	*Semantics*
BlockSize	integer	(*Read-only*) Indicates the cartridge (or rom) formatting size of a page (for the logical and physical size of the media). The formatting size of a page for a cartridge is 1 byte per block. **Legal values:** Any non-zero positive integer (typically 1). **Errors:** None
CartridgeID	integer	*(Read-only)* Indicates an ID that uniquely identifies this cartridge on a product. **CartridgeID** is used by the interpreter to determine if a cartridge has been removed from a slot and a different cartridge inserted. **Legal values:** Any integer. **Errors:** None
CartridgeType	integer	*(Read-only)* Indicates the category classification of the cartridge. This classification is a registry maintained by Adobe.

Table 10.32 *Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Legal values: Any integer.
		Errors: None
Free	integer	*(Read-only*) Indicates the amount of free space available on the media for the device in pages, where the page size is indicated by the parameter **BlockSize**. This parameter is valid only if the device is mounted (that is, **Mounted** is set to *true*). A value of 0 indicates that either the device is not mounted, or the media is completely full.
		Legal values: 0 or any positive integer.
		Errors: None
HasNames	boolean	*(Read-only*) Indicates whether the device represented by the parameter set supports named files. If *false*, the device is not mounted.
		Legal values: *true, false*
		Errors: None
InitializeAction	integer	Specifies an action for initializing the device. The following are valid values for writeable cartridges (setting a value for **InitializeAction** for a *read-only* cartridge has no effect):
		0 — Indicates no action and is the value returned when the parameter is read.
		1 — Reformats the entire media and then creates a new file system using the full size of the cartridge.
		Legal values: 0, 1
		Errors: **rangecheck**, **typecheck**
LogicalSize	integer	When set, specifies the size of the file system to be created and used as an argument to the action carried out by **InitializeAction**. If the value of **LogicalSize** is 0, **InitializeAction** uses a default size that is normally the size of the entire media within the device. For further information, see the descriptions of **LogicalSize** in Table 10.31 on page 330 and **InitializeAction**.
		Legal values: Any positive integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.
		Errors: **rangecheck, typecheck**

Table 10.32 *Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Mounted	boolean	If set to *true*, the system attempts to mount the device. If set to *false*, the system attempts to dismount the device. Mounting a device makes it known to the system and makes it at least readable, depending on the nature of the device. A device will not mount successfully if it does not contain a valid file system. For further information, see the description of **Mounted** in Table 10.31 on page 330. **Legal values:** *true*, *false* **Errors:** **configurationerror, typecheck**
PhysicalSize	integer	*(Read-only*) Indicates the size of the media in pages, where the page size is indicated by the parameter **BlockSize**. This parameter is valid only when the device is mounted (that is, **Mounted** is set to *true*). A value of 0 indicates that the device is not mounted. **Legal values:** Any positive integer or 0. **Errors:** None
Removable	boolean	*(Read-only*) Indicates whether the device supports removable media. Depending on how the removable media device operates, setting **Mounted** to *false* will either eject the media or allow its removal. When the media has been removed, it cannot be mounted again until it is re-inserted. **Legal values:** *true*, *false* **Errors:** None
Searchable	boolean	Indicates whether the device participates in searches in file system operations that have specified a file name without specifying a device. See Section 3.8.2 in the *PostScript Language Reference Manual*, *Second Edition*, for further information. *Note* *Devices that support removable media (on some products) will initially have the parameter* ***Searchable*** *set to false.* ***Searchable*** *must be explicitly set to true to have the media be searched.* **Legal values:** *true*, *false* **Errors:** None
SearchOrder	integer	Indicates the priority at which the device participates when searching for a file in operations where no device has been specified. The lower the integer, the higher the priority. This parameter is ignored if the **Searchable** parameter is *false*. **Legal values:** Any positive integer or 0. **Errors:** None

Table 10.32 *Entries in the %cartridge%, %cartridge1%, %cartridge2%, … and %rom%, %rom1%, %rom2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Type	name	*(Read-only)* Designates the general category of parameters in a device parameter set. Must have a value of /FileSystem. **Legal values:** /FileSystem **Errors:** None
Writeable	boolean	*(Writeable, but only during a mount)* Indicates whether the files on the device can be open for write access. This parameter can be set to *true* or *false* only during a mount (that is, when **Mounted** is being set to *true* in a call to the **setdevparams** operator) and only if the media is not write protected. If the media is already write protected, this parameter is a constant equal to *false*. When the device is not mounted, this parameter indicates whether or not the drive will support writeable media. **Legal values:** *true*, *false* **Errors:** None

Table 10.33 *Entries in the %ram%, %ram1%, %ram2%, … parameter sets*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only)* Indicates the disk formatting size for the logical and physical size of the media. The formatting size for %ram% is 1 byte per block. **Legal values:** 1 **Errors:** None
Free	integer	*(Read-only)* Indicates the amount of free space available on the media for the device. A value of 0 indicates that the device is completely full. **Legal values:** Any positive integer or 0, up to the capacity of the media. **Errors:** None
HasNames	boolean	*(Read-only)* Indicates whether the device represented by the parameter set supports named files. Always set to *true*. **Legal values:** *true* **Errors:** None
InitializeAction	integer	Specifies an action for initializing the device. The only valid values are: 0 — Indicates no action and is the value returned when the parameter is read. 1 — Allows the value of **LogicalSize** to be changed. **Legal values:** 0, 1

Table 10.33 *Entries in the %ram%, %ram1%, %ram2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
		Errors: **rangecheck, typecheck**
LogicalSize	integer	The setting of **LogicalSize** when **InitializeAction** is processed designates the amount of memory the user wants to allocate to the %ram% file system. Actual allocation may be less and may not exceed the value of the **PhysicalSize** parameter. For further information, see the descriptions of **LogicalSize** in Table 10.31 on page 330 and **InitializeAction**.
		Legal values: Any non-negative integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.
		Errors: **rangecheck, typecheck**
Mounted	boolean	(*Read-only*) Always set to *true*. For further information, see the description of **Mounted** in Table 10.31 on page 330.
		Legal values: *true*
		Errors: None
PhysicalSize	integer	*(Read-only*) The value of **PhysicalSize** is set to the maximum allowable size of the %ram% file system. The user may designate a smaller allocation using the **LogicalSize** parameter. For further information, see **PhysicalSize** in Table 10.31 on page 330.
		Legal values: Any positive integer or 0.
		Errors: None
Removable	boolean	*(Read-only*) Indicates whether the device supports removable media. Always set to *false*.
		Legal values: *false*
		Errors: None
Searchable	boolean	*(Read-only*) Indicates whether the device participates in searches in file system operations that have specified a file name without specifying a device. See Section 3.8.2 in the *PostScript Language Reference Manual, Second Edition,* for further information.
		Legal values: *false*
		Errors: None
SearchOrder	integer	*(Read-only*) Indicates the priority at which the device participates when searching for a file in operations where no device has been specified. The lower the integer, the higher the priority. This parameter is ignored if the **Searchable** parameter is *false*.
		Legal values: Any positive integer or 0.
		Errors: None

Table 10.33 *Entries in the %ram%, %ram1%, %ram2%, … parameter sets (continued)*

Key	*Type*	*Semantics*
Type	name	*(Read-only*) Designates the general category of parameters in a device parameter set. Must have a value of /FileSystem. **Legal values:** /FileSystem **Errors:** None
Writeable	boolean	*(Read-only*) Indicates whether the files on the device can be open for write access. Always set to *true*. **Legal values:** *true* **Errors:** None

Table 10.34 *Entries in the %os%, %osB%, %osC%, … parameter sets*

Key	*Type*	*Semantics*
BlockSize	integer	*(Read-only*) Indicates the size of a page (for the size of the partition dedicated to PostScript). **BlockSize** is set by the environment, not PostScript. **Legal values:** A positive integer, typically 1024. **Errors:** None
Free	integer	*(Read-only*) Indicates the amount of free space (in pages) available in the PostScript partition, where the page size is indicated by the parameter **BlockSize**. **Legal values:** Any positive integer or 0. **Errors:** None
HasNames	boolean	*(Read-only*) Indicates whether the device represented by the parameter set supports named files. Always returns a value of *true*. **Legal values**: *true* **Errors**: None
InitializeAction	integer	*(Read-only)* Specifies an action for initializing the device. The values must be 0. A value of zero indicates no action. It is the value returned when the parameter is read. **Legal values**: 0 **Errors**: None
LogicalSize	integer	**Legal values:** Any non-negative integer or 0. The value of **LogicalSize** must be less than or equal to the value of **PhysicalSize**.

Table 10.34 *Entries in the %os%, %osB%, %osC%, … parameter sets* *(continued)*

Key	*Type*	*Semantics*
		Errors: **rangecheck, typecheck**
Mounted	boolean	(*Read-only*) Always returns a value of *true*. For the general definition of **Mounted**, see Table 10.31 on page 330. **Legal values**: *true* **Errors**: **typecheck**
PhysicalSize	integer	*(Read-only*) Indicates the size of the partition (in pages) dedicated to PostScript, where the page size is indicated by the parameter **BlockSize**. **Legal values:** Any positive integer or 0. **Errors:** None
Removable	boolean	(*Read-only*) Always returns a value of *false*. For the general definition of **Removable**, see Table 10.31 on page 330. **Legal values**: *false* **Errors**: None
Searchable	boolean	(*Read-only*) The value of **Searchable** is product-dependent and may be either *true* or *false*. For the general definition of **Searchable**, see Table 10.31 on page 330. **Legal values**: *true, false* **Errors**: None
SearchOrder	integer	*Initially* returns a value of 2. For the general definition of **SearchOrder**, see Table 10.31 on page 330. **Legal values**: Any positive integer or 0. **Errors**: None
Type	name	*(Read-only*) Designates the general category of parameters in a device parameter set. Must have a value of /FileSystem. **Legal values:** /FileSystem **Errors:** None
Writeable	boolean	(*Read-only*) Always returns a value of *true*. For the general definition of **Writeable**, see Table 10.31 on page 330. **Legal values:** *true* **Errors:** None

APPENDIX A

Acronyms

The following acronyms are used in this document:

ACK	Acknowledgment
ASN.1	Abstract Syntax Notation 1
ASCII	American Standard Code for Information Interchange
BOOTP	Bootstrap protocol
BSD	Berkeley Software Distribution
CCITT	Comité Consultatif International de Télégraphique et Téléphonique (International Telegraph and Telephone Consultative Committee)
CFF	Compact font format
CID	Character identifier
CIE	Commission Internationale de l'Éclairage
CMY	Cyan-magenta-yellow
CMYK	Cyan-magenta-yellow-black
CRD	Color rendering dictionary
CSA	Color space array
CTM	Current transformation matrix
DARPA	Defense Advanced Research Project Agency
DDC	Device-dependent color space
DIC	Device-independent color space
DSR	Data set ready
DTMF	Dual tone multifrequency (touch-tone dialing)
DTR	Data terminal ready
ECM	Error correction mode
ECS	Extended character set

EOD	End of data
EOF	End of file
ELAP	EtherTalk link access protocol
EPS	Encapsulated PostScript
ETX	End-of-text
GMT	Greenwich mean time
HMI	Horizontal motion index
ICC	International Color Consortium
IDE	Integrated development environment
IDP	Internetwork datagram protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet protocol
IPU	Inches per unit
IPX	Internetwork packet exchange
ISO	International Standards Organization
ISV	Independent software vendor
ITU	International Telecommunications Union
JISC	Japanese Industrial Standards Committee
JPEG	Joint Photographic Expert Group
LAP	LocalTalk link access protocol
LAT	Local area transport
LPT	Parallel interface of a personal computer
LZW	Lempel-Ziv-Welch
MAC	Media access control
MIB	Management information database
MMR	Modified modified read
MR	Modified read
OEM	Original equipment manufacturer
OSI	Open systems interconnection
PBX	Private branch exchange
PC	Personal computer
PCL	Page control language

PDL	Page description language
PJL	Printer job language
PNG	Portable Network Graphics
PPD	PostScript printer description
RAM	Random access memory
RARP	Reverse address resolution protocol
RGB	Red-green-blue
RIF	Ring identification
RIP	Raster image processor
RIP	Routing information protocol
ROM	Read-only memory
SA	Service advertisement
SCC	Serial communications controller
SCSI	Small computer system interface
SDK	Software development kit
SNMP	Simple network management protocol
SPX	Sequenced packet exchange (Novell transport protocol)
SWOP	Specifications for web offset publications
TCP	Transmission control protocol
TLAP	TokenTalk link access protocol
UDP	User datagram protocol
VM	Virtual memory
W3C	World Wide Web Consortium
XNS	Xerox Network System

APPENDIX B

Implementation Limits

This appendix supplements Appendix B in the *PostScript Language Reference Manual, Second Edition.* It provides information about limits that are typical of PostScript interpreter implementations from Adobe Systems.

Separations Limit

Only 250 separations are allowed. This limitation applies to **SeparationColorNames** in the page device dictionary and the number of colorants specified in **DeviceN** color space.

APPENDIX C

Emulators

This chapter provides information about the emulator parameter sets that allow PostScript printer systems to emulate other printer systems.

C.1 Emulator Parameter Sets (Type = /Emulator)

An emulator is an alternative interpreter for the input stream. Some PostScript printer systems have the ability to emulate other printer systems. The **Interpreter** device parameter specifies what rules a printer system will use to interpret the stream of input characters in order to make marks on the page. (The **Interpreter** device parameter is described in Table 10.3, “Entries in all communications parameter sets (Type = /Communications),” on page 269.) If the value of the **Interpreter** parameter is other than /PostScript, the printer system is being asked to emulate the functions of some other printer system.

Consider, for example, the Diablo®630, which is a daisy wheel printer with very limited capabilities for putting marks on a page. The input stream is code for characters; the printer system assumes one character will follow another until a carriage return or line feed is reached.

To emulate a Diablo630 printer, the code

```
(%Serial%) <</Interpreter /Diablo630 /Protocol /Raw>> setdevparams
```

gives Diablo630-like functionality to the serial input channel on a PostScript printer system that has a Diablo630 emulator. This functionality is invoked at the next job boundary.

The LaserJet 4 emulator, the LaserJet III emulator, the LaserJet IIP emulator, the color version of the HP7475A plotter emulator, and the Diablo630 emulator have parameters that allow the user to specify default values. The emulator parameters can be set with the **setdevparams** operator and read with the **currentdevparams** operator. The LaserJet 4 emulator has a device parameter set called %PCL% that breaks the tradition of naming these parameter sets with the emulator name.

The required entries in the parameter sets for the LaserJet 4 emulator, the LaserJet III emulator, the LaserJet IIP emulator, the color version of the HP7475A plotter emulator, and the Diablo630 emulator are summarized below; the details of the required entries are given in the referenced tables.

%PCL% (Table C.1 on page 349)	
MaxPermanentStorage	**Type**

%LaserJetIII% (Table C.2 on page 349)		
Copies	**FontSource**	**PaperSize**
Duplex	**FontSymbolSet**	**TopMargin**
FontFixed	**FontTypeface**	**Type**
FontHeight	**FontWeight**	**VMI**
FontItalic	**Landscape**	**WaitTimeout**
FontNumber	**LineWrap**	
FontPitch	**MaxLJMemory**	

%LaserJetIIP% (Table C.3 on page 353)		
Copies	**FontSymbolSet**	**ManualFeed**
FontFixed	**FontTypeface**	**MaxLJMemory**
FontHeight	**FontWeight**	**Type**
FontItalic	**Landscape**	**WaitTimeout**
FontPitch	**LinesPerInch**	

%HP7475A% (Table C.4 on page 354)	
ColorSetup	**Type**

%Diablo630% (Table C.5 on page 354)		
AutoLF	**ECSDataWidth**	**Type**
BoldFontName	**Pitch**	
ECS	**RegFontName**	

Table C.1 *Entries in the %PCL% parameter set (when used as LaserJet 4 emulator)*

Key	*Type*	*Semantics*
MaxPermanentStorage	integer	Specifies the maximum amount of memory to be dedicated for use by the PCL emulator. **Legal values:** Product-dependent integer. **Errors:** **rangecheck**, **typecheck**
Type	name	*(Read-only)* Designates the general category of parameters in a device parameter set. Must be /Emulator. **Legal values:** /Emulator **Errors:** None

Table C.2 *Entries in the %LaserJetIII% parameter set (LaserJet III emulator)*

Key	*Type*	*Semantics*
Copies	integer	Specifies the default number of copies of a document to be printed.
Duplex	integer	Sets the initial state of duplexing within a PCL job for printer systems that are capable of duplex operation. Language commands within the print stream can override the setting of this parameter. The following are acceptable values for **Duplex**: 0 Simplex 1 Long-edge binding duplex 2 Short-edge binding duplex Default value: 0 (Duplexing is not performed.)
FontFixed	boolean	If *true*, a fixed-pitch font (for example, `Courier`) is requested. If *false*, a proportional-spaced font is requested.
FontHeight	integer	Height of the font, applicable to scalable proportional fonts. This value is a point-size quantity, multiplied by 100 to avoid floating-point representation. Thus, a font that is 8.5 points in height would have the value 850. Note that this value is used only if the font specified by the combination **FontSource** and **FontNumber** is scalable and proportional.
FontItalic	boolean	If *true*, an italic or oblique font is requested.

Table C.2 *Entries in the %LaserJetIII% parameter set (LaserJet III emulator) (continued)*

Key	*Type*	*Semantics*
FontNumber	integer	Selects the default font within the current **FontSource**. Applicable values are determined based upon the **FontSource** and the number of fonts that are available from that font source. Use font numbers printed on the font sample page. If a **FontNumber** is specified that is outside the range, the value 0 is used.
FontPitch	integer	Number of characters-per-inch for monospace scalable fonts. The value is multiplied by 100 to avoid floating-point representations. Thus, to select a 12-pitch font, use the value 1200. The PCL5 interpreter uses this parameter only if the font specified by the combination **FontSource** and **FontNumber** is scalable and monospace.
FontSource	integer	Selects the source of the desired font. 0 Internal font. 1 Downloaded font. –1 Used when the default font is not to be selected. If the –1 value is used, then the default font is selected via an obsolete method which uses the parameters **FontFixed**, **FontItalic**, **FontWeight**, and **FontTypeface**. If the value is not –1, these four parameters are not used to select the default font.
FontSymbolSet	integer	Equivalent of the symbol set code. The applicable values are described in Hewlett–Packard manuals. Note that this value is consulted only if the font specified by the combination **FontSource** and **FontNumber** is an unbound font. There are 35 legal values, as follows: 4 OD (ISO-60 Norweg) 6 OF (ISO-25 French) 7 OG (German) 9 OI (ISO-15 Italian) 11 OK (ISO-14 JISASCII) 14 ON (ECMA-94 Latin 1) 19 OS (ISO-11 Swedish) 21 OU (ISO-6 ASCII) 36 1D (ISO-61 Norweg) 37 1E (ISO-4 UK) 38 1F (ISO-69 French) 39 1G (ISO21 German) 51 1S (Spanish) 53 1U (Legal) 75 2K (ISO -57 Chinese)

Table C.2 *Entries in the %LaserJetIII% parameter set (LaserJet III emulator) (continued)*

Key	*Type*	*Semantics*	
		83	2S (ISO-17 Spanish)
		85	2U (ISO-2 IRV)
		115	3S (ISO-10 Swedish)
		147	4S (ISO-16 Portug)
		173	5M (PS-Math)
		179	5S (ISO-84 Portug)
		202	6J (Microsoft® Pub)
		205	6M (Corel VENTURA™ Math)
		211	6S (ISO-85 Spanish)
		234	7J (Desktop)
		269	8M (Math-8)
		277	8U (Roman-8)
		309	9U (Windows™)
		330	10J (PS-Text)
		341	10U (PC-8 US)
		373	11U (PC-8 DN)
		405	12U (PC-850)
		426	13J (Ventura Intl)
		458	14J (Ventura US)
		501	15U (PiFont)
FontTypeface	integer	Describes the typeface (for example, Times, Helvetica, Palatino). The integer value, which can be up to 16 bits, comes from a table published by Hewlett–Packard.	
FontWeight	integer	This value, between –7 and +7, describes the “weight” or “boldness” of the font. –7 is very light and +7 is very bold.	
Landscape	boolean	If *true*, the default orientation of the page is landscape unless otherwise specified in the PCL description of the page.	
LineWrap	boolean	If *true*, long lines wrap to the next line. If *false*, long lines are truncated.	
MaxLJMemory	integer	Specifies the maximum amount of memory that the emulator will ask for from the page allocator to store downloaded fonts and macros. This limit is important because the emulator will acquire memory at the expense of the PostScript interpreter’s memory needs for VM or the font cache, for example. **MaxLJMemory** is rounded to the nearest multiple of a memory block size (8192 bytes).	

Table C.2 *Entries in the %LaserJetIII% parameter set (LaserJet III emulator) (continued)*

Key	*Type*	*Semantics*
PaperSize	integer	Sets the paper size to be used within the PCL job. This parameter produces results similar to the "paper size command" ([Esc]&l#A) within the PCL5 language. The **PaperSize** parameter can specify any of the supported page sizes available to the LaserJet III printer. In addition, there is a special value, –1, which means "unspecified." This allows the printer system to draw paper from the default slot. The paper sizes available to the LaserJet III printer and their associated integer values are as follows:

Value	*Paper Size*	*Dimensions (in default units)*
–1	Unspecified	Default slot
1	Executive	522 × 756
2	Letter	612 × 792
3	Legal	612 × 1008
26	A4	595 × 842
80	Monarch Envelope	279 × 540
81	Com-10 Envelope	297 × 684
90	International DL Envelope	312 × 624
91	International C5 Envelope	459 × 649

Note *In the above table, a "default unit" is 1/72 of an inch.*

Default value: –1 (Indicates "unspecified," the default tray.)

TopMargin	integer	Amount of white space at the top of the page, specified in IPU (1/7200 inch). Default value: 3600 (1/2 inch)
Type	name	*(Read-only*) Designates the general category of parameters in a device parameter set. Must have the value of /Emulator.
VMI	integer	Specifies the space between lines of text in 1/7200 inch units.
WaitTimeout	integer	Specifies the wait time out in seconds after which a page is ejected. Default value: 30

Table C.3 *Entries in the %LaserJetIIP% parameter set (LaserJet IIP emulator)*

Key	*Type*	*Semantics*
Copies	integer	Specifies the default number of copies of a document to be printed.
FontFixed	boolean	If *true*, a fixed-pitch font is requested. If *false*, a proportional-spaced font is requested.
FontHeight	real	Specifies the desired font height in 1/72 of an inch units.
FontItalic	boolean	If *true*, an italic (or oblique) font is requested.
FontPitch	real	Used only if **FontFixed** is *true*. **FontPitch** takes a real number specifying the number of characters per inch.
FontSymbolSet	integer	Specifies the mapping from 7- or 8-bit numbers to glyphs that appear on the page. The value of this parameter is the number associated with this field in a downloaded font.
FontTypeface	integer	Number assigned to a particular font (for example, Times, Helvetica, Palatino). The integer value, which can be up to 16 bits, comes from a table published by Hewlett–Packard.
FontWeight	integer	Specifies the "weight" or "boldness" of desired font. The parameter ranges from –7 to +7, where –7 is very light and +7 is very bold.
Landscape	boolean	If *true*, the initial orientation of the page is landscape instead of portrait.
LinesPerInch	real	Specifies the default value for the "vertical motion index." This determines the interline spacing (and hence the number of lines on the page).
ManualFeed	boolean	See Section 4.11.3 in the *PostScript Language Reference Manual, Second Edition*.
MaxLJMemory	integer	Allows the user to limit the amount of memory that the LaserJet IIP emulator will use. This limit is important because the emulator will acquire memory at the expense of the PostScript interpreter's memory needs for VM or the font cache, for example. Within a given emulation job, the LaserJet IIP emulator will use temporary memory in excess of **MaxLJMemory** to hold fonts and macros.
Type	name	*(Read-only)* Designates the general category of parameters in a device parameter set. Must have the value of /Emulator.
WaitTimeout	integer	The value of **WaitTimeout** (in seconds) is used by the LaserJet IIP emulator as the minimum amount of time the emulator will wait for additional incoming characters before declaring the job finished. A value of 0 indicates to the emulator that it should wait forever. The parameter typically has a default value of 30.

Table C.4 *Entries in the %HP7475A% (color version) parameter set (HP7475A plotter emulator)*

Key	*Type*	*Semantics*
ColorSetup	string	Allows the user to change the default pen color. The **ColorSetup** parameter is a string that contains a list of numbers. There must be a multiple of five numbers in the string. Each set of five specifies the pen number (integer), the width of the pen's line in millimeters (real), the red color value (real, between 0 and 1.0), the green color value (real, between 0 and 1.0), and the blue color value (real, between 0 and 1.0).
Type	name	(*Read-only*) Designates the general category of parameters in a device parameter set. Must have the value of /Emulator.

Table C.5 *Entries in the %Diablo630% parameter set (Diablo630 emulator)*

Key	*Type*	*Semantics*
AutoLF	boolean	If *true*, automatic line feeding is specified.
BoldFontName	name	Specifies the name of the PostScript language font used for boldface printing when **ECS** is *false*.
ECS	boolean	If *true*, the printer system emulates the IBM PC Graphics ECS (extended character set) print wheel. If *false*, the printer system emulates the 96-character plastic print wheel.
ECSDataWidth	integer	Selects 7- or 8-bit data when **ECS** is *true*.
Pitch	integer	The font width and initial HMI (horizontal motion index) is determined from **Pitch**. **Pitch** can have a value of 10, 12, or 15. Any other value will result in a **rangecheck** error.
RegFontName	name	Specifies the name of the PostScript language font used for regular printing when **ECS** is *false*.
Type	name	(*Read-only*) Designates the general category of parameters in a device parameter set. Must have the value of /Emulator.

APPENDIX D

Updates to the PostScript Language Reference Manual, Second Edition

This appendix supersedes information in the *Adobe Technical Note #5085, 7 July 1995*, produced by Adobe Developer Support.

D.1 Introduction

This appendix describes content changes and additions to the *PostScript Language Reference Manual, Second Edition*. It is intended only as a supplement for those who have the manual.

The changes described here apply to all implementations of LanguageLevel 2. These changes are limited to corrections and clarifications of material in the *PostScript Language Reference Manual, Second Edition*. They do not include any changes to the language that have been introduced either as extensions to LanguageLevel 2 or as part of the definition of LanguageLevel 3.

Section D.2, “Errata for the Fifth and Subsequent Printings,” details the errata for printings of the *PostScript Language Reference Manual, Second Edition,* since the fifth printing. Section D.3, “Changes Made in the Third and Fifth Printings,” details the changes made in the third and fifth printings. No changes were made in any other printings.

D.2 Errata for the Fifth and Subsequent Printings

This section details errata (not yet published) to the fifth and subsequent printings of the *PostScript Language Reference Manual, Second Edition*.

D.2.1 Chapter 3 Errata

The following changes should be made in Chapter 3:

- Page 31, third paragraph. Change "...a prefix indicating that the following name is a literal." to "...a prefix indicating that the following sequence of zero or more regular characters constitutes a literal name object." Following the paragraph, insert:

  Note *The token / (a slash followed by no regular characters) is a valid literal name."*

- Page 33, Table 3.2. The *save* object is composite, not simple.

- Page 41, first bullet. Append the following: "It also contains other definitions, including the dictionaries listed in Section 3.7.5, Tables 3.3 and 3.4, as well as various named constants such as *true* and *false*."

- Page 45, last paragraph. The last sentence, "the integer result 50" should be "the real result 50."

- Page 47, item 5. The last sentence, "the integer object 50," should be "the real object 50."

- Page 51, Section 3.6.1. Add the following bullet:

  - **cleartomark** removes the elements above the highest mark and then removes the mark itself.

- Page 65, near top. Add the following note:

  Note ***systemdict****, a global dictionary, contains several entries whose values are local dictionaries, such as* ***userdict*** *and* ***$error****. This is an exception to the normal rule that prohibits objects in global VM referring to objects in local VM, described in Section 3.7.2.*

- Page 80, first paragraph. The last sentence should be a separate paragraph, preceded by: "For both the %statementedit and %lineedit special files."

- Page 81, item 2, Filter parameters. Append to paragraph: "However, all filters accept an optional parameter dictionary, immediately below the filter name on the operand stack. The presence of this operand is distinguished by type, enabling straightforward extension of a filter's parameters."

- Page 87, second paragraph. Append the following: “In general, PostScript language programs using resources should not depend on knowing anything about the policies used by the resource machinery, since those policies can vary among different resource implementations.”

- Page 90 before Encoding section heading. Insert the following paragraph:

  When **findresource** or **findfont** loads a font from an external source into VM, it may choose to use global VM rather than the current VM allocation mode. This choice depends on memory management algorithms used by the interpreter. It also depends on **FontType**, since certain Type 3 fonts do not work correctly when loaded into global VM. The details of this policy are implementation dependent; a PostScript language program should not depend on knowing what they are.

- Page 92, second paragraph. Change “The **defineresource** operator is ordinarily not allowed” to “The **defineresource** and **undefineresource** operators are ordinarily not allowed.”

- Page 94, Table 3.10, **FindResource**. Append the following sentence: “This procedure determines the policy for using global versus current VM when loading a resource from an external source.”

- Page 95, Table 3.10, **ResourceForAll**. Append the following sentence: “Note that **ResourceForAll** should remove the category implementation dictionary from the dictionary stack before executing the procedure operand of **resourceforall**, and it should begin that dictionary again prior to returning. This ensures that the procedure operand is executed in the dictionary context in effect at the time **resourceforall** was invoked.”

- Page 97, sixth paragraph. Replace the paragraph beginning “If this is successful, …” with:

  If **ResourceFileName** for a particular category and instance name exists and executes without a PostScript error, it will leave on the stack a string. If that category maintains all of its instances as named files, this string is the name of the file for that instance. This file name may or may not contain the %device% prefix. Use of this file name with file operators may not succeed for a variety of reasons, including:

  - The category may not maintain all of its instances as named files.

  - The operator tried to delete a file from a read-only file system.

  - The operator tried to write to a file system with insufficient space.

- Page 120, third paragraph. Change “128 to 159 inclusive,” to “128 to 131 inclusive.”

- Page 130, **LZWDecode** Filter. Change the first two lines to:

  The syntax for using the **LZWDecode** filter is either of the following:

  source /LZWDecode filter

  source dictionary /LZWDecode filter

- Page 130, **LZWEncode** Filter. Change the first two lines to:

  The syntax for using the **LZWEncode** filter is either of the following:

  target /LZWEncode filter

  target dictionary /LZWEncode filter

- Page 132. Prior to paragraph beginning “LZW had been adopted…,” insert the following:

  The PostScript language LZW filters accept optional argument dictionaries that are not documented in the *PostScript Language Reference Manual, Second Edition*. All entries in the **LZWEncode** and **LZWDecode** dictionaries are optional, and have default values. The ones that currently have an effect are:

  - **Predictor** *integer* (default 1). If **Predictor** is 1, normal LZW encoding or decoding is done. If **Predictor** is 2, normal LZW encoding is preceded by differencing the data to be encoded, while normal LZW decoding is followed by a complementary undifferencing operation.

  - **Columns** *integer* (default 1). Only has effect if **Predictor** is 2. **Columns** is the number of samples in a sampled row. The first sample of each row is not differenced; every other one is differenced with the prior sample in its row. Each row begins on a byte boundary. Any extra bits beyond (**Columns** × **Colors** × **BitsPerComponent**) that might be needed to complete a multiple of 8 are not differenced.

  - **Colors** *integer* (default 1). Only has effect if **Predictor** is 2. The number of interleaved colors per sample. Each color is differenced with the same color in the prior pixel. Legal values are 1, 2, 3, or 4.

  - **BitsPerComponent** *integer* (default 8). Only has effect if **Predictor** is 2. The number of bits used to represent each color component of a sample. Legal values are 1, 2, 4, or 8.

  - There is another optional LZW parameter that does not relate to **Predictor**:

EarlyChange *integer* (default 1). The TIFF 5.0 specification can be interpreted to imply that code word length increases are postponed as long as possible. However, the LZW sample code distributed by Aldus increases the code word length one code word earlier than necessary. Both interpretations are implemented in the PostScript interpreter.

If **EarlyChange** is 0, code word length increases are postponed as long as possible. If it is 1, they occur one code word early. A decode filter's **EarlyChange** parameter must match the **EarlyChange** parameter used by the encode filter that generated its input data.

- Page 142, last paragraph. Append: "… , as described in Section 3.13.1 on Pages 123–125. Note that there is no **NullDecode** filter as such, because the **SubFileDecode** filters can be configured to serve that function."

D.2.2 Chapter 4 Errata

The following changes should be made in Chapter 4:

- Pages 187–188, **DecodeABC** and **DecodeLMN** procedures. Add note: These procedures may be called with arguments outside the corresponding ranges specified by **RangeABC** and **RangeLMN**. They should deal with such arguments robustly.

- Page 192, **DecodeA** and **DecodeLMN** procedures. Add note: These procedures may be called with arguments outside the corresponding ranges specified by **RangeA** and **RangeLMN**. They should deal with such arguments robustly.

- Page 196, third paragraph. Change "... it is truncated to an integer" to "... it is rounded to the nearest integer."

- Page 197, after first full paragraph. Insert the following:

  Note *In the remainder of this section, the term "separation" refers to any color component that the device can paint on the current page. The term applies whether the colorants are combined on one piece of media ("composite") or are painted individually on separate pieces of media ("separations").*

- Page 199, fifth paragraph. Change "However, it applies only when separations are being produced, …" to "However, it is related to use of **Separation** color spaces, …"

- Page 200, second paragraph. Change "When the device is not producing separations, …" to "If the device is incapable of overprinting, or if the alternative color space has been selected, …."

- Page 202, **PaintType** 1. Append the following sentence: “When the **PaintProc** begins execution, the current color is the one that was in effect at **makepattern** time.”

- Page 214, last paragraph. Append the following sentence: “If the data sources are strings, all of them must be the same length.”

- Page 222, after first full paragraph. Append new paragraph:

  After having been mapped through **Decode**, color components are rounded to the nearest representable value. For an **Indexed** color space, color components are rounded to the nearest integer.

- Page 222, **Interpolate**, third paragraph. Append the following sentences:

  “The interpolation algorithm is implementation-dependent and not under user control. Interpolation may not always be performed for some classes of images or on some devices.”

- Page 235, **NumCopies**. Change second sentence to: “A *null* value indicates that the interpreter should consult the value of **#copies** in the current dictionary stack each time a page is printed (by **showpage**, **copypage**, or device deactivation). This is compatible with the convention used by LanguageLevel 1 implementations.”

- Page 246, Table 4.14, **PolicyNotFound**, option 1. *(This change supersedes Tech Note 5085, dated 7 July 1995.)* Replace with the following:

  “Disregard this feature request. A subsequent call to the **currentpagedevice** operator will return a dictionary in which the entry for this feature is modified as described below:

  - Replaced by *null* if it is a media selection request

  - Unchanged from its former value (prior to calling the **setpagedevice** operator) if the feature is known to the device

  - Absent if the feature is unknown to the device”

- Page 247, Table 4.14, **PageSize**, option 1. Replace with the following: “Disregard the **PageSize** key in media selection. A subsequent call to the **currentpagedevice** operator will return a dictionary whose **PageSize** key indicates the media that was actually selected.”

- Page 248, Note. Reword as follows: “If a media source or destination has a **MatchAll** attribute of *true*, its attributes will not be matched by media requests that have been ignored.”

D.2.3 Chapter 5 Errata

The following changes should be made in Chapter 5:

- Pages 277–278, Section 5.6.3. When invoking a charstring procedure, the PostScript interpreter does not consult the values of **Metrics**, **Metrics2**, or **CDevProc**.

- Page 280, after first paragraph. Insert the following new paragraph:

  The results of **BuildGlyph** should depend only on the complete transformation from character space to device space, and not on the relative contributions of the **FontMatrix** and the CTM prior to **BuildGlyph**. In particular, **BuildGlyph** should not attempt to vary its results depending on the font dictionary's **FontMatrix**.

- Page 280, third paragraph. Replace by the following:

  The **BuildGlyph** procedure must execute one of the following operators to pass width and bounding box information to the PostScript interpreter. This must precede execution of any path construction or painting operators describing the character.

- Page 285, Section 5.8.2, third paragraph, following "...of any length." Append: "subject to an implementation limit. (See Appendix B.)"

- Page 291, following paragraph in progress at top of page. Add another bulleted paragraph:

  - When a modal font is first encountered, if the next byte of the show string is not an escape code, descendant font 0 of the modal font is chosen and the byte is passed down to that font. This also occurs if an escape is followed by another escape but the currently selected font has no parent.

- Page 291, third bullet. Replace by the following:

  - If the descendant of a non-modal composite font is itself a non-modal composite font, the second part (character code) of the value extracted from the show string is used in place of the first byte that would be extracted by the descendant font's mapping algorithm. This rule is independent of the number of bits actually contained in the code contributed by the parent font.

D.2.4 Chapter 6 Errata

The following changes should be made in Chapter 6:

- Page 303. Append to the note: “If the current color space is **Separation** (having selected its alternative color space) or **Indexed**, these operators are applied to the underlying color space.”

- Page 315, Table 6.3, **Angle**. The angle is measured counterclockwise from the *x* axis in device space. Note that most products have left-handed device spaces. In such products, a counterclockwise angle in device space will correspond to a clockwise angle in default user space and on the physical media.

- Page 318, Section 6.4.6. Replace the last sentence of the second paragraph, “The values are type 1 or type 3 halftone dictionaries, each of which …,” with the following:

  Descendents of a type 5 halftone dictionary are halftone dictionaries for single color components—that is, they are halftone types other than 2, 4, or 5.

D.2.5 Chapter 8 Errata

The following changes should be made in Chapter 8:

- Page 343, last paragraph. After “...and consumes them,” insert the sentence “Then it executes.”

- Page 344, Table 8.1. Insert additional entries:

  key Object of any type except *null*, used as key in dictionary or resource

  obj Object of any type except *mark*

- Page 361, >>. Add **typecheck** to the list of errors.

- Page 365, **arc**. After the third paragraph, insert the following new paragraph:

  If *ang2* is less than *ang1*, *ang2* is first increased by a multiple of 360 so that it becomes greater than or equal to *ang1*. There are no other adjustments; in particular, the angle subtended by the arc is not reduced modulo 360. If the difference *ang2* − *ang1* exceeds 360, the resulting path will trace portions of the circle more than once.

  Add **rangecheck** to error list.

- Page 366, **arcn**. Append to description: “If *ang2* is greater than *ang1*, *ang2* is first decreased by a multiple of 360 so that it becomes less than or equal to *ang1*.”

Add **rangecheck** to error list.

- Page 366, **arct**, figure. Both of the radius lines should be perpendicular to the tangent lines that they meet. The line to (xt_2, yt_2) is noticeably not perpendicular and should be corrected.

  Add **rangecheck** to error list.

- Page 371, **bytesavailable**. Add **invalidaccess** to error list.

- Page 371, **cachestatus**. Append the following paragraph:

  Except for *blimit*, which corresponds to the **MaxFontItem** user parameter, the values returned by **cachestatus** cannot be controlled directly by a PostScript language program. They will vary as a function of the **MaxFontCache** system parameter, but the behavior is implementation dependent.

- Page 379, **copypage**. A page device dictionary's **BeginPage** or **EndPage** procedure may not execute **copypage** (an **undefined** error will occur). Add **undefined** to error list.

- Page 381, **cshow**. Remove **nocurrentpoint** from error list.

- Page 382, **currentcmykcolor**. Insert before last sentence: "If the current color space is **Separation** (having selected its alternative color space) or **Indexed**, the **currentcmykcolor** operator is applied to the underlying color space."

- Page 384, **currentdevparams**. Add **undefined** to error list.

- Page 385, **currentfile**. The returned file has the literal attribute.

- Page 386, **currentgray**. Insert before last sentence: "If the current color space is **Separation** (having selected its alternative color space) or **Indexed**, the **currentgray** operator is applied to the underlying color space."

- Page 387, **currenthsbcolor**. Insert before last sentence: "If the current color space is **Separation** (having selected its alternative color space) or **Indexed**, the **currenthsbcolor** operator is applied to the underlying color space."

- Page 391, **currentrgbcolor**. Insert before last sentence: "If the current color space is **Separation** (having selected its alternative color space) or **Indexed**, the **currentrgbcolor** operator is applied to the underlying color space."

- Page 393, **curveto**. Add **rangecheck** to error list.

- Page 394, **cvi**. Change “it interprets” to “it invokes the equivalent of the **token** operator to interpret.”

- Page 395, **cvr**. Change “it interprets” to “it invokes the equivalent of the **token** operator to interpret.”

- Page 415, **filenameforall**. Append to description of **** special character: Note that **** is treated as an escape character in a string literal. Thus, if *template* is a string literal, one must write **** to represent **** in the resulting string.

- Page 416, **filter**. The **filter** operator disregards the current VM allocation mode. Instead, the returned file object is created in global VM if and only if all the composite objects it retains after filter creation are in global VM. Those objects include:

  - The underlying source or target object (file, procedure, or string);

  - The end-of-data string for **SubFileDecode**;

  - The dictionary operand of **DCTDecode** or **DCTEncode**, which can contain various strings and arrays used during operation of the filter.

    Note that the dictionary operand is not retained by filters other than **DCTDecode** and **DCTEncode**. The dictionary is used only as a container for arguments, which are completely processed by the **filter** operator. Therefore, the VM allocation mode of this dictionary is irrelevant.

- Page 418, **findfont**, third paragraph. Remove the sentence: “Generally, Type 1 fonts are loaded into global VM; fonts of other types are loaded into local VM.”

- Page 419, **findresource**. Replace the second paragraph by the following:

  When **findresource** loads an object into VM, it may use global VM even if the current VM allocation mode is local. In other words, it may set the VM allocation mode to global (*true* **setglobal**) while loading the resource instance and executing **defineresource**. The policy for whether to use global or local VM resides in the **FindResource** procedure for the specific resource category; see Section 3.9.2, “Resource Categories.”

- Page 423, **forall**. Append to second paragraph: Existing entries removed from the dictionary by *proc* will not be enountered later in the enumeration.

- Page 436, **ineofill**. Replace description with the following:

  is similar to **infill**, but the inside of the current path is determined according to the even-odd rule (see Section 4.5, “Painting”). In the second form, the inside of the aperture specified by *userpath* is determined according to the non-zero winding number rule.

- Page 440, **inueofill**. Replace description with the following:

  is similar to **inufill**, but the inside of the path specified by *userpath* or $userpath_2$ is determined according to the even-odd rule (see Section 4.5, “Painting”). In the second form, the inside of the aperture specified by $userpath_1$ is determined according to the non-zero winding number rule.

- Page 447, **kshow**. Append the following sentence to the second paragraph: “When *proc* completes execution, the value of **currentfont** is restored.”

- Page 449, **lineto**. Add **rangecheck** to error list.

- Page 456, **moveto**. Add **rangecheck** to error list.

- Page 457, **noaccess**. Append to second paragraph: Applying **noaccess** to a dictionary whose access is already *read-only* causes an **invalidaccess** error.

- Page 469, **rcurveto**. Add **rangecheck** to error list.

- Page 471, **readstring**. Append the following: “The string operand must have non-zero length, else a **rangecheck** error will occur.”

- Page 478, **resourceforall**. Append to description of **** special character: Note that **** is treated as an escape character in a string literal. Thus, if *template* is a string literal, one must write **** to represent **** in the resulting string.

- Page 482, **reversepath**. Replace the last sentence with the following:

  “The relative order of subpaths within the path is unspecified and unpredictable.”

- Page 482, **rlineto**. Add **rangecheck** to error list.

- Page 483, **rmoveto**. Add **rangecheck** to error list.

- Page 486, **save**. The *save* object is composite and logically belongs to the local VM, regardless of the current VM allocation mode.

- Page 490, **selectfont**, second paragraph, last sentence. Remove “**exch**,” from this sentence (since there is no “**exch**” in the example).

- Page 491, **setbbox**. Add the following material:

  setbbox immediately transforms the corners of the bounding box into device space, just as path construction operators do. It then constructs a bounding box in device space enclosing all four corners and oriented with the device space axes. This result is what a subsequent **pathbbox** will read out (see **pathbbox** operator description). All bounding box checking is done in device space.

- Page 492, **setcachedevice**. Replace last sentence on page by: “If any marks fall outside this bounding box, the results produced will be unpredictable.”

- Page 494, **setcachelimit**. Remove **limitcheck** and **rangecheck** from error list.

- Page 495, **setcacheparams**, sixth paragraph. Remove “(the *bmax* value returned by **cachestatus**).”

  Changing size is allowed only in a system manager job, since it is equivalent to changing the **MaxFontCache** system parameter.

  Add **invalidaccess** to error list and remove **rangecheck**.

- Page 497, **setcolor**. Add the following:

  Note that if the current color space is a **Pattern** color space, the number of operands consumed depends on the **PaintType** of the pattern dictionary, which is the topmost operand. See Section 4.9.3, “Pattern Color Space.”

- Page 500, **setdash**. Add **rangecheck** to error list.

- Page 501, **setdevparams**. Add **undefined** to error list.

- Page 505, **sethalftone**, first paragraph. Replace last sentence with the following:

  “If a required entry is missing, an **undefined** error occurs; if its value is of the wrong type, a **typecheck** error occurs.”

- Page 510, **setobjectformat**. Last paragraph. “seperately” should be “separately.”

- Page 512, **setpagedevice**. After “**setpagedevice** reinitializes everything”, insert “except the current font.”

  A page device dictionary’s **BeginPage** or **EndPage** procedure may not execute **setpagedevice** (an **undefined** error will occur). Add **undefined** to error list.

- Page 519, **setvmthreshold**, second paragraph. The word “threshhold” should be “threshold” and “seperately” should be “separately.”

- Pages 520–521, **showpage**. A page device dictionary’s **BeginPage** or **EndPage** procedure may not execute **showpage** (an **undefined** error will occur). Add **undefined** to error list.

- Page 521, **showpage**. Add **limitcheck** and **undefined** error to error list.

- Page 527, **stopped**. Append the following:

  “When an error occurs, the standard error handler sets **newerror** to true in the **$error** dictionary. (See Section 3.10.2.) When using the **stopped** operator to catch and continue from an error (without invoking the **handleerror** operator), it is prudent to explicitly reset **newerror** to *false* in **$error**. Otherwise, any subsequent execution of **stop** may result in inadvertent reporting of the leftover error. Also, note that the standard error handler sets the VM allocation mode to local.”

- Page 529, **strokepath**. The result is not influenced by the current clipping path or view clip. The rules for degenerate subpaths, discussed under **stroke**, also apply to **strokepath**. That is, if **stroke** would produce no output, **strokepath** will produce an empty subpath.

- Page 532, **token**, last paragraph before example. The text “one of), >,], or }” should be “one of), >, or }.”

- Page 534, **type**. The gstatetype name object should be annotated “(LanguageLevel 2 only)”; and the conditiontype and locktype name objects should be annotated “(DPS only).”

- Page 535, **uappend**. Add the following material:

  The bounding box specified by **setbbox** in the user path applies only to elements of the user path. It does not apply to other elements of the current path constructed either before or after execution of **uappend**.

- Page 541, **upath**. Change “**upath** is equivalent to” to “**upath** is approximately equivalent to.” Also, following the example, insert:

  “If the path ends with a **moveto**, **upath** adjusts the bounding box if necessary to enclose it. (Note that **pathbbox** disregards a trailing **moveto**.)”

D.2.6 Appendix B Errata

- Page 566, Table B.1, Architectural limits. Add entry:

XUID 16 Maximum number of elements in an **XUID** array.

D.2.7 Appendix E Errata

The following changes should be made in Appendix E:

Pages 596–597, Table E.5. Four of the accent characters (acute, macron, dieresis, and cedilla) appear in two places in the **ISOLatin1Encoding** vector. This is correctly indicated for acute, but not for the others. There should be second entries for each of those accents.

The **ISOLatin1Encoding** vector deviates from the ISO 8859-1 standard in one respect: The character at position 140 (octal) is quoteleft, whereas the ISO standard specifies grave. A PostScript language program desiring to conform to the ISO standard should create a modified encoding vector with this entry changed.

It should be noted that the character names **guillemotleft** and **guillemotright** are misspelled. A guillemot is a species of seabird. The correct spelling for these punctuation characters is guillemet. However, the misspelled names are the ones actually used in the fonts and encodings containing these characters.

D.2.8 Appendix G Errata

The following changes should be made in Appendix G:

- Page 632, Section G.4.5, first paragraph. Change “the following five categories” to “the following six categories.”
- Page 659, **%%DocumentNeededResource** heading should be:

  ```
  %%DocumentNeededResource: <resource> | (atend)
  ```

- Page 660, **%%DocumentSuppliedResource** heading should be:

  ```
  %%DocumentSuppliedResource: <resource> | (atend)
  ```

- Page 692, first full paragraph: Response should be “/FontName:Yes” or “/FontName:No”, with no space between colon and following word.

D.2.9 Appendix H Errata

- Pages 714–715, Section H.2.4. Move **setglobal** and **setshared** from the “must not be used” list on page 714 to the “if used properly” list on page 715.

- Page 715, Section H.2.5, second paragraph: “it definitions” should be “its definitions.”

- Page 718, Section H.2.9. Remove LF CR from the list of line terminator sequences.

- Page 734, near bottom. Change “4 4 604 403 rect” to “0 0 604 403 rect.”

- Page 727, Section H.4.1. The sentence, “A file of type EPSF should contain a PICT resource...” should be changed to read, “A file of type EPSF may contain a PICT resource… .” This clarification permits Macintosh EPS files to have type 'EPSF' regardless of whether they contain a PICT resource.

D.2.10 Appendix I Errata

- Page 738, Table I.1. Change **setglobal** and **setshared** entries for EPS file from “No” to “Careful.”

- Page 741, **setglobal**. Change first sentence to “If used improperly, can disrupt page independence and nesting of included documents.” Append paragraph:

  There are proper uses of **setglobal** and global VM that do not violate page independence or EPS embedding. Global VM can be used to hold data that should be unaffected by **save** and **restore** operators occurring within a page. It can also be used for read-only resources that are loaded by the **findresource** operator on one page and are available for access (also by **findresource**) on subsequent pages.

- Page 743, **setshared**. Change entire description to “See **setglobal**.”

D.2.11 Index Errata

The following changes should be made in the Index.

- Page 747, **#copies** should have page references for **NumCopies** page device entry (page 235) and **showpage** operator description (pages 520–521).

- There should be entries for Display PostScript and LanguageLevel 2, referring to the place where these icons are explained (pages 6–7).

- The item JPEG should be indexed (page 137).

- The type names, listed in Table 8.1 on page 344, should be indexed.

D.3 Changes Made in the Third and Fifth Printings

This section details the changes made in the third printing of the *PostScript Language Reference Manual, Second Edition*, published in April 1991 and changes made in the fifth printing of the *PostScript Language Reference Manual, Second Edition*, published in January 1992. This is supplemental information for those who have a printing earlier than the sixth printing.

D.3.1 Chapter 3 Changes

The following changes were made in Chapter 3:

- Page 52. Last bullet item, third sentence, "…between the **[** and the **]** causes one or more objects…" was changed to "…between the **[** and the **]** causes zero or more objects…"

- Page 71. Clarified the **exitserver** description by adding a new paragraph and changing the note. The new paragraph (before the note):

  "In many implementations, successful execution of **exitserver** sends the message %%[exitserver: permanent state may be changed]%% to the standard output file. This message is not generated by **startjob**. It is suppressed if binary is true in the **$error** dictionary. See Section 3.10.2, 'Error handling.'"

  Changed the note to:

  Note *Aside from **exitserver**, the other contents of **serverdict** are not specified as part of the language. In LanguageLevel 2, the effect of executing **exitserver** more than once in the same file is the same as that of executing the equivalent **startjob** sequence multiple times. In LanguageLevel 1, the effect of executing the **exitserver** operator multiple times is undefined and unpredictable.*

- Page 98, first bulleted paragraph. Change the second sentence to read:

  "If the **FontType** is 1, **findresource** executes true **setglobal** prior to executing the font file; otherwise, it leaves the VM allocation mode unchanged."

- Page 136, table. Add the following description for the key:

  DamagedRowsBeforeError integer (*Optional*)
  Affects **CCITTFaxDecode** only. If **DamagedRowsBeforeError** is positive **EndOfLine** is true, and K is non-negative, then up to **DamagedRowsBeforeError** rows of data will be tolerated before an **IOError** is generated. Tolerating a damaged row means locating its end in

the encoded data by searching for an **EndOfLine** pattern, then substituting decoded data from the previous row if the previous row was not damaged, or a white scan line if the previous row was also damaged.
Default value: 0.

D.3.2 Chapter 4 Changes

The following changes were made in Chapter 4:

- Page 168. Paragraph before example 4.3, "…a homogeneous number array" was changed to "…an encoded number string."

- Page 179. Figure 4.6, box labelled "CIE based color rendering dictionary" was changed to have **setcolorrendering** below it. "...depending on type of rendering dictionary" was changed to "...depending on contents of rendering dictionary." At bottom of page, line entering from the left is now labelled "Tint."

- Page 238, **Install,** first paragraph. Changed "…before executing the implicit **initgraphics**" to "…before executing the implicit **erasepage** and **initgraphics**."

- Page 245. Inserted the following note before the last paragraph of Section 4.11.4:

  Note *If an entry in **InputAttributes** or **OutputAttributes** has a null value instead of a dictionary, it indicates an input or output position that is unavailable for use—for instance, no paper tray is installed.*

- Page 251, Section 4.11.6. Removed second bullet "• Perform logical...".

D.3.3 Chapter 5 Changes

The following changes were made in Chapter 5:

- Page 268, Table 5.4. The fifth key down "**Version**" was changed to "**version**" (no initial cap).

- Page 287, Table 5.6, escape mapping entry. Appended the following sentence:

  "A font number equal to the escape code is treated specially; see Section 5.9.3 'Nested Composite Fonts'."

- Page 288, Table 5.6, double escape mapping entry. Changed "...followed by a second escape code..." to "...followed by an escape code...".

- Page 288. Changed the paragraph beginning “The first byte of a **SubsVector**…” to read as follows:

  “The first byte of a **SubsVector** string specifies one fewer than the *code length*—the number of bytes to be extracted from the **show** string for each operation of the mapping algorithm. A value of 0 specifies a code length of one byte, 1 specifies two bytes, and so on. When a character code is longer than one byte, the bytes comprising it are interpreted high-order byte first. The code length cannot exceed the number of bytes representable in an integer. (See Appendix B.)”

D.3.4 Chapter 6 Changes

The following changes were made in Chapter 6:

- Page 317. Added the following after the bulleted items:

  Note *A threshold value of 0 is treated as if it were 1; therefore, a gray value of 0 paints all pixels black, regardless of what is in the threshold array.*

- Page 318, first paragraph of Section 6.4.6. Removed the words “or separations” at the end of the paragraph.

D.3.5 Chapter 7 Changes

The following changes were made in Chapter 7:

- Page 327. Second paragraph, second sentence: “..., all of which are empty.” was changed to “..., all of which have their standard initial contents.”

- Page 335, fifth paragraph, second sentence. Replaced the second sentence with the following, “A context initially has no view clipping path (see **initviewclip** in Chapter 8).”

- Page 341, second paragraph. Replaced first sentence (“An application...”) with:

  “The Display PostScript™ system determines whether to use bitmap widths or scalable widths for a font by checking the **BitmapWidths** entry in the font dictionary.”

- Page 342. At the end, added the following:

“Since font dictionaries are read-only, the usual way to change whether bitmap widths are used for a font and to control their behavior is to create a copy of the font dictionary, modify the copy, and execute a new **definefont**. Example 7.3 creates a copy of the Helvetica* font and adds the **BitmapWidths** key.

Example 7.3

```
/Helvetica findfont
dup length 1 add dict begin
   {1 index /FID ne {def} {pop pop} ifelse} forall
   /BitmapWidths true def
   currentdict
end
/Helvetica-BitmapWidths exch definefont pop
”
```

D.3.6 Chapter 8 Changes

The following changes were made in Chapter 8:

- Page 366, **arct**. Replace the last paragraph with the following:

  “If the two tangent lines are collinear, (xt_1, yt_1) and (xt_2, yt_2) are identical. In this case, the joining arc has length zero and **arct** merely appends a straight line segment from (x_0, y_0) to (x_1, y_1).”

- Page 370, **bind** operator. Changed second paragraph to:

  “For each procedure object in *proc*, **bind** applies itself recursively to that procedure, makes the procedure read-only, and stores it back into *proc*. The **bind** operator applies to both arrays and packed arrays, but it treats their access attributes differently. It will ignore a read-only array; that is, it will neither bind elements of the array nor examine nested procedures. On the other hand, **bind** will operate on a packed array (which is always read-only), disregarding its access attribute. No error occurs in either case.”

- Page 387, **currentgstate**, Errors section. Removed **setgtate** from the list. It should only appear in the “See Also” section.

- Page 391, **currentscreen**. Changed entry per the following:

currentscreen – **currentscreen** *frequency angle proc*
– **currentscreen** *frequency angle halftone*

returns the current halftone screen parameters (*frequency*, *angle*, and *proc*) in the graphics state, assuming the current screen was established by **setscreen**. If **setcolorscreen** was executed, **currentscreen** returns the

parameters for the gray screen. If **sethalftone** was executed, **currentscreen** returns the *frequency*, *angle*, and halftone dictionary. For type 1 halftone dictionaries, the *frequency* and *angle* values are extracted from the halftone dictionary. For all other types, **currentscreen** returns a frequency of 60 and an angle of 0.

See *Technical Note #5119*, "LanguageLevel 2 Compatibility: The **setscreen** and **currentscreen** Operators," for more detailed information.

- Page 400, **defineuserobject**. Second line of PostScript language definition changed to "3 1 roll put".

- Page 437, **initclip**. At the end of the first paragraph, inserted the following:

  "For a display device, the clipping region established by **initclip** is not well defined. Display PostScript applications should not make the assumption that the clipping region corresponds to the window boundary (see **viewclippath).**"

- Page 439, **initviewclip**. Replaced entire description with the following:

  "returns the context to its initial view clipping state, in which no view clipping path exists."

  Also, added **viewclippath** to the See Also section.

- Page 440, **internaldict**, first paragraph, third sentence: "The internal dictionary is in global VM…" changed to "The internal dictionary is in local VM…"

- Page 490, **selectfont**, second paragraph, second sentence: "if key is not present" changed to "if the font is not defined."

- Page 501, **setdevparams**. Inserted the following before first bulleted item:

  - If a parameter name is not known to the implementation, an **undefined** error occurs.

- Page 512,

- **setpagedevice**, Add **invalidaccess** to error list. Added **currentpagedevice** to the "See Also" list (as the first entry).

- Page 514, **setscreen**. Change the definition and the third paragraph as:

setscreen *frequency angle proc* **setscreen** –
frequency angle halftone **setscreen** –

setscreen sets the screens for all four color components (red, green, blue, and gray) to the same value. **setcolorscreen** sets the screens individually. If the topmost operand is a halftone dictionary instead of a procedure, **setscreen** performs the equivalent of sethalftone with the following exceptions. If the halftone dictionary is of type 1, the *frequency* and *angle* operands will be copied into the halftone dictionary overriding the values of the dictionary's **Frequency** and **Angle** keys. If the dictionary is read-only, **setscreen** makes a copy of it before copying the values. If the halftone dictionary is a type other than 1, the *frequency* and *angle* operands are ignored.

- Page 518, **setundercolorremoval**, second paragraph, first sentence. Changed "…from **DeviceRGB** or **DeviceGray** color space…" to "…from **DeviceRGB** color space…"

- Page 518–519, **setuserparams**. Added **invalidaccess** to error list.

- Page 545, **viewclippath**. Replaced the description with the following:

  "replaces the current path by a copy of the current view clipping path. If no view clipping path exists, it replaces the current path by one that exactly corresponds to the bounding rectangle of the imageable area of the output device.

  Example

  ```
  initviewclip viewclippath pathbbox
  ```

  If the current device is a window device, this returns the bounding box of the window."

D.3.7 Appendix C Changes

The following change was made in Appendix C:

- Page 577, second paragraph. **setucacheparams** changed to **setcacheparams**.

D.3.8 Appendix D Changes

The following change was made in Appendix D:

- Page 581, last paragraph, second sentence. Changed entire sentence to read:

"Applications that work with LanguageLevel 1 interpreters, using language features as documented in the first *PostScript Language Reference Manual*, will also work with LanguageLevel 2 interpreters."

D.3.9 Appendix G Changes

The following changes were made in Appendix G:

- Page 669, Example G.3, 14th line. "#copies" changed to "/#copies".
- Page 669, one-line example after first paragraph. Removed "/Staple true".
- Page 682, first paragraph after example. The line "See section, 'Elementary Types,' for..." was changed to "See Section G.4.6, 'Comment Syntax,' for...".

APPENDIX E

Standard Fonts, Character Sets, and Encodings

This appendix supplements Appendix E in the *PostScript Language Reference Manual, Second Edition.* It lists the typical font set available on most Adobe PostScript 3 products, defines a new character in the Symbol font set, and lists the Central and Eastern European (CE) character encodings added to PostScript character sets.

E.1 Typical Font Set Available on Adobe PostScript 3 Products

While there is no standard set of required fonts, Adobe PostScript 3 products typically have the following 136 fonts built into the interpreter. Table E.1 lists the 136 fonts and indicates whether there is a Central and Eastern European encoding for the font. Some printers may only have a subset of these CE fonts.

Table E.1 *Adobe PostScript 3 font set*

PS FontName	*CE FontName*
AlbertusMT	—
AlbertusMT-Italic	—
AlbertusMT-Light	—
AntiqueOlive-Roman	AntiqueOliveCE-Roman
AntiqueOlive-Italic	AntiqueOliveCE-Italic
AntiqueOlive-Bold	AntiqueOliveCE-Bold
AntiqueOlive-Compact	AntiqueOliveCE-Compact
Apple-Chancery	Apple-ChanceryCE
ArialMT	ArialCE
Arial-ItalicMT	ArialCE-Italic
Arial-BoldMT	ArialCE-Bold
Arial-BoldItalicMT	ArialCE-BoldItalic

Table E.1 *Adobe PostScript 3 font set (continued)*

PS FontName	*CE FontName*
AvantGarde-Book	AvantGardeCE-Book
AvantGarde-BookOblique	AvantGardeCE-BookOblique
AvantGarde-Demi	AvantGardeCE-Demi
AvantGarde-DemiOblique	AvantGardeCE-DemiOblique
Bodoni	BodoniCE
Bodoni-Italic	BodoniCE-Italic
Bodoni-Bold	BodoniCE-Bold
Bodoni-BoldItalic	BodoniCE-BoldItalic
Bodoni-Poster	BodoniCE-Poster
Bodoni-PosterCompressed	BodoniCE-PosterCompressed
Bookman-Light	BookmanCE-Light
Bookman-LightItalic	BookmanCE-LightItalic
Bookman-Demi	BookmanCE-Demi
Bookman-DemiItalic	BookmanCE-DemiItalic
Carta	—
Chicago	ChicagoCE
Clarendon	ClarendonCE
Clarendon-Light	ClarendonCE-Light
Clarendon-Bold	ClarendonCE-Bold
CooperBlack	—
CooperBlack-Italic	—
Copperplate-ThirtyTwoBC	—
Copperplate-ThirtyThreeBC	—
Coronet-Regular	CoronetCE-Regular
Courier	CourierCE
Courier-Oblique	CourierCE-Oblique
Courier-Bold	CourierCE-Bold
Courier-BoldOblique	CourierCE-BoldOblique
Eurostile	EurostileCE
Eurostile-Bold	EurostileCE-Bold

Table E.1 *Adobe PostScript 3 font set (continued)*

PS FontName	*CE FontName*
Eurostile-ExtendedTwo	EurostileCE-ExtendedTwo
Eurostile-BoldExtendedTwo	EurostileCE-BoldExtendedTwo
Geneva	GenevaCE
GillSans	GillSansCE-Roman
GillSans-Italic	GillSansCE-Italic
GillSans-Bold	GillSansCE-Bold
GillSans-BoldItalic	GillSansCE-BoldItalic
GillSans-Condensed	GillSansCE-Condensed
GillSans-BoldCondensed	GillSansCE-BoldCondensed
GillSans-Light	GillSansCE-Light
GillSans-LightItalic	GillSansCE-LightItalic
GillSans-ExtraBold	GillSansCE-ExtraBold
Goudy	—
Goudy-Italic	—
Goudy-Bold	—
Goudy-BoldItalic	—
Goudy-ExtraBold	—
Helvetica	HelveticaCE
Helvetica-Oblique	HelveticaCE-Oblique
Helvetica-Bold	HelveticaCE-Bold
Helvetica-BoldOblique	HelveticaCE-BoldOblique
Helvetica-Condensed	HelveticaCE-Cond
Helvetica-Condensed-Oblique	HelveticaCE-CondObl
Helvetica-Condensed-Bold	HelveticaCE-CondBold
Helvetica-Condensed-BoldObl	HelveticaCE-CondBoldObl
Helvetica-Narrow	HelveticaCE-Narrow
Helvetica-Narrow-Oblique	HelveticaCE-NarrowOblique
Helvetica-Narrow-Bold	HelveticaCE-NarrowBold
Helvetica-Narrow-BoldOblique	HelveticaCE-NarrowBoldOblique
HoeflerText-Regular	HoeflerTextCE-Regular

Table E.1 *Adobe PostScript 3 font set (continued)*

PS FontName	*CE FontName*
HoeflerText-Italic	HoeflerTextCE-Italic
HoeflerText-Black	HoeflerTextCE-Black
HoeflerText-BlackItalic	HoeflerTextCE-BlackItalic
HoeflerText-Ornaments	—
JoannaMT	JoannaMTCE
JoannaMT-Italic	JoannaMTCE-Italic
JoannaMT-Bold	JoannaMTCE-Bold
JoannaMT-BoldItalic	JoannaMTCE-BoldItalic
LetterGothic	LetterGothicCE
LetterGothic-Slanted	LetterGothicCE-Slanted
LetterGothic-Bold	LetterGothicCE-Bold
LetterGothic-BoldSlanted	LetterGothicCE-BoldSlanted
LubalinGraph-Book	LubalinGraphCE-Book
LubalinGraph-BookOblique	LubalinGraphCE-BookOblique
LubalinGraph-Demi	LubalinGraphCE-Demi
LubalinGraph-DemiOblique	LubalinGraphCE-DemiOblique
Marigold	—
Monaco	MonacoCE
MonaLisa-Recut	—
NewCenturySchlbk-Roman	NewCenturySchlbkCE-Roman
NewCenturySchlbk-Italic	NewCenturySchlbkCE-Italic
NewCenturySchlbk-Bold	NewCenturySchlbkCE-Bold
NewCenturySchlbk-BoldItalic	NewCenturySchlbkCE-BoldItalic
NewYork	NewYorkCE
Optima	OptimaCE-Roman
Optima-Italic	OptimaCE-Italic
Optima-Bold	OptimaCE-Bold
Optima-BoldItalic	OptimaCE-BoldItalic
Oxford	—
Palatino-Roman	PalatinoCE-Roman

Table E.1 *Adobe PostScript 3 font set (continued)*

PS FontName	*CE FontName*
Palatino-Italic	PalatinoCE-Italic
Palatino-Bold	PalatinoCE-Bold
Palatino-BoldItalic	PalatinoCE-BoldItalic
StempelGaramond-Roman	StempelGaramondCE-Roman
StempelGaramond-Italic	StempelGaramondCE-Italic
StempelGaramond-Bold	StempelGaramondCE-Bold
StempelGaramond-BoldItalic	StempelGaramondCE-BoldItalic
Symbol	—
Tekton	—
Times-Roman	TimesCE-Roman
Times-Italic	TimesCE-Italic
Times-Bold	TimesCE-Bold
Times-BoldItalic	TimesCE-BoldItalic
TimesNewRomanPSMT	TimesNewRomanCE
TimesNewRomanPS-ItalicMT	TimesNewRomanCE-Italic
TimesNewRomanPS-BoldMT	TimesNewRomanCE-Bold
TimesNewRomanPS-BoldItalicMT	TimesNewRomanCE-BoldItalic
Univers	UniversCE-Medium
Univers-Oblique	UniversCE-Oblique
Univers-Bold	UniversCE-Bold
Univers-BoldOblique	UniversCE-BoldOblique
Univers-Light	UniversCE-Light
Univers-LightOblique	UniversCE-LightOblique
Univers-Condensed	UniversCE-Condensed
Univers-CondensedOblique	UniversCE-CondensedOblique
Univers-CondensedBold	UniversCE-CondensedBold
Univers-CondensedBoldOblique	UniversCE-CondensedBoldOblique
Univers-Extended	UniversCE-Extended
Univers-ExtendedObl	UniversCE-ExtendedObl
Univers-BoldExt	UniversCE-BoldExt

Table E.1 *Adobe PostScript 3 font set (continued)*

PS FontName	*CE FontName*
Univers-BoldExtObl	UniversCE-BoldExtObl
Wingdings-Regular	—
ZapfChancery-MediumItalic	ZapfChanceryCE-MediumItalic
ZapfDingbats	—

E.2 New Symbol Font Character

Table E.2 shows a character that has been added to the Symbol font character set shown in the *PostScript Language Reference Manual, Second Edition,* Section E.11 on page 604 and Section E.12 on page 606.

Table E.2 *New Symbol character*

Glyph	*Name*	*Code*
€	Euro	240

E.3 CE Character Encoding Values

Table E.3 lists the CE character encodings that have been added to PostScript character sets. This list provides information similar to that provided in the encoding vector tables shown in the *PostScript Language Reference Manual, Second Edition*, Section E.6 on page 598 and Section E.7 on page 599.

Table E.3 *CE character encodings*

CE encoding	*Glyph*	*Name*
040		space
041	!	exclam
042	“	quotedbl
043	#	numbersign
044	$	dollar
045	%	percent
046	&	ampersand
047	‘	quotesingle
050	(	parenleft
051	)	parenright

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
052	*	asterisk
053	+	plus
054	,	comma
055	-	hyphen
056	.	period
057	/	slash
060	0	zero
061	1	one
062	2	two
063	3	three
064	4	four
065	5	five
066	6	six
067	7	seven
070	8	eight
071	9	nine
072	:	colon
073	;	semicolon
074	<	less
075	=	equal
076	>	greater
077	?	question
100	@	at
101	A	A
102	B	B
103	C	C
104	D	D
105	E	E
106	F	F
107	G	G

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
110	H	H
111	I	I
112	J	J
113	K	K
114	L	L
115	M	M
116	N	N
117	O	O
120	P	P
121	Q	Q
122	R	R
123	S	S
124	T	T
125	U	U
126	V	V
127	W	W
130	X	X
131	Y	Y
132	Z	Z
133	[	bracketleft
134	\	backslash
135	]	bracketright
136	^	asciicircum
137	_	underscore
140	`	grave
141	a	a
142	b	b
143	c	c
144	d	d
145	e	e

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
146	f	f
147	g	g
150	h	h
151	i	i
152	j	j
153	k	k
154	l	l
155	m	m
156	n	n
157	o	o
160	p	p
161	q	q
162	r	r
163	s	s
164	t	t
165	u	u
166	v	v
167	w	w
170	x	x
171	y	y
172	z	z
173	{	braceleft
174	|	bar
175	}	braceright
176	~	asciitilde
202	‚	quotesinglbase
204	„	quotedblbase
205	…	ellipsis
206	†	dagger
207	‡	daggerdbl

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
211	‰	perthousand
212	Š	Scaron
213	‹	guilsinglleft
214	Ś	Sacute
215	Ť	Tcaron
216	Ž	Zcaron
217	Ź	Zacute
221	‘	quoteleft
222	’	quoteright
223	“	quotedblleft
224	”	quotedblright
225	•	bullet
226	–	endash
227	—	emdash
231	™	trademark
232	š	scaron
233	›	guilsinglright
234	ś	sacute
235	ť	tcaron
236	ž	zcaron
237	ź	zacute
240		space
241	ˇ	caron
242	˘	breve
243	Ł	Lslash
244	¤	currency
245	Ą	Aogonek
246	¦	brokenbar
247	§	section
250	¨	dieresis

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
251	©	copyright
252	Ș	Scommaaccent
253	«	guillemotleft
254	¬	logicalnot
255	-	hyphen
256	®	registered
257	Ż	Zdotaccent
260	°	degree
261	±	plusminus
262	˛	ogonek
263	ł	lslash
264	´	acute
265	µ	mu
266	¶	paragraph
267	·	periodcentered
270	¸	cedilla
271	ą	aogonek
272	ș	scommaaccent
273	»	guillemotright
274	Ľ	Lcaron
275	˝	hungarumlaut
276	ľ	lcaron
277	ż	zdotaccent
300	Ŕ	Racute
301	Á	Aacute
302	Â	Acircumflex
303	Ă	Abreve
304	Ä	Adieresis
305	Ĺ	Lacute
306	Ć	Cacute

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
307	Ç	Ccedilla
310	Č	Ccaron
311	É	Eacute
312	Ę	Eogonek
313	Ë	Edieresis
314	Ě	Ecaron
315	Í	Iacute
316	Î	Icircumflex
317	Ď	Dcaron
320	Đ	Dcroat
321	Ń	Nacute
322	Ň	Ncaron
323	Ó	Oacute
324	Ô	Ocircumflex
325	Ő	Ohungarumlaut
326	Ö	Odieresis
327	×	multiply
330	Ř	Rcaron
331	Ů	Uring
332	Ú	Uacute
333	Ű	Uhungarumlaut
334	Ü	Udieresis
335	Ý	Yacute
336	Ţ	Tcommaaccent
337	ß	germandbls
340	ŕ	racute
341	á	aacute
342	â	acircumflex
343	ă	abreve
344	ä	adieresis

Table E.3 *CE character encodings (continued)*

CE encoding	*Glyph*	*Name*
345	Í	lacute
346	ć	cacute
347	ç	ccedilla
350	č	ccaron
351	é	eacute
352	ę	eogonek
353	ë	edieresis
354	ě	ecaron
355	í	iacute
356	î	icircumflex
357	ď	dcaron
360	đ	dcroat
361	ń	nacute
362	ň	ncaron
363	ó	oacute
364	ô	ocircumflex
365	ő	ohungarumlaut
366	ö	odieresis
367	÷	divide
370	ř	rcaron
371	ů	uring
372	ú	uacute
373	ű	uhungarumlaut
374	ü	udieresis
375	ý	yacute
376	ţ	tcommaaccent
377	˙	dotaccent

E.4 CE Characters Not Encoded

This section lists new CE characters that are not encoded.

Table E.4 *CE characters that are not encoded*

Glyph	*Char name*
Ā	Amacron
Δ	Delta
Ė	Edotaccent
Ē	Emacron
Ğ	Gbreve
Ģ	Gcommaaccent
İ	Idotaccent
Ī	Imacron
Į	Iogonek
Ķ	Kcommaaccent
Ļ	Lcommaaccent
Ņ	Ncommaaccent
Ō	Omacron
Ŗ	Rcommaaccent
Ş	Scedilla
Ū	Umacron
Ų	Uogonek
ā	amacron
¸	commaaccent
ė	edotaccent
ē	emacron
ğ	gbreve
ģ	gcommaaccent
≥	greaterequal
ī	imacron
į	iogonek
ķ	kcommaaccent
ļ	lcommaaccent

Table E.4 *CE characters that are not encoded (continued)*

Glyph	*Char name*
≤	lessequal
◊	lozenge
ņ	ncommaaccent
≠	notequal
ō	omacron
∂	partialdiff
√	radical
ŗ	rcommaaccent
ş	scedilla
∑	summation
ū	umacron
ų	uogonek

APPENDIX **F**

System Name Encodings

This appendix supplements Appendix F in the *PostScript Language Reference Manual, Second Edition.* The following names have been added to the system name encodings:

Index	*Name*
478	**CIEBasedDEF**
479	**CIEBasedDEFG**
480	**DeviceN**

APPENDIX G

Bibliography

G.1 Books

PostScript Language Program Design (Addison-Wesley: Reading, MA, 1988) teaches programming principles unique to LanguageLevel 1. This book, which contains many usable samples, is designed for programmers interested in the effective and efficient design of PostScript language programs and printer drivers.

PostScript Language Reference Manual, Second Edition, (Addison-Wesley: Reading, MA, 1990) is the programmer's reference manual for the PostScript language. It describes the syntax and semantics of the language, the imaging model, and the effects of the graphical operators.

G.2 Technical Notes from Adobe Systems Incorporated

Adobe CMap and CID Font Files Specification Version 1.0, Technical Note 5014.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5014.CIDFont_Spec.pdf

Adobe Communications Protocols Specification, Technical Note 5009.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5009.Comm_Spec.pdf

Adobe Type 1 Font Format Supplement, Technical Note 5015.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5015.Type1_Supp.pdf

CID-Keyed Font Technology Overview, Technical Note 5092.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5092.CID_Overview.pdf

Color Separation Conventions for PostScript Language Programs, Technical Note 5044.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5044.ColorSep_Conv.pdf

Compact Font Format Specification, Technical Note 5176.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5176.CFF.pdf

The Type 42 Font Format Specification, Technical Note 5012.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5012.Type42_Spec.pdf

Type 2 Charstring Format, Technical Note 5177.

- http://www.adobe.com/supportservice/devrelations/PDFS/TN/5177.Type2.pdf

G.3 Other

L. Peter Deutsch, *DEFLATE Compressed Data Format Specification* Version 1.3.

- http://www.internic.net/rfc/rfc1951.txt

Index

Symbols

#copies 360, 369, 376
$Blend 129
$error 356
%%DocumentNeededResource 368
%%DocumentSuppliedResource 368
%CommName% 265
%CommName_NV% 266
%CommName_Pending% 267
%fontset% 133
%lineedit 356
%statementedit 356
/FontName 368
>> 362

Numerics

11x17 compatibility object 238
11x17tray compatibility object 233, 239
1-Input stitching function 61

A

a3 compatibility object 238
a3tray compatibility object 214, 239
a4 compatibility object 238, 239
a4small compatibility object 238, 239
a4tray compatibility object 214, 239
AbsoluteColorimetric 161
AccurateScreens 244
accuratescreens compatibility object 213, 214
Acronyms
  definitions of 341–343
Acute 368
addglyph operator 131, 194
Address 301, 319
Angle 362
AntiAlias 79, 121
AppleTalk DDP 308
appletalktype compatibility object 214
AppSocket device parameter set 288–291
arc operator 362
arcn operator 362
arct operator 363, 373
Array construction operators 370
AsyncRead 55
AutoLF 354
Axial shading 81

B

b5 compatibility object 238, 239
b5tray compatibility object 215, 239
Background 78
Baud 274
BBox 79
BeginPage 371
Bind 105
bind operator 45–46, 196, 373
BindDetails 105
Bitmap fonts (Type 32) 129
BitmapFontInit instance 41
BitmapWidths 373
BitsPerComponent 53, 74, 75, 76, 85, 90, 93, 358
BitsPerCoordinate 85, 90, 93
BitsPerFlag 86, 93
BitsPerSample 58
BlackColorLimit 101
BlackDensityLimit 101
BlackWidth 101
Blend 129

BlockSize 330, 334, 337, 339
BoldFontName 354
Booklet 105
BookletDetails 105
BootDelay 320, 322
Bounds 62
Bridging 301, 319
BroadcastAddress 312
BSizeStandard 322
BuildGlyph 143, 361
BuildTime 248
buildtime compatibility object 215
Bus 281, 330
byteorder compatibility object 215
bytesavailable operator 54, 363

C

C0 61
C1 61
cachestatus operator 363
Calendar device parameter set 306, 327–328
Calibrated CMYK color space 65
Calibrated RGB color space 65
Cartridge device parameter set 334–337
CartridgeID 334
CartridgeType 334
CCITTFaxDecode filter 48, 370
CDevProc 140, 361
CE character encoding 377, 382
CE characters, not encoded 390
Cedilla 368
Central and Eastern European characters
  encoded 377
  not encoded 390
Chameleon fonts (Type 14) 132–135
Character encoding values, CE 382
Character mapping 153
Character, undefined
  handling of 154
CharSet 37, 324
Charstring procedure 361
CharStrings 132, 144
CheckParity 274, 320
checkpassword compatibility object 215, 234
checkscreen compatibility object 213, 215, 234
Checksum 311, 316
Chroma-key masking 72
  see also, Image dictionary, type 4
CID fonts 137
CIDCount 140, 144
CIDFont dictionaries 138–144
CIDFont resource 35, 136
CIDFontName 138
CIDFontType 130, 137, 138
CIDFontType 0 139
  dictionary 140
  private dictionary 142
CIDFontType 1 143
  dictionary 143
CIDFontType 2 143
  dictionary 144
CIDInit instance 41
CIDInit procedure set 149
CID-keyed fonts 135–153
CIDMap 144
CIDMapOffset 140
CIDSystemInfo 138, 149
  dictionary 145
CIE 1976 L*u*v color space 65
CIE-based color spaces 65–68
CIEBasedDEF color space 65, 393
CIEBasedDEFG color space 65, 393
  dictionary 67
cleartomark operator 356
cliprestore operator 196
clipsave operator 196
closefile operator 54
CloseSource 47, 56
CloseTarget 47
CMap 153
  dictionary 148
  example 150
CMap mapping 154
CMap resource 35, 148
CMapName 148
CMapVersion 148
CodeMap 149
CollateDetails 105
  dictionary 118
Color conversion
  device-dependent 176
Color cube 186
Color map 185
  animation 188
Color model
  Display PostScript 186
Color operators
  in Display PostScript 187
Color rendering diagram 371
Color rendering mechanism
  in Display PostScript 187
Color space 65–71
  device pixel 185–192
Color specification 371
Color values
  device-dependent 70
ColorantDetails 100, 119
  dictionary 119
ColorantName 120
Colorants
  definition of 69
ColorantType 120
ColorantZoneDetails 101
  dictionary 103
ColorRendering instance 42
ColorRenderingType resource 44
Colors
  custom 187
  LZW filters 358
  overlapping 189–192
Colors 52, 69
ColorSetup 354
ColorSpace 78
  special considerations 79
ColorSpaceFamily resource 43
Columns 48, 52
  LZW filters 358
Communications device
  definition of 21
Communications parameter sets 261, 262–303
  setting up 264
Compact font format (Type 2) 132–135
Compatibility operations 233–240, 376
Compatibility operators and keys 212–233
composefont operator 197
Composite font 361
  extensions 153
  nesting 155
  Type 0 153
CompressImageSource 248
configurationerror 234
ConnectorType 318, 319

Console device parameter set 305, 324–327
Contexts
multiple execution 372
ControlLanguage resource 35
ControlPortNumber 288
Coons patch meshes 90–96
Coords 81, 83
Copies 349, 353
copypage operator 197, 363
Country 37, 325
CRD selection
based on rendering intent 159–163
modification of 162
crdInfoTag 164
CRDs
synchronizing with ICC profiles 163–166
CreationDate 163
cshow operator 198, 363
CurBufferType 248
CurInputDevice 248
CurOutputDevice 249
Current transformation matrix 183–184
currentcmykcolor operator 363
currentdevparams operator 363
currentfile operator 363
currentfont operator 365
currentgray operator 363
currentgstate operator 373
currenthalftone operator 168–174
currenthsbcolor operator 363
currentrgbcolor operator 363
currentscreen operator 373
currentsmoothness operator 199
currenttrapparams operator 42, 199
CurSourceList 249
CurStoredFontCache 249
CurStoredScreenCache 249
curveto operator 364
cvi operator 364
cvr operator 364

D

DamagedRowsBeforeError 370
Darkness 323
DarknessBlack 323
DarknessCyan 323
DarknessMagenta 323
DarknessYellow 323
DataBits 236, 275
in Level 1/Level 2 mapping 236
DataDict 73
dictionary 74
DataPortNumber 288
DataSource 58, 74, 75, 76, 85, 89, 93, 179, 181
Day 327
DCTDecode 364
DCTEncode 364
Decode 58, 74, 75, 76, 86, 90, 93, 360
Decode array
mapping with 59–60
DecodeA 359
DecodeABC 359
DecodeDEF color space 66
DecodeDEFG color space 66, 67
DecodeLMN 359
DecodeParms 55
DefaultColorRendering 160
defaulttimeouts compatibility object 216
DeferredMediaSelection 105
defineresource operator 357
defineuserobject operator 374
DelayedOutputClose 269, 275, 279, 282, 289, 291, 292, 293, 295, 297, 299, 301
Details dictionaries 116
devdismount compatibility object 213, 216
devforall compatibility object 213, 216, 234
devformat compatibility object 213, 217
Device
definition of 21
Device colorant name dictionary 120
Device compatibility operators and keys 212
Device parameter sets 260
Diablo630 354
emulator 347–354
HP7475A 354
LaserJetIII 349
LaserJetIIP 353
PCL 349
Device parameters 259–340
interdependencies 260
Device pixel color space 185–192
color animation 188
custom colors 187
Device-dependent color components 70
Device-dependent color conversion
DeviceN 176
DeviceN 39, 393
color rendering 175–176
DeviceN color space 69, 70
implementation limits 345
DeviceRenderingInfo 106
DeviceRenderingInfo dictionaries 120
devmount compatibility object 213, 217
devstatus compatibility object 213, 217
Diablo630 device parameter set 348, 354
Dieresis 368
Disk device parameter set 330–334
diskonline compatibility object 218
diskstatus compatibility object 218
Domain 57, 58, 61, 81, 82, 83
DoPrintErrors 249
doprinterrors compatibility objects 219
DoStartPage 250
dostartpage compatibility objects 219
dosysstart compatibility object 219
Double escape mapping 371
Drawing function 189–191
Duplex 349
duplexmode compatibility object 213, 219

E

EarlyChange
LZW filters 359
ECS 354
ECSDataWidth 354
Edge flag 86, 94
Effort 51
ELAP 263
emulate compatibility object 220, 234

Emulator parameter sets 261
Emulator resource 43, 44
Emulators 347–354
Enabled 101, 270, 275, 279, 282, 286, 289, 291, 292, 294, 295, 297, 299, 301
Encode 58, 62
Encode filters 47
Encoded number string 371
Encoded system name 393
Encoding 144
EndPage 371
Engine device parameter set 305, 322–324
Envelopes
orientation in user space 123
EnvironmentSave 250
Errata
PostScript Language Reference Manual 355–376
Error behavior
compatibility objects 234
Errors, generated by page device parameters 121–123
Escape mapping 371
EthernetAddress 299, 318
EthernetPhysical device parameter set 305, 318–319
EtherTalk device parameter set 299–301
EtherTalk Link Access Protocol (ELAP) 263
EtherTalkType 299
EtherTalkZone 300
ExitJamRecovery 106
exitserver compatibility object 370
exitserver operator 235, 370
Exponential interpolation function 61
Extend 82, 83

F

FactoryDefaults 250, 259
FatalErrorAddress 251
FDArray 140
dictionary 142
FDBytes 140
File system device
definition of 21
File system parameter sets 261, 328–340
filenameforall operator 364
fileposition operator 55
Filter 55
filter operator 54, 199, 364
Filter resource 43
Filtering 282, 286, 289, 293, 294, 295, 297, 300, 301
Filters 47–56, 356, 358
modifications for color 69
findcolorrendering operator 160–163, 200
findfont operator 357, 364
FindResource 357
findresource operator 357, 364, 370
firstside compatibility object 213, 220, 234
Flate filters 50–51
FlowControl 236, 275
flushfile operator 55
FMapType 208
mapping algorithm 154
resource 44
Fold 106
FoldDetails 106
Font
composite 153, 361
modal 361
Font set, typical
for Adobe PostScript 3 products 377
FontBBox 131, 138
FontFixed 349, 353
FontHeight 349, 353
FontInfo 138
FontItalic 349, 353
FontMatrix 130, 132, 138, 140, 142, 361
FontName 142, 368
FontNumber 350
FontPitch 350, 353
FontResourceDir 251
Fonts
Type 0 153
Type 14 132
Type 2 132
Type 32 129
Type 42 131
fontset device parameter set 133
FontSet resource 36, 133
FontSetInit instance 42
FontSource 350
FontSymbolSet 350, 353
FontType 131, 137, 138, 144, 370
FontType resource 44, 357
FontTypeface 351, 353
FontWeight 351, 353
forall operator 364
FormType resource 44
Free 330, 335, 337, 339
Function 81, 82, 83, 86, 90, 93
dictionary 56–62
Function dictionary
see also, Smooth shadings
Function-based shading 80
FunctionType 56, 57, 58, 61
FunctionType 0 57–60
FunctionType 2 61
FunctionType 3 61
FunctionType resource 43

G

GatewayAddress 312
GDBytes 141, 144
GenericResourceDir 252
GenericResourcePathSep 252
GetHalftoneName operator 162, 167, 168, 169, 172, 173, 174, 200
GetPageDeviceName operator 162, 201
GetSubstituteCRD operator 162, 201
Global VM
referring to objects in local VM 356
Glyph procedures 145
GlyphData 141
GlyphDirectory 141, 144
CIDFontType 0 146
CIDFontType 2 146
memory use 147
glyphshow operator 201
Gouraud-shaded triangle meshes
free-form 85–89
lattice form 89
Gradient fills
see Smooth shadings
Graphics state object 181
Graphics state parameters 161

Gray levels 166
Grayscales
see **MaxSuperScreen** and Halftone dictionaries 19
Groups 295

H

Halftone cells 166, 171
graphic representation 170
Halftone dictionaries 166–174
type 1 through 5 167
type 10 170–172
type 100 174
type 16 172–174
type 6 168
type 9 169
HalftoneMode 245
HalftoneName 101, 167, 168, 169, 172, 173, 174
HalftoneType 168, 169, 172, 173, 174
HalftoneType 5 dictionaries 372
HalftoneType resource 44
Handshake 279
hardwareiomode compatibility object 221
Hard-wired parameter sets 267
HasNames 271, 276, 280, 282, 286, 289, 292, 293, 294, 296, 297, 300, 302, 330, 335, 337, 339
Height 74, 75, 168, 173, 179
Height2 173
HiFi color 68–71
Homogeneous number array 371
HopCount 316
Hour 327
HP7475A device parameter set 348, 354
HWOptions resource 36
HWResolution 39

I

ICC format 160
ICC profiles
synchronizing with CRDs 163–166
Ide device parameter set 305, 322
IdiomRecognition 45, 246
IdiomSet resource 45
IDP 263
Image dictionary
type 2 178–185
type 3 72–75
type 4 75
Image space
transforming to user space 183
ImageInternalTrapping 101
imagemask operator 71
ImageMatrix 74, 75, 180
ImageResolution 102
Imagesetter compatibility operators and keys 212, 213
ImageShift 106
ImageToObjectTrapping 102
ImageTrapPlacement 102
ImageType 72, 73, 74, 75, 179
ImageType resource 44
ImagingBBox 238
Implementation 77
Implementation limits 345
Implicit resources 42–44
Indexed color space 69, 360, 362
ineofill operator 365
initclip operator 374
InitializeAction 330, 335, 337, 339
initializedisk compatibility object 221
InitiatorID 321
initviewclip operator 374
InkParams resource 37
InputAttributes 106, 371
In-RIP trapping 99–103
InsertSheet 107
Install 371
InstalledRam 253
Intent 55
Interleave 331
InterleaveType 72
internaldict operator 374
Internet Protocol (IP) 263
Internetwork Datagram Protocol (IDP) 263
Internetwork Packet Exchange (IPX) 263
Interpolate 74, 75, 76, 360
Interpolation
of sampled functions 58
Interpreter 271, 276, 280, 282, 286, 289, 292, 293, 294, 296, 298, 300, 302
inueofill operator 365
invalidaccess error 123, 234, 235
IODevice resource 43
IP 263
IP device parameter set 304, 312–315
IPAddress 313
IPAddressDynamic 314
IPX 263
IPX device parameter set 305, 316–318
ISOLatin1Encoding 368

J

Job, unencapsulated 235, 258
JobName 246
jobname compatibility object 221
JobTimeout 246, 253
jobtimeout compatibility object 221
Jog 107
JPEG 369

K

KeepaliveTimer 296
key 362
kshow operator 365

L

Laminate 108
Landscape 351, 353
Language 37, 326
LanguageFamily 36, 41
LanguageLevel 36, 41
languagelevel operator 202
LanguageVersion 36, 41
LaserJetIII device parameter set 348, 349
LaserJetIIP device parameter set 348, 353
LAT device parameter set 295–297
Layered windows
and device pixel color space 192
LeadingEdge 108
ledger compatibility object 238
ledgertray compatibility object 222, 239
legal compatibility object 239

legaltray compatibility object 222, 239
letter compatibility object 238
lettersmall compatibility object 238
lettertray compatibility object 222, 239
Level 1/Level 2 compatibility 209–240, 376
LicenseID 253
limitcheck error 123, 238
Limits
  implementation 345
LinesPerInch 353
lineto operator 365
LineWrap 351
Literal name 356
LLAP 263
Local area transport device parameter set 295–297
Localization resource 37
LocalTalk device parameter set 297–299
LocalTalk Link Access Protocol (LLAP) 263
LocalTalkType 298
LogHost 310
LogicalSize 332, 335, 338, 339
LoginPassword 294
LogPriority 310
LowBitFirst 48
LPR device parameter set 286–288
LZW filters 48, 51, 358

M

Macron 368
makepattern operator 360
ManualFeed 353
manualfeed compatibility object 222
ManualFeedTimeout 109
manualfeedtimeout compatibility object 223
ManualSize 39
Mapping
  FMapType 154
  halftone cells 171
  with **Decode** array 59–60
Margins 109
margins compatibility object 223
MaskColor 75
MaskDict 73
  dictionary 73
Masked images 71–76
MatchAll 360
Matrix 81
MaxDisplayAndSourceList 253
MaxHWRenderingBuffer 26, 253
MaxImageBuffer 26, 254
MaxLJMemory 351, 353
MaxPermanentStorage 349
MaxSeparations 109
MaxStoredFontCache 255
MaxStoredScreenCache 255
MaxSuperScreen 166, 246
MediaClass 39, 109
MediaPosition 110
Memory
  GlyphDirectory 147
Metrics 141, 361
Metrics2 141, 361
MinBandBuffers 255
Minute 328
mirrorprint compatibility object 223
Month 328
Mounted 332, 335, 338, 340
moveto operator 365
MulticastTimer 296
Multiple execution contexts 372
Multiple master font dictionaries 129
MultipleDataSources 74, 76
Multitone color 68–71

N

N 61
nAckPulseWidth 280
Name
  binding of 45–46
Name 318, 320
Names
  encoded system 393
Nesting, composite fonts 155
Network communications parameter sets 283–303
Network communications protocols 263
NetworkAddress 317
NetworkMask 314
NeutralDensity 120
newsheet compatibility object 213, 223, 234
noaccess operator 365
Node address 308
NodeID 298, 300, 302
Nonvolatile parameter sets 266
notdef 154
note compatibility object 238
Novell SPX/IPX 308
nStrobeExpectedPulseWidth 280
NullDecode filter 359
NullEncode filter 47
NumCopies 360

O

obj 362
Objects, compatibility
  see Compatibility operators and keys
On 272, 276, 280, 282, 286, 289, 292, 293, 294, 296, 298, 300, 302, 311, 315, 316, 317, 319, 320
One-input stitching function 61
Open Systems Interconnection model 261, 304
Operand and result types 362
Order 58
Ordering 145
Os device parameter set 339–340
OSI model 261, 304
OutputAttributes 371
OutputDevice 25, 111, 238, 281
OutputDevice resource 38
OutputPage 111

P

Page device
  definition of 21
  parameters 104–118
Page duplexing compatibility operators and keys 212
Page size compatibility operators
  in **userdict** 238
PageCount 256, 323
pagecount compatibility object 224
PageDeviceName 25, 111
pagemargin compatibility object 213, 224
PageOffset 111

pageparams compatibility object 213, 224
PageSize 39, 111, 123, 124, 238, 239, 360
PageSize 7 105, 110, 125, 238
pagestackorder compatibility object 224
Painting
operators 131
with a pattern dictionary 98
PaintProc 360
PaintType 141
PaintType 1 360
Paper size operations 238
Paper tray operations 239
PaperSize 352
Parallel device parameter set 279–281
Parameter sets 260
device 259–340
hierarchy 265
nonvolatile, multiple 266
Parameters
emulator 347–354
graphics state 161
Parameters parameter sets 261, 303–328
Parity
in Level 1/Level 2 mapping 236
Parity 236, 277
Password 235
changing persistent values with 234
Pattern dictionary
painting with 98
type 2 77
PatternType 77
PatternType resource 44
PCL device parameter set 348, 349
PDL resource 40
Perceptual 161
Physical 296, 315, 317
PhysicalSize 333, 336, 338, 340
Pitch 354
Pixel
transparent 192
PixelCopy 180, 184
PNG predictor functions 52
Point-to-point communications parameter sets 274
Policies 357
dictionary 124
Policies PageSize 7 105, 110, 125, 238
PolicyNotFound 360
Poll 321, 322
PostRenderingEnhance 112
PostRenderingEnhanceDetails 112
dictionary 118
PostScript interpreter
definition of 21
PostScript Language Reference Manual
errata 355–376
PostScript printer product
definition of 21
Predictor 52
LZW filters 358
PreferredServer 294
PrepareAction 333
PreRenderingEnhance 112
PreRenderingEnhanceDetails 112
PrinterControl 272, 277, 281, 283, 286, 289, 292, 293, 295, 296, 298, 300, 302
PrinterName 256
printername compatibility operator 225
PrintHost 287, 290
PrintQuality 118
PrintServer device parameter set 293–295
Private 142
PrivateHost 308
ProcessColorModel 40, 112, 175–176
processcolors compatibility object 225
ProcSet resource 41
product compatibility object 225
ProofSet 118
Protocol 277, 281

R

Radial shading 83
Ram device parameter set 337–339
RamSize 256
ramsize compatibility operator 225
Range 57, 58, 61
rangecheck error 122, 234
RangeDEF 67
RangeDEFG 67
RangeHIJ 67
RangeHIJK 67
rcurveto operator 365
readstring operator 365
realformat compatibility object 225
ReceiveWindowSize 287, 290, 311, 316
RegFontName 354
Registry 145
RelativeColorimetric 161
RemotePrinter device parameter set 292–293
Removable 333, 336, 338, 340
removeall operator 131, 202
removeglyphs operator 131, 202
Rendering intent 159, 161, 162
CRD selection 159–163
resetfile operator 55
resolution compatibility object 226
ResourceForAll 357
resourceforall operator 357, 365
Resources
implicit 42–44
regular 34–42
restore operator 148
RetransmitLimit 297
RetransmitTimer 297
ReusableStreamDecode filter
see also, Smooth shadings
ReusableStreamDecode filter 53
REValue 118
reversepath operator 365
Revision 256
revision operator 226
rlineto operator 365
rmoveto operator 365
RollFedMedia 25, 113
Rom device parameter set 334–337
Running 328

S

Sampled functions 57–60
Saturation 161
save object 356, 365
save operator 148
SCC compatibility operators 212
SCC operations 235–238

sccbatch compatibility object 212, 226
sccinteractive compatibility object 212, 226, 234
Scsi device parameter set 305, 320–321
ScsiComm device parameter set 281–283
SDBytes 142
Searchable 333, 336, 338, 340
SearchOrder 334, 336, 338, 340
Second 328
Selectable Separations
see **MaxSeparations**, **DeviceN** color model, and **Indexed** color space 19
selectfont operator 365, 374
Selector 36, 41
SendWindowSize 288, 291, 311
Separation
definition of 69, 359
Separation color space 69, 362
SeparationColorNames 69, 100, 113
implementation limit 345
SeparationOrder 114
Separations 114
Sequenced Packet Exchange (SPX) 263
Serial Communications Controller (SCC) operations 235–238
Serial device parameter set 274–279
setaccuratescreens compatibility object 213, 227
setbbox operator 366, 367
setcachedevice operator 366
setcachelimit operator 366
setcacheparams operator 366, 375
setcolor operator 203, 366
setcolorscreen operator 203
setcolorspace operator 203
setdash operator 366
setdefaulttimeouts compatibility object 227
setdevparams operator 235, 366, 374
setdoprinterrors compatibility operator 227
setdostartpage compatibility object 227
setdosysstart compatibility object 227
setduplexmode compatibility object 213, 228
setfileposition operator 55
setglobal operator 368, 369
sethalftone operator 168–174, 204, 366
sethardwareiomode compatibility object 228
setjobtimeout compatibility object 228
setmargins compatibility object 229
setmirrorprint compatibility object 229
setobjectformat operator 366
setpage compatibility object 213, 229
setpagedevice operator 238, 366
setpagemargin compatibility object 213, 229
setpageparams compatibility object 213, 230
setpagestackorder compatibility object 230
setpapertray compatibility object 234
setprintername compatibility object 230
setresolution compatibility object 230
setrgbcolor operator 186
setsccbatch compatibility object 213, 231
setsccinteractive compatibility object 213, 231, 234
setscreen operator 204, 374
setshared operator 368, 369
setsmoothness operator 204
setsoftwareiomode compatibility object 231
setsystemparams operator 235
settrapparams operator 42, 205
settrapzone operator 42, 100, 206
settumble compatibility object 213, 232
setucacheparams operator 375
setundercolorremoval operator 375
setuserdiskpercent compatibility object 232, 234
setuserparams operator 375
setvmthreshold operator 367
sfnts 132, 144
Shading 77
Shading dictionaries 77–98
ShadingType 78, 81, 83, 85, 89, 93
ShadingType 1 dictionary 80
ShadingType 2 dictionary 81
ShadingType 3 dictionary 83
ShadingType 4 dictionary 85–89
ShadingType 5 dictionary 89
ShadingType 6 dictionary 90–96
ShadingType 7 dictionary 96–98
ShadingType resource 44
shfill operator 77, 206
show string parsing 155
showpage operator 367
Signature 115
Size 58
SlidingTrapLimit 102
SlipSheet 115
Smooth shadings 76–99
SNMP device parameter set 303, 308–309
softwareiomode compatibility object 232
Source gstate 181
SourceListBypass 257
Speed 320
SPX 263
SPX device parameter set 304, 316
Staple 115
StapleDetails 115
StartData operator 133, 194, 206
startjob operator 235
StartJobPassword 215, 235, 257, 258, 259
StartupMode 257
statusdict
compatibility objects in 211
StatusPortNumber 291
StepLimit 102, 103
Stitching function
1-input 61
StopBits 236, 278
stopped operator 367
strokepath operator 367
StrokeWidth 142
SubFileDecode filter 48, 359
SubrCount 142
SubrMapOffset 142
SubsVector 372

Supercell 166
Supplement 145
Symbol font
new character 382
SysContact 308
SysLocation 308
Syslog device parameter set 304, 310
SysName 309
System name encodings 393
System parameters 247–258
systemdict
compatibility objects in 212
SystemParamsPassword 215, 235, 258, 259

T

Table 67
TargetID 321
TCP 263
TCP device parameter set 304, 311
TCP/IP 308
Telnet device parameter set 291–292
Tensor product patch meshes 96–98
Threshold arrays 372
Thresholds 169, 172, 174
TIFF predictor functions 52
Tiling 77
in device space 170, 172
TimeToStandby 324
TLAP 263
token operator 367
TokenRingPhysical device parameter set 305, 319–320
TokenTalk device parameter set 301–303
TokenTalk Link Access Protocol (TLAP) 263
TokenTalkType 302
TopMargin 352
Transfer functions 176
TransferFunction 169, 172, 174
Transmission Control Protocol (TCP) 263
TransmitEncapsulation 26, 315, 317
Transparent pixel 192
TrapColorScaling 102, 103
TrapHost 309
TrapParams resource 42
Trapping 100, 115
Trapping instance 42
Trapping parameter dictionary 101
Trapping zones
setting 100
TrappingDetails 100, 116
dictionary 119
TrappingDetailsType 40
TrappingOrder 119
TrapSetName 102
TrapWidth 103
TraySwitch 116
Triangle meshes, Gouraud-shaded 85, 85–89
Trim 116
TrueType fonts (Type 42) 131
tumble compatibility object 232
Type 117, 118, 119, 121, 279, 281, 283, 288, 291, 292, 293, 295, 297, 298, 300, 303, 309, 310, 311, 312, 315, 316, 318, 319, 320, 321, 322, 324, 327, 328, 334, 336, 338, 340, 349, 352, 353, 354
Type 0 fonts 153
Type 10 halftone dictionary 170–172
Type 100 halftone dictionary 174
Type 14 fonts 132
Type 16 halftone dictionary 172–174
Type 2 fonts 132
Type 2 image dictionary 178–185
Type 32 fonts 129
Type 32 operators 131
Type 42 fonts 131
Type 6 halftone dictionary 168
Type 9 halftone dictionary 169
type operator 367
Type, definition of 260
typecheck error 121

U

uappend operator 367
UDP 263
UDP device parameter set 304, 311
UIDOffset 148
Undefined character (notdef) 154
undefined error 122
undefineresource operator 357
Unencapsulated job 235
UnitSize 48
UnpaintedPath 179, 180
upath operator 367
UseCIEColor 116, 158, 176
User Datagram Protocol (UDP) 263
User parameters 243–247
User space
transforming from image space 183
userdict 356
compatibility objects in 212
page size compatibility operators in 238
userdiskpercent compatibility object 233

V

ValidNV 258
ValuesPerColorComponent 121
Version 371
VerticesPerRow 90
View clipping 372
viewclippath operator 374, 375
VMI 352

W

WaitTimeout 26, 247, 258, 352, 353
waittimeout compatibility object 233
WeightVector 129
Width 73, 74, 75, 169, 174, 179
Width2 174
widthshow operator 208
Windows, layered
and device pixel color space 192
WMode 138, 149
Writeable 334, 337, 339, 340

X

Xerox Network System (XNS) protocols 263
XNS 263
XOrigin 179
Xsquare 172
XUID 77, 138, 148, 368

Y

Year 328
YOrigin 179
Ysquare 172

Z

Zone 303