PDF Reference

sixth edition

Adobe® Portable Document Format

Version 1.7

Adobe Systems Incorporated

PDF Reference, sixth edition: Adobe Portable Document Format version 1.7.

Edition 1.0, October 2006

Contents

Preface 23

Chapter 1: Introduction 25

1.1 About This Book 25
1.2 Introduction to PDF 1.7 Features 28
1.3 Related Publications 31
1.4 Intellectual Property 32

Chapter 2: Overview 33

2.1 Imaging Model 34
2.2 Other General Properties 38
2.3 Creating PDF 43
2.4 PDF and the PostScript Language 45

Chapter 3: Syntax 47

3.1 Lexical Conventions 48
3.2 Objects 51
3.3 Filters 65
3.4 File Structure 90
3.5 Encryption 115
3.6 Document Structure 137
3.7 Content Streams and Resources 151
3.8 Common Data Structures 155
3.9 Functions 166
3.10 File Specifications 178

Chapter 4: Graphics 193

4.1 Graphics Objects 194
4.2 Coordinate Systems 199
4.3 Graphics State 210
4.4 Path Construction and Painting 224
4.5 Color Spaces 235
4.6 Patterns 289
4.7 External Objects 332
4.8 Images 334
4.9 Form XObjects 355
4.10 Optional Content 364

Chapter 5: Text 387

5.1 Organization and Use of Fonts 388
5.2 Text State Parameters and Operators 396
5.3 Text Objects 404
5.4 Introduction to Font Data Structures 410
5.5 Simple Fonts 412
5.6 Composite Fonts 433
5.7 Font Descriptors 455
5.8 Embedded Font Programs 465
5.9 Extraction of Text Content 469

Chapter 6: Rendering 477

6.1 CIE-Based Color to Device Color 478
6.2 Conversions among Device Color Spaces 480
6.3 Transfer Functions 484
6.4 Halftones 486
6.5 Scan Conversion Details 508

Chapter 7: Transparency 513

7.1 Overview of Transparency 514
7.2 Basic Compositing Computations 516
7.3 Transparency Groups 530
7.4 Soft Masks 545
7.5 Specifying Transparency in PDF 547
7.6 Color Space and Rendering Issues 561

Chapter 8: Interactive Features 577

8.1 Viewer Preferences 577
8.2 Document-Level Navigation 581
8.3 Page-Level Navigation 594
8.4 Annotations 604
8.5 Actions 647
8.6 Interactive Forms 671
8.7 Digital Signatures 725
8.8 Measurement Properties 744
8.9 Document Requirements 751

Chapter 9: Multimedia Features 755

9.1 Multimedia 755
9.2 Sounds 782
9.3 Movies 784
9.4 Alternate Presentations 786
9.5 3D Artwork 789

Chapter 10: Document Interchange 841

10.1 Procedure Sets 842
10.2 Metadata 843
10.3 File Identifiers 847
10.4 Page-Piece Dictionaries 848
10.5 Marked Content 850
10.6 Logical Structure 855
10.7 Tagged PDF 883
10.8 Accessibility Support 935
10.9 Web Capture 946
10.10 Prepress Support 962

Appendix A: Operator Summary 985

Appendix B: Operators in Type 4 Functions 989

B.1 Arithmetic Operators 989
B.2 Relational, Boolean, and Bitwise Operators 990
B.3 Conditional Operators 990
B.4 Stack Operators 990

Appendix C: Implementation Limits 991

Appendix D: Character Sets and Encodings 995

D.1 Latin Character Set and Encodings 997
D.2 PDFDocEncoding Character Set 1001
D.3 Expert Set and MacExpertEncoding 1010
D.4 Symbol Set and Encoding 1013
D.5 ZapfDingbats Set and Encoding 1016

Appendix E: PDF Name Registry 1019

Appendix F: Linearized PDF 1021

F.1 Background and Assumptions 1022
F.2 Linearized PDF Document Structure 1024
F.3 Hint Tables 1039
F.4 Access Strategies 1051

Appendix G: Example PDF Files 1057

G.1 Minimal PDF File 1057
G.2 Simple Text String Example 1060
G.3 Simple Graphics Example 1062
G.4 Page Tree Example 1065
G.5 Outline Hierarchy Example 1070
G.6 Updating Example 1074
G.7 Structured Elements That Describe Hierarchical Lists 1082

Appendix H: Compatibility and Implementation Notes 1095

H.1 PDF Version Numbers 1095
H.2 Feature Compatibility 1098
H.3 Implementation Notes 1099

Appendix I: Computation of Object Digests 1131

I.1 Basic Object Types 1131
I.2 Selective Computation 1133

Color Plates 1139

Bibliography 1151

Index 1159

Figures

2.1 Creating PDF files using the Adobe PDF printer 44
2.2 Creating PDF files using Acrobat Distiller 45
3.1 PDF components 48
3.2 Initial structure of a PDF file 91
3.3 Structure of an updated PDF file 100
3.4 Public-key encryption algorithm 130
3.5 Structure of a PDF document 138
3.6 Inheritance of attributes 149
3.7 Relationship between string types 158
3.8 Mapping with the ***Decode*** array 173
4.1 Graphics objects 197
4.2 Device space 200
4.3 User space 202
4.4 Relationships among coordinate systems 204
4.5 Effects of coordinate transformations 205
4.6 Effect of transformation order 206
4.7 Miter length 217
4.8 Cubic Bézier curve generated by the ***c*** operator 228
4.9 Cubic Bézier curves generated by the ***v*** and ***y*** operators 229
4.10 Nonzero winding number rule 233
4.11 Even-odd rule 234
4.12 Color specification 238
4.13 Color rendering 239
4.14 Component transformations in a CIE-based ABC color space 245
4.15 Component transformations in a CIE-based A color space 246
4.16 Starting a new triangle in a free-form Gouraud-shaded triangle mesh 316
4.17 Connecting triangles in a free-form Gouraud-shaded triangle mesh 317
4.18 Varying the value of the edge flag to create different shapes 318
4.19 Lattice-form triangle meshes 319
4.20 Coordinate mapping from a unit square to a four-sided Coons patch 322
4.21 Painted area and boundary of a Coons patch 323
4.22 Color values and edge flags in Coons patch meshes 325
4.23 Edge connections in a Coons patch mesh 326
4.24 Control points in a tensor-product patch 328
4.25 Typical sampled image 334
4.26 Source image coordinate system 338
4.27 Mapping the source image 338

5.1 Glyphs painted in 50% gray 391
5.2 Glyph outlines treated as a stroked path 392
5.3 Graphics clipped by a glyph path 393
5.4 Glyph metrics 394
5.5 Metrics for horizontal and vertical writing modes 396
5.6 Character spacing in horizontal writing 399
5.7 Word spacing in horizontal writing 399
5.8 Horizontal scaling 400
5.9 Leading 400
5.10 Text rise 403
5.11 Operation of the ***TJ*** operator in horizontal writing 408
5.12 Output from Example 424
5.13 Characteristics represented in the ***Flags*** entry of a font descriptor 459
6.1 Various halftoning effects 494
6.2 Halftone cell with a nonzero angle 500
6.3 Angled halftone cell divided into two squares 501
6.4 Halftone cell and two squares tiled across device space 501
6.5 Tiling of device space in a type 16 halftone 503
6.6 Flatness tolerance 509
6.7 Rasterization without stroke adjustment 512
8.1 Presentation timing 601
8.2 Open annotation 604
8.3 Coordinate adjustment with the NoRotate flag 610
8.4 Free text annotation with callout 625
8.5 Leader lines 628
8.6 Lines with captions appearing as part of the line 629
8.7 Line with a caption appearing as part of the offset 629
8.8 Square and circle annotations 631
8.9 QuadPoints specification 634
8.10 FDF file structure 712
9.1 Default view of artwork 802
9.2 Annotation 2 rotated 803
9.3 Shared artwork (annotations 2 &3) modified 803
9.4 Rotation around the center of orbit 807
9.5 Perspective projection of 3D artwork onto the near plane 810
9.6 Objects projected onto the near clipping plane, as seen from the position of the camera 811
9.7 Positioning and scaling the near plane onto the annotation's 3D view box 811
9.8 3D annotation positioned on the page 812
9.9 Rendering of the 3D artwork using View0 (no cross section) 824
9.10 Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis) 825

9.11 Rendering of the 3D artwork using View2 (cross section rotated around the y axis by -30 degrees) 826
9.12 Rendering of the 3D artwork using View3 (cross section rotated around the z axis by 30 degrees) 827
9.13 Rendering of the 3D artwork using View4 (cross section rotated around the y axis by -30 degrees and around the z axis by 30 degrees) 828
9.14 Rendering of the 3D artwork using View1 (all shapes visible and opaque) 831
9.15 Rendering of the 3D artwork using View2 (the cone is hidden and the sphere is semi-transparent) 832
9.16 3D artwork set to its default view 838
9.17 3D artwork set to CommentView1 839
9.18 3D artwork set to CommentView2 839
10.1 Simple Web Capture file structure 948
10.2 Complex Web Capture file structure 949
10.3 Page boundaries 964
10.4 Trapping example 974
G.1 Output of Example G.3 1063
G.2 Page tree for Example G.4 1065
G.3 Document outline as displayed in Example G.5 1070
G.4 Document outline as displayed in Example G.6 1072
G.5 Table of contents 1082
G.6 Association between content and marked content identifiers 1083
G.7 Hierarchy of structure elements and relationship with marked content 1084
G.8 Index 1089
G.9 Hierarchy of structure elements and relationship with marked content 1090
Plate 1 Additive and subtractive color (Section 4.5.3, "Device Color Spaces," page 241)
Plate 2 Uncalibrated color (Section 4.5.4, "CIE-Based Color Spaces," page 244)
Plate 3 ***Lab*** color space ("Lab Color Spaces," page 250)
Plate 4 Color gamuts ("Lab Color Spaces," page 250)
Plate 5 Rendering intents ("Rendering Intents," page 260)
Plate 6 Duotone image ("DeviceN Color Spaces," page 269)
Plate 7 Quadtone image ("DeviceN Color Spaces," page 269)
Plate 8 Colored tiling pattern ("Colored Tiling Patterns," page 295)
Plate 9 Uncolored tiling pattern ("Uncolored Tiling Patterns," page 299)
Plate 10 Axial shading ("Type 2 (Axial) Shadings," page 310)
Plate 11 Radial shadings depicting a cone ("Type 3 (Radial) Shadings," page 312)
Plate 12 Radial shadings depicting a sphere ("Type 3 (Radial) Shadings," page 313)
Plate 13 Radial shadings with extension ("Type 3 (Radial) Shadings," page 313)
Plate 14 Radial shading effect ("Type 3 (Radial) Shadings," page 313)

Plate 15 Coons patch mesh (“Type 6 Shadings (Coons Patch Meshes),” page 321)
Plate 16 Transparency groups (Section 7.1, “Overview of Transparency,” page 515)
Plate 17 Isolated and knockout groups (Sections 7.3.4, “Isolated Groups,” page 539 and 7.3.5, “Knockout Groups,” page 540)
Plate 18 RGB blend modes (Section 7.2.4, “Blend Mode,” page 520)
Plate 19 CMYK blend modes (Section 7.2.4, “Blend Mode,” page 520)
Plate 20 Blending and overprinting (“Compatibility with Opaque Overprinting,” page 569)

Tables

3.1 White-space characters 50
3.2 Escape sequences in literal strings 54
3.3 Examples of literal names using the # character 57
3.4 Entries common to all stream dictionaries 62
3.5 Standard filters 67
3.6 Typical LZW encoding sequence 73
3.7 Optional parameters for LZWDecode and FlateDecode filters 74
3.8 Predictor values 76
3.9 Optional parameters for the CCITTFaxDecode filter 78
3.10 Optional parameter for the JBIG2Decode filter 82
3.11 Optional parameter for the DCTDecode filter 85
3.12 Optional parameters for Crypt filters 90
3.13 Entries in the file trailer dictionary 97
3.14 Additional entries specific to an object stream dictionary 101
3.15 Additional entries specific to a cross-reference stream dictionary 107
3.16 Entries in a cross-reference stream 109
3.17 Additional entries in a hybrid-reference file's trailer dictionary 110
3.18 Entries common to all encryption dictionaries 116
3.19 Additional encryption dictionary entries for the standard security handler 122
3.20 User access permissions 123
3.21 Additional encryption dictionary entries for public-key security handlers 129
3.22 Entries common to all crypt filter dictionaries 132
3.23 Standard crypt filter names 134
3.24 Additional crypt filter dictionary entries for public-key security handlers 134
3.25 Entries in the catalog dictionary 139
3.26 Required entries in a page tree node 143
3.27 Entries in a page object 145
3.28 Entries in the name dictionary 150
3.29 Compatibility operators 152
3.30 Entries in a resource dictionary 154
3.31 PDF data types 155
3.32 String Types 157
3.33 Entries in a name tree node dictionary 162

3.34 Entries in a number tree node dictionary 166
3.35 Entries common to all function dictionaries 168
3.36 Additional entries specific to a type 0 function dictionary 170
3.37 Additional entries specific to a type 2 function dictionary 173
3.38 Additional entries specific to a type 3 function dictionary 174
3.39 Operators in type 4 functions 176
3.40 Examples of file specifications 181
3.41 Entries in a file specification dictionary 182
3.42 Additional entries in an embedded file stream dictionary 185
3.43 Entries in an embedded file parameter dictionary 186
3.44 Entries in a Mac OS file information dictionary 186
3.45 Entries in a collection item dictionary 189
3.46 Entries in a collection subitem dictionary 189
4.1 Operator categories 196
4.2 Device-independent graphics state parameters 210
4.3 Device-dependent graphics state parameters 212
4.4 Line cap styles 216
4.5 Line join styles 216
4.6 Examples of line dash patterns 218
4.7 Graphics state operators 219
4.8 Entries in a graphics state parameter dictionary 220
4.9 Path construction operators 226
4.10 Path-painting operators 230
4.11 Clipping path operators 235
4.12 Color space families 237
4.13 Entries in a CalGray color space dictionary 246
4.14 Entries in a CalRGB color space dictionary 248
4.15 Entries in a Lab color space dictionary 251
4.16 Additional entries specific to an ICC profile stream dictionary 253
4.17 ICC specification versions supported by ICCBased color spaces 253
4.18 ICC profile types 254
4.19 Ranges for typical ICC color spaces 255
4.20 Rendering intents 261
4.21 Entries in a DeviceN color space attributes dictionary 272
4.22 Entries in a DeviceN process dictionary 274
4.23 Entries in a DeviceN mixing hints dictionary 274
4.24 Color operators 287
4.25 Additional entries specific to a type 1 pattern dictionary 292
4.26 Entries in a type 2 pattern dictionary 302
4.27 Shading operator 303
4.28 Entries common to all shading dictionaries 305
4.29 Additional entries specific to a type 1 shading dictionary 308
4.30 Additional entries specific to a type 2 shading dictionary 309

4.31 Additional entries specific to a type 3 shading dictionary 311
4.32 Additional entries specific to a type 4 shading dictionary 315
4.33 Additional entries specific to a type 5 shading dictionary 320
4.34 Additional entries specific to a type 6 shading dictionary 324
4.35 Data values in a Coons patch mesh 327
4.36 Data values in a tensor-product patch mesh 331
4.37 XObject operator 332
4.38 Additional entries specific to a PostScript XObject dictionary 333
4.39 Additional entries specific to an image dictionary 340
4.40 Default Decode arrays 345
4.41 Entries in an alternate image dictionary 347
4.42 Inline image operators 352
4.43 Entries in an inline image object 353
4.44 Additional abbreviations in an inline image object 353
4.45 Additional entries specific to a type 1 form dictionary 358
4.46 Entries common to all group attributes dictionaries 361
4.47 Entries in a reference dictionary 362
4.48 Entries in an optional content group dictionary 364
4.49 Entries in an optional content membership dictionary 366
4.50 Entries in the optional content properties dictionary 375
4.51 Entries in an optional content configuration dictionary 376
4.52 Entries in an optional content usage dictionary 380
4.53 Entries in a usage application dictionary 382
5.1 Text state parameters 397
5.2 Text state operators 398
5.3 Text rendering modes 402
5.4 Text object operators 405
5.5 Text-positioning operators 406
5.6 Text-showing operators 407
5.7 Font types 411
5.8 Entries in a Type 1 font dictionary 413
5.9 Entries in a Type 3 font dictionary 420
5.10 Type 3 font operators 423
5.11 Entries in an encoding dictionary 427
5.12 Differences between MacRomanEncoding and Mac OS Roman encoding 431
5.13 Entries in a CIDSystemInfo dictionary 435
5.14 Entries in a CIDFont dictionary 436
5.15 Predefined CJK CMap names 442
5.16 Character collections for predefined CMaps, by PDF version 446
5.17 Additional entries in a CMap dictionary 448
5.18 Entries in a Type 0 font dictionary 452
5.19 Entries common to all font descriptors 456

5.20 Font flags 458
5.21 Additional font descriptor entries for CIDFonts 461
5.22 Glyph classes in CJK fonts 463
5.23 Embedded font organization for various font types 465
5.24 Additional entries in an embedded font stream dictionary 466
6.1 Predefined spot functions 489
6.2 PDF halftone types 496
6.3 Entries in a type 1 halftone dictionary 497
6.4 Additional entries specific to a type 6 halftone dictionary 499
6.5 Additional entries specific to a type 10 halftone dictionary 502
6.6 Additional entries specific to a type 16 halftone dictionary 504
6.7 Entries in a type 5 halftone dictionary 505
7.1 Variables used in the basic compositing formula 518
7.2 Standard separable blend modes 520
7.3 Standard nonseparable blend modes 524
7.4 Variables used in the source shape and opacity formulas 528
7.5 Variables used in the result shape and opacity formulas 529
7.6 Revised variables for the basic compositing formulas 532
7.7 Arguments and results of the group compositing function 534
7.8 Variables used in the group compositing formulas 536
7.9 Variables used in the page group compositing formulas 543
7.10 Entries in a soft-mask dictionary 553
7.11 Restrictions on the entries in a soft-mask image dictionary 554
7.12 Additional entry in a soft-mask image dictionary 555
7.13 Additional entries specific to a transparency group attributes dictionary 556
7.14 Overprinting behavior in the opaque imaging model 570
7.15 Overprinting behavior in the transparent imaging model 571
8.1 Entries in a viewer preferences dictionary 578
8.2 Destination syntax 582
8.3 Entries in the outline dictionary 585
8.4 Entries in an outline item dictionary 585
8.5 Outline item flags 587
8.6 Entries in a collection dictionary 589
8.7 Entries in a collection schema dictionary 590
8.8 Entries in a collection field dictionary 591
8.9 Entries in a collection sort dictionary 592
8.10 Entries in a page label dictionary 595
8.11 Entries in a thread dictionary 596
8.12 Entries in a bead dictionary 597
8.13 Entries in a transition dictionary 599
8.14 Entries in a navigation node dictionary 602
8.15 Entries common to all annotation dictionaries 606

8.16 Annotation flags 608
8.17 Entries in a border style dictionary 611
8.18 Entries in a border effect dictionary 612
8.19 Entries in an appearance dictionary 614
8.20 Annotation types 615
8.21 Additional entries specific to markup annotations 618
8.22 Annotation states 620
8.23 Additional entries specific to a text annotation 621
8.24 Additional entries specific to a link annotation 622
8.25 Additional entries specific to a free text annotation 624
8.26 Additional entries specific to a line annotation 626
8.27 Line ending styles 630
8.28 Additional entries specific to a square or circle annotation 631
8.29 Additional entries specific to a polygon or polyline annotation 632
8.30 Additional entries specific to text markup annotations 634
8.31 Additional entries specific to a caret annotation 635
8.32 Additional entries specific to a rubber stamp annotation 635
8.33 Additional entries specific to an ink annotation 636
8.34 Additional entries specific to a pop-up annotation 637
8.35 Additional entries specific to a file attachment annotation 638
8.36 Additional entries specific to a sound annotation 638
8.37 Additional entries specific to a movie annotation 639
8.38 Additional entries specific to a screen annotation 640
8.39 Additional entries specific to a widget annotation 641
8.40 Entries in an appearance characteristics dictionary 642
8.41 Additional entries specific to a watermark annotation 644
8.42 Entries in a fixed print dictionary 645
8.43 Entries common to all action dictionaries 648
8.44 Entries in an annotation's additional-actions dictionary 649
8.45 Entries in a page object's additional-actions dictionary 650
8.46 Entries in a form field's additional-actions dictionary 651
8.47 Entries in the document catalog's additional-actions dictionary 651
8.48 Action types 653
8.49 Additional entries specific to a go-to action 654
8.50 Additional entries specific to a remote go-to action 655
8.51 Additional entries specific to an embedded go-to action 656
8.52 Entries specific to a target dictionary 657
8.53 Additional entries specific to a launch action 660
8.54 Entries in a Windows launch parameter dictionary 660
8.55 Additional entries specific to a thread action 661
8.56 Additional entries specific to a URI action 662
8.57 Entry in a URI dictionary 663
8.58 Additional entries specific to a sound action 664

8.59 Additional entries specific to a movie action 665
8.60 Additional entries specific to a hide action 666
8.61 Named actions 666
8.62 Additional entries specific to named actions 667
8.63 Additional entries specific to a set-OCG-state action 667
8.64 Additional entries specific to a rendition action 669
8.65 Additional entries specific to a transition action 670
8.66 Additional entries specific to a go-to-3D-view action 670
8.67 Entries in the interactive form dictionary 672
8.68 Signature flags 674
8.69 Entries common to all field dictionaries 675
8.70 Field flags common to all field types 676
8.71 Additional entries common to all fields containing variable text 678
8.72 XHTML elements used in rich text strings 681
8.73 Attributes of the <body> element 681
8.74 CSS2 style attributes used in rich text strings 682
8.75 Field flags specific to button fields 686
8.76 Additional entry specific to check box and radio button fields 688
8.77 Field flags specific to text fields 691
8.78 Additional entry specific to a text field 692
8.79 Field flags specific to choice fields 693
8.80 Additional entries specific to a choice field 694
8.81 Additional entries specific to a signature field 696
8.82 Entries in a signature field lock dictionary 697
8.83 Entries in a signature field seed value dictionary 697
8.84 Entries in a certificate seed value dictionary 700
8.85 Additional entries specific to a submit-form action 703
8.86 Flags for submit-form actions 704
8.87 Additional entries specific to a reset-form action 707
8.88 Flag for reset-form actions 708
8.89 Additional entries specific to an import-data action 708
8.90 Additional entries specific to a JavaScript action 709
8.91 Entry in the FDF trailer dictionary 713
8.92 Entries in the FDF catalog dictionary 714
8.93 Entries in the FDF dictionary 714
8.94 Additional entry in an embedded file stream dictionary for an encrypted FDF file 716
8.95 Entries in the JavaScript dictionary 716
8.96 Entries in an FDF field dictionary 717
8.97 Entries in an icon fit dictionary 719
8.98 Entries in an FDF page dictionary 720
8.99 Entries in an FDF template dictionary 721
8.100 Entries in an FDF named page reference dictionary 721

8.101 Additional entry for annotation dictionaries in an FDF file 722
8.102 Entries in a signature dictionary 727
8.103 Entries in a signature reference dictionary 730
8.104 Entries in the DocMDP transform parameters dictionary 733
8.105 Entries in the UR transform parameters dictionary 734
8.106 Entries in the FieldMDP transform parameters dictionary 736
8.107 Entries in a permissions dictionary 741
8.108 Entries in a legal attestation dictionary 742
8.109 Entries in a viewport dictionary 745
8.110 Entries in a measure dictionary 746
8.111 Additional entries in a rectilinear measure dictionary 746
8.112 Entries in a number format dictionary 748
8.113 Entries common to all requirement dictionaries 751
8.114 Entries in a requirement handler dictionary 752
9.1 Entries common to all rendition dictionaries 759
9.2 Entries in a rendition MH/BE dictionary 760
9.3 Entries in a media criteria dictionary 760
9.4 Entries in a minimum bit depth dictionary 761
9.5 Entries in a minimum screen size dictionary 762
9.6 Additional entries in a media rendition dictionary 762
9.7 Additional entries specific to a selector rendition dictionary 763
9.8 Entries common to all media clip dictionaries 764
9.9 Additional entries in a media clip data dictionary 764
9.10 Entries in a media permissions dictionary 766
9.11 Entries in a media clip data MH/BE dictionary 767
9.12 Additional entries in a media clip section dictionary 767
9.13 Entries in a media clip section MH/BE dictionary 768
9.14 Entries in a media play parameters dictionary 769
9.15 Entries in a media play parameters MH/BE dictionary 769
9.16 Entries in a media duration dictionary 771
9.17 Entries in a media screen parameters dictionary 772
9.18 Entries in a media screen parameters MH/BE dictionary 772
9.19 Entries in a floating window parameters dictionary 774
9.20 Entries common to all media offset dictionaries 775
9.21 Additional entries in a media offset time dictionary 776
9.22 Additional entries in a media offset frame dictionary 776
9.23 Additional entries in a media offset marker dictionary 776
9.24 Entries in a timespan dictionary 776
9.25 Entries in a media players dictionary 777
9.26 Entries in a media player info dictionary 779
9.27 Entries in a software identifier dictionary 780
9.28 Monitor specifier values 782
9.29 Additional entries specific to a sound object 783

9.30 Entries in a movie dictionary 784
9.31 Entries in a movie activation dictionary 785
9.32 Entries in a slideshow dictionary 787
9.33 Additional entries specific to a 3D annotation 791
9.34 Entries in a 3D activation dictionary 794
9.35 Entries in a 3D stream dictionary 797
9.36 Entries in an 3D animation style dictionary 799
9.37 Animation styles 800
9.38 Entries in a 3D reference dictionary 801
9.39 Entries in a 3D view dictionary 804
9.40 Entries in a projection dictionary 808
9.41 Entries in a 3D background dictionary 812
9.42 Entries in a render mode dictionary 813
9.43 Render modes 815
9.44 Entries in a 3D lighting scheme dictionary 817
9.45 3D lighting scheme styles 817
9.46 Entries in a 3D cross section dictionary 819
9.47 Entries in a 3D node dictionary 829
9.48 Entries in an external data dictionary used to markup 3D annotations 835
10.1 Predefined procedure sets 842
10.2 Entries in the document information dictionary 844
10.3 Additional entries in a metadata stream dictionary 846
10.4 Additional entry for components having metadata 846
10.5 Entries in a page-piece dictionary 849
10.6 Entries in an application data dictionary 849
10.7 Marked-content operators 851
10.8 Entries in the mark information dictionary 856
10.9 Entries in the structure tree root 857
10.10 Entries in a structure element dictionary 858
10.11 Entries in a marked-content reference dictionary 863
10.12 Entries in an object reference dictionary 868
10.13 Additional dictionary entries for structure element access 870
10.14 Entry common to all attribute object dictionaries 873
10.15 Additional entries in an attribute object dictionary for user properties 876
10.16 Entries in a user property dictionary 876
10.17 Property list entries for artifacts 886
10.18 Derivation of font characteristics 893
10.19 Font Selector Attributes 894
10.20 Standard structure types for grouping elements 899
10.21 Block-level structure elements 901
10.22 Standard structure types for paragraphlike elements 902
10.23 Standard structure types for list elements 902
10.24 Standard structure types for table elements 903

10.25 Standard structure types for inline-level structure elements 905
10.26 Standard structure types for Ruby and Warichu elements (*PDF 1.5*) 911
10.27 Standard structure types for illustration elements 912
10.28 Standard attribute owners 914
10.29 Standard layout attributes 916
10.30 Standard layout attributes common to all standard structure types 917
10.31 Additional standard layout attributes specific to block-level structure elements 922
10.32 Standard layout attributes specific to inline-level structure elements 926
10.33 Standard column attributes 932
10.34 Standard list attribute 933
10.35 PrintField attributes 934
10.36 Standard table attributes 935
10.37 Entries in the Web Capture information dictionary 947
10.38 Entries common to all Web Capture content sets 953
10.39 Additional entries specific to a Web Capture page set 954
10.40 Additional entries specific to a Web Capture image set 955
10.41 Entries in a source information dictionary 955
10.42 Entries in a URL alias dictionary 957
10.43 Entries in a Web Capture command dictionary 958
10.44 Web Capture command flags 958
10.45 Entries in a Web Capture command settings dictionary 960
10.46 Entries in a box color information dictionary 967
10.47 Entries in a box style dictionary 967
10.48 Additional entries specific to a printer's mark annotation 968
10.49 Additional entries specific to a printer's mark form dictionary 968
10.50 Entries in a separation dictionary 969
10.51 Entries in a PDF/X output intent dictionary 971
10.52 Additional entries specific to a trap network annotation 976
10.53 Additional entries specific to a trap network appearance stream 978
10.54 Entry in an OPI version dictionary 979
10.55 Entries in a version 1.3 OPI dictionary 980
10.56 Entries in a version 2.0 OPI dictionary 983
A.1 PDF content stream operators 985
C.1 Architectural limits 992
D.1 Latin-text encodings 996
F.1 Entries in the linearization parameter dictionary 1029
F.2 Standard hint tables 1033
F.3 Page offset hint table, header section 1041
F.4 Page offset hint table, per-page entry 1042
F.5 Shared object hint table, header section 1044
F.6 Shared object hint table, shared object group entry 1045
F.7 Thumbnail hint table, header section 1046

F.8 Thumbnail hint table, per-page entry 1047
F.9 Generic hint table 1048
F.10 Extended generic hint table 1049
F.11 Embedded file stream hint table, header section 1050
F.12 Embedded file stream hint table, per-embedded file stream group entries 1050
G.1 Objects in minimal example 1058
G.2 Objects in simple text string example 1060
G.3 Objects in simple graphics example 1062
G.4 Object usage after adding four text annotations 1075
G.5 Object usage after deleting two text annotations 1078
G.6 Object usage after adding three text annotations 1080
H.1 Abbreviations for standard filter names 1100
H.2 Acrobat behavior with unknown filters 1101
H.3 Names of standard fonts 1109
H.4 Recommended media types 1123
I.1 Data added to object digest for basic object types 1132

Preface

The origins of the Portable Document Format and the Adobe® Acrobat® product family date to early 1990. At that time, the PostScript® page description language was rapidly becoming the worldwide standard for the production of the printed page. PDF builds on the PostScript page description language by layering a document structure and interactive navigation features on PostScript's underlying imaging model, providing a convenient, efficient mechanism enabling documents to be reliably viewed and printed anywhere.

The PDF specification was first published at the same time the first Acrobat products were introduced in 1993. Since then, updated versions of the specification have been and continue to be available from Adobe on the World Wide Web. It includes the precise documentation of the underlying imaging model from PostScript along with the PDF-specific features that are combined in version 1.7 of the PDF standard.

Over the past eleven years, aided by the explosive growth of the Internet, PDF has become the *de facto* standard for the electronic exchange of documents. Well over 500 million copies of the free Adobe Reader® software have been distributed around the world, facilitating efficient sharing of digital content. In addition, PDF is now the industry standard for the intermediate representation of printed material in electronic prepress systems for conventional printing applications. As major corporations, government agencies, and educational institutions streamline their operations by replacing paper-based workflow with electronic exchange of information, the impact and opportunity for the application of PDF will continue to grow at a rapid pace.

PDF is the file format that underlies the Adobe® Intelligent Document Platform, facilitating the process of creating, managing, securing, collecting, and exchanging digital content on diverse platforms and devices. The Intelligent Document

Platform fulfills a set of requirements related to business process needs for the global desktop user, including:

- Preservation of document fidelity across the enterprise, independently of the device, platform, and software
- Merging of content from diverse sources—Web sites, word processing and spreadsheet programs, scanned documents, photos, and graphics—into one self-contained document while maintaining the integrity of all original source documents
- Real-time collaborative editing of documents from multiple locations or platforms
- Digital signatures to certify authenticity
- Security and permissions to allow the creator to retain control of the document and associated rights
- Accessibility of content to those with disabilities
- Extraction and reuse of content using other file formats and applications
- Electronic forms to gather data and integrate it with business systems.

The emergence of PDF as a standard for electronic information exchange is the result of concerted effort by many individuals in both the private and public sectors. Without the dedication of Adobe employees, our industry partners, and our customers, the widespread acceptance of PDF could not have been achieved. We thank all of you for your continuing support and creative contributions to the success of PDF.

Chuck Geschke and John Warnock
November 2004

CHAPTER 1

Introduction

The Adobe Portable Document Format (PDF) is the native file format of the Adobe® Acrobat® family of products. The goal of these products is to enable users to exchange and view electronic documents easily and reliably, independently of the environment in which they were created. PDF relies on the same imaging model as the PostScript® page description language to describe text and graphics in a device-independent and resolution-independent manner. To improve performance for interactive viewing, PDF defines a more structured format than that used by most PostScript language programs. PDF also includes objects, such as annotations and hypertext links, that are not part of the page itself but are useful for interactive viewing and document interchange.

1.1 About This Book

This book provides a description of the PDF file format and is intended primarily for developers of *PDF producer* applications that create PDF files directly. It also contains enough information to allow developers to write *PDF consumer* applications that read existing PDF files and interpret or modify their contents.

Although the *PDF Reference* is independent of any particular software implementation, some PDF features are best explained by describing the way they are processed by a typical application program. In such cases, this book uses the Acrobat family of PDF viewer applications as its model. (The prototypical viewer is the fully capable Acrobat product, not the limited Adobe Reader® product.) Appendix C discusses some implementation limits in the Acrobat viewer applications, even though these limits are not part of the file format itself. Appendix H provides compatibility and implementation notes that describe how Acrobat viewers behave when they encounter newer features they do not understand and specify areas in which the Acrobat products diverge from the specification presented in this book. Implementors of PDF producer and consumer applications can use this information as guidance.

This edition of the *PDF Reference* describes version 1.7 of PDF. (See implementation note 1 in Appendix H.) Throughout the book, information specific to particular versions of PDF is marked with indicators such as *(PDF 1.3)* or *(PDF 1.4)*. Features so marked may be new or substantially redefined in that version. Features designated *(PDF 1.0)* have generally been superseded in later versions; unless otherwise stated, features identified as specific to other versions are understood to be available in later versions as well. (PDF consumer applications designed for a specific PDF version generally ignore newer features they do not recognize; implementation notes in Appendix H point out exceptions.)

Note: *In this edition, the term* consumer *is generally used to refer to PDF processing applications;* viewer *is reserved for applications that implement features that interact with users. This distinction is not always clear, however, since non-interactive applications may process objects in PDF documents (such as annotations) that represent interactive features.*

The rest of the book is organized as follows:

- Chapter 2, "Overview," briefly introduces the overall architecture of PDF and the design considerations behind it, compares it with the PostScript language, and describes the underlying imaging model that they share.
- Chapter 3, "Syntax," presents the syntax of PDF at the object, file, and document level. It sets the stage for subsequent chapters, which describe how that information is interpreted as page descriptions, interactive navigational aids, and application-level logical structure.
- Chapter 4, "Graphics," describes the graphics operators used to describe the appearance of pages in a PDF document.
- Chapter 5, "Text," discusses PDF's special facilities for presenting text in the form of character shapes, or glyphs, defined by fonts.
- Chapter 6, "Rendering," considers how device-independent content descriptions are matched to the characteristics of a particular output device.
- Chapter 7, "Transparency," discusses the operation of the transparent imaging model, introduced in PDF 1.4, in which objects can be painted with varying degrees of opacity, allowing the previous contents of the page to show through.

- Chapter 8, "Interactive Features," describes those features of PDF that allow a user to interact with a document on the screen by using the mouse and keyboard.
- Chapter 9, "Multimedia Features," describes those features of PDF that support embedding and playing multimedia content, including video, music and 3D artwork.
- Chapter 10, "Document Interchange," shows how PDF documents can incorporate higher-level information that is useful for the interchange of documents among applications.
- Appendix A, "Operator Summary," lists all the operators used in describing the visual content of a PDF document.
- Appendix B, "Operators in Type 4 Functions," summarizes the PostScript operators that can be used in PostScript calculator functions, which contain code written in a small subset of the PostScript language.
- Appendix C, "Implementation Limits," describes typical size and quantity limits imposed by the Acrobat viewer applications.
- Appendix D, "Character Sets and Encodings," lists the character sets and encodings that are assumed to be predefined in any PDF consumer application.
- Appendix E, "PDF Name Registry," discusses a registry, maintained for developers by Adobe Systems, that contains private names and formats used by PDF producers or Acrobat plug-in extensions.
- Appendix F, "Linearized PDF," describes a special form of PDF file organization designed to work efficiently in network environments.
- Appendix G, "Example PDF Files," presents several examples showing the structure of actual PDF files, ranging from one containing a minimal one-page document to one showing how the structure of a PDF file evolves over the course of several revisions.
- Appendix H, "Compatibility and Implementation Notes," provides details on the behavior of Acrobat viewer applications and describes how consumer applications should handle PDF files containing features that they do not recognize.
- Appendix I, "Computation of Object Digests," describes in detail an algorithm for calculating an object digest (discussed in Section 8.7, "Digital Signatures").

A color plate section provides illustrations of some of PDF's color-related features. References in the text of the form "see Plate 1" refer to the contents of this section.

The book concludes with a Bibliography and an Index.

1.2 Introduction to PDF 1.7 Features

Several features have been introduced or modified in PDF 1.7. The following is a list of the most significant additions, along with references to the primary sections where those additions are discussed:

1.2.1 Presentation of 3D Artwork

PDF 1.7 introduces new features that increase the control the PDF viewing application has over the appearance and behavior of 3D artwork:

- More control over the appearance of 3D artwork, without having to change the original artwork and without the use of embedded JavaScript. Specific views of 3D artwork can specify how that artwork should be rendered, colored, lit, and cross-sectioned. They can also specify which nodes (three-dimensional areas) of 3D artwork should be included in a view, where those nodes should be placed in the view, and whether they should be transparent. These features can expose areas of geometry that would otherwise be difficult to view.
- The ability to place markup annotations on specific views of 3D artwork. This ensures that markups applied to 3D artwork can later be shown properly with respect to both the artwork as a whole and individual elements within the artwork. Markup annotations applied to 3D artwork provide a means of ensuring the artwork has not changed since the markup annotation was applied.
- Control over the user interfaces and toolbars presented on activation of 3D artwork.
- Control over the timeframe, repetition, and style of play of keyframe animations. The styles of play are linear repetition (as in a walking character) and a cosine-based repetition (as in an exploding-contracting image).

1.2.2 Interactive Features

Several additions to markup annotations make them more suitable for technical communication and review, or for use in a legal setting.

Interactive Features That Aid Technical Communication

Several additions to markup annotations aid technical communication and review:

- The addition of dimension intents for polyline and polygon markup annotations. Dimension intent supports the association of user-provided dimension information with the line segments that compose polyline and polygon markup annotations. This feature is similar to the dimension intent introduced for line markup annotations in PDF 1.6.
- The ability to specify units and scaling for the dimension intents of line, polyline, and polygon markup annotations. This feature enables users to measure distances in the document, such as the width of an architectural diagram or the diameter of a 3D cross section.
- The ability to place markup annotations on specific views of 3D artwork
- The ability to lock the contents of an annotation

Interactive Feature for Use in a Legal Setting

One addition to markup annotations is intended for use in a legal setting, especially banking. The addition of new viewer preference settings that specify print characteristics, such as paper selection and handling, page range, copies, and scaling. When a user prints a PDF document with those viewer preference settings, the print dialog is pre-populated as specified in those settings. This capability increases the predictability of how PDF documents are printed, which can make PDF documents more suitable for use in a legal setting.

1.2.3 Accessibility Related Features

Additions to TaggedPDF identify the roles of more types of page content:

- The ability to identify the roles of form fields in non-interactive PDF documents. This change identifies button fields (pushbuttons, check boxes and radio buttons) and text fields (populated or unpopulated).

- The ability to provide table summaries associated with table structures. This feature can help a visually impaired person understand the purpose and structure of a table without having to read the content in that table.
- The ability to identify background page artifacts, which can be important to document reflowing. Background artifacts are collections of objects that do not contribute to the meaning of the author's original content, such as a colored rectangle behind a sidebar or a full-page background image. Such page backgrounds may not correlate to any logical structure, but they may be useful in reproducing the appearance of original document.
- The ability to differentiate the pagination artifacts: watermarks, headers and footers.

1.2.4 Document Navigation Feature

Additions to document navigation specify the viewing and organizational characteristics of *portable collections*, in which multiple file attachments are displayed within a single window. Portable collections are used to present, sort, and search collections of related documents, such as email archives, photo collections, and engineering bid sets.

1.2.5 Security-Related Features

Additions to PDF introduced in 1.7 increase the control the document author can impose upon digital signatures and over requirements PDF consumer applications must satisfy:

- Additional digital signature constraints, which are enforced at the time the signature is applied. These constraints include preferred digest methods, revocation checking of the certificate used in a signature, and flags that clarify the interpretation of other parameters.
- Additional constraints regarding the certificate to be used when signing. These constraints include Subject Distinguished Name (DN) dictionaries that must be present in the certificate, KeyUsage extensions that must be present in the signing certificate, and flags that clarify the interpretation of other parameters that specify certificate constraints.
- The ability to specify requirement handlers that verify some requirement that the PDF consumer applications must satisfy before processing or displaying a PDF document. This feature provides an approach that ensures backward com-

patibility with PDF documents that may include JavaScript segments to verify a requirement. Before this feature was added, JavaScript was the only way to perform such requirement-checking. The feature ensures that either the JavaScript segment verifies the requirement or a named handler verifies the requirement.

1.2.6 General Features

Additions to PDF 1.7 provide more cross-platform and cross-application stability, by providing encoding information for strings and file names:

- The clarification of string types to describe the encodings used for strings. Throughout the entire PDF Reference, any uses of the string type are replaced with one of the more specific string types. This clarification does not require changes to PDF consumer applications. Instead, it provides a clearer understanding of the encoding supported by each PDF string entry. This understanding can be especially important when comparing strings in a PDF document to strings in an external source, such as an XML document or 3D artwork.
- The ability to specify file names using Unicode in addition to specifying file names using the standard encoding for the platform on which the document is being viewed. This feature reduces problems in decoding file path names that have been encoded on a different platform or in a different language.

1.2.7 PDF Reference Changes

This release of the *PDF Reference* includes clarifications not related to new features or additional capabilities:

- A description of the formulas for all blend modes.
- An explanation of the TaggedPDF representation of nested table of contents entries or list entries.

1.3 Related Publications

PDF and the PostScript page description language share the same underlying Adobe imaging model. A document can be converted straightforwardly between PDF and the PostScript language; the two representations produce the same output when printed. However, PostScript includes a general-purpose programming language framework not present in PDF. The *PostScript Language Reference* is the comprehensive reference for the PostScript language and its imaging model.

PDF and PostScript support several standard formats for font programs, including Adobe Type 1, CFF (Compact Font Format), TrueType, OpenType and CID-keyed fonts. The PDF manifestations of these fonts are documented in this book. However, the specifications for the font files themselves are published separately, because they are highly specialized and are of interest to a different user community. A variety of Adobe publications are available on the subject of font formats. The Bibliography lists these publications, as well as additional documents related to PDF and the contents of this book.

1.4 Intellectual Property

Adobe owns copyrights in the *PDF Reference*. Adobe will enforce its copyrights. One reason Adobe must retain its copyrights in the *PDF Reference* is to maintain the integrity of the Portable Document Format standard and ensure that the public can distinguish between the Portable Document Format and other interchange formats for electronic documents. Nonetheless, Adobe desires to promote the use of the Portable Document Format for information interchange among diverse products and applications. Accordingly, Adobe gives permission to everyone under its copyrights to copy, modify, and distribute any example code in the written specification, to the extent necessary to implement the Portable Document Format in a manner compliant with the *PDF Reference*.[1]

Adobe Systems Incorporated and its subsidiaries own a number of patents covering technology disclosed in the *PDF Reference*. Nothing in the *PDF Reference* itself grants rights under any patent. Nonetheless, Adobe desires to encourage implementation of the PDF computer file format on a wide variety of devices and platforms, and for this reason offers certain royalty-free patent licenses to PDF implementors worldwide. To review those licenses, please visit
http://www.adobe.com/go/developer_legalnotices.

1.This example code includes, but is not limited to, the copyrighted list of data structures, operators, and PostScript language function definitions, that were referenced in PDF Reference, fifth edition, version 1.6, Section 1.5 (Intellectual Property).

CHAPTER 2

Overview

PDF is a file format for representing documents in a manner independent of the application software, hardware, and operating system used to create them and of the output device on which they are to be displayed or printed. A *PDF document* consists of a collection of *objects* that together describe the appearance of one or more *pages*, possibly accompanied by additional interactive elements and higher-level application data. A *PDF file* contains the objects making up a PDF document along with associated structural information, all represented as a single self-contained sequence of bytes.

A document's pages (and other visual elements) can contain any combination of text, graphics, and images. A page's appearance is described by a PDF *content stream*, which contains a sequence of *graphics objects* to be painted on the page. This appearance is fully specified; all layout and formatting decisions have already been made by the application generating the content stream.

In addition to describing the static appearance of pages, a PDF document can contain interactive elements that are possible only in an electronic representation. PDF supports *annotations* of many kinds for such things as text notes, hypertext links, markup, file attachments, sounds, and movies. A document can define its own user interface; keyboard and mouse input can trigger *actions* that are specified by PDF objects. The document can contain *interactive form* fields to be filled in by the user, and can export the values of these fields to or import them from other applications.

Finally, a PDF document can contain higher-level information that is useful for interchange of content among applications. In addition to specifying appearance, a document's content can include identification and logical structure information that allows it to be searched, edited, or extracted for reuse elsewhere. PDF is particularly well suited for representing a document as it moves through successive stages of a prepress production workflow.

2.1 Imaging Model

At the heart of PDF is its ability to describe the appearance of sophisticated graphics and typography. This ability is achieved through the use of the *Adobe imaging model*, the same high-level, device-independent representation used in the PostScript page description language.

Although application programs could theoretically describe any page as a full-resolution pixel array, the resulting file would be bulky, device-dependent, and impractical for high-resolution devices. A high-level imaging model enables applications to describe the appearance of pages containing text, graphical shapes, and sampled images in terms of abstract graphical elements rather than directly in terms of device pixels. Such a description is economical and device-independent, and can be used to produce high-quality output on a broad range of printers, displays, and other output devices.

2.1.1 Page Description Languages

Among its other roles, PDF serves as a *page description language*, a language for describing the graphical appearance of pages with respect to an imaging model. An application program produces output through a two-stage process:

1. The application generates a device-independent description of the desired output in the page description language.
2. A program controlling a specific output device interprets the description and *renders* it on that device.

The two stages may be executed in different places and at different times. The page description language serves as an interchange standard for the compact, device-independent transmission and storage of printable or displayable documents.

2.1.2 Adobe Imaging Model

The Adobe imaging model is a simple and unified view of two-dimensional graphics borrowed from the graphic arts. In this model, "paint" is placed on a page in selected areas:

- The painted figures can be in the form of character shapes *(glyphs)*, geometric shapes, lines, or sampled images such as digital representations of photographs.
- The paint may be in color or in black, white, or any shade of gray. It may also take the form of a repeating *pattern (PDF 1.2)* or a smooth transition between colors *(PDF 1.3)*.
- Any of these elements may be *clipped* to appear within other shapes as they are placed onto the page.

A page's content stream contains *operands* and *operators* describing a sequence of graphics objects. A PDF consumer application maintains an implicit *current page* that accumulates the marks made by the painting operators. Initially, the current page is completely blank. For each graphics object encountered in the content stream, the application places marks on the current page, which replace or combine with any previous marks they may overlay. Once the page has been completely composed, the accumulated marks are rendered on the output medium and the current page is cleared to blank again.

PDF 1.3 and earlier versions use an *opaque imaging model* in which each new graphics object painted onto a page completely obscures the previous contents of the page at those locations (subject to the effects of certain optional parameters that may modify this behavior; see Section 4.5.6, "Overprint Control"). No matter what color an object has—white, black, gray, or color—it is placed on the page as if it were applied with opaque paint. PDF 1.4 introduces a *transparent imaging model* in which objects painted on the page are not required to be fully opaque. Instead, newly painted objects are *composited* with the previously existing contents of the page, producing results that combine the colors of the object and its backdrop according to their respective opacity characteristics. The transparent imaging model is described in Chapter 7.

The principal graphics objects (among others) are as follows:

- A *path object* consists of a sequence of connected and disconnected points, lines, and curves that together describe shapes and their positions. It is built up through the sequential application of *path construction operators*, each of which appends one or more new elements. The path object is ended by a *path-painting operator*, which paints the path on the page in some way. The principal path-painting operators are **S** (stroke), which paints a line along the path, and **f** (fill), which paints the interior of the path.

- A *text object* consists of one or more glyph shapes representing characters of text. The glyph shapes for the characters are described in a separate data structure called a *font*. Like path objects, text objects can be stroked or filled.
- An *image object* is a rectangular array of *sample values*, each representing a color at a particular position within the rectangle. Such objects are typically used to represent photographs.

The painting operators require various parameters, some explicit and others implicit. Implicit parameters include the current color, current line width, current font (typeface and size), and many others. Together, these implicit parameters make up the *graphics state*; there are operators for setting the value of each implicit parameter in the graphics state. Painting operators use the values currently in effect at the time they are invoked.

One additional implicit parameter in the graphics state modifies the results of painting graphics objects. The *current clipping path* outlines the area of the current page within which paint can be placed. Although painting operators may attempt to place marks anywhere on the current page, only those marks falling within the current clipping path affect the page; those falling outside it do not affect the page. Initially, the current clipping path encompasses the entire imageable area of the page. It can temporarily be reduced to the shape defined by a path or text object, or to the intersection of multiple such shapes. Marks placed by subsequent painting operators are confined within that boundary.

2.1.3 Raster Output Devices

Much of the power of the Adobe imaging model derives from its ability to deal with the general class of *raster output devices.* These encompass such technologies as laser, dot-matrix, and ink-jet printers, digital imagesetters, and raster-scan displays. The defining property of a raster output device is that a printed or displayed image consists of a rectangular array, or *raster*, of dots called *pixels* (picture elements) that can be addressed individually. On a typical *bilevel* output device, each pixel can be made either black or white. On some devices, pixels can be set to intermediate shades of gray or to some color. The ability to set the colors of individual pixels makes it possible to generate printed or displayed output that can include text, arbitrary graphical shapes, and reproductions of sampled images.

The *resolution* of a raster output device measures the number of pixels per unit of distance along the two linear dimensions. Resolution is typically—but not necessarily—the same horizontally and vertically. Manufacturers' decisions on device technology and price/performance trade-offs create characteristic ranges of resolution:

- Computer displays have relatively low resolution, typically 75 to 110 pixels per inch.
- Dot-matrix printers generally range from 100 to 250 pixels per inch.
- Ink-jet and laser-scanned xerographic printing technologies achieve medium-level resolutions of 300 to 1400 pixels per inch.
- Photographic technology permits high resolutions of 2400 pixels per inch or more.

Higher resolution yields better quality and fidelity of the resulting output but is achieved at greater cost. As the technology improves and computing costs decrease, products evolve to higher resolutions.

2.1.4 Scan Conversion

An abstract graphical element (such as a line, a circle, a character glyph, or a sampled image) is rendered on a raster output device by a process known as *scan conversion*. Given a mathematical description of the graphical element, this process determines which pixels to adjust and what values to assign to those pixels to achieve the most faithful rendition possible at the available device resolution.

The pixels on a page can be represented by a two-dimensional array of pixel values in computer memory. For an output device whose pixels can only be black or white, a single bit suffices to represent each pixel. For a device that can reproduce gray levels or colors, multiple bits per pixel are required.

Note: *Although the ultimate representation of a printed or displayed page is logically a complete array of pixels, its actual representation in computer memory need not consist of one memory cell per pixel. Some implementations use other representations, such as display lists. The Adobe imaging model has been carefully designed not to depend on any particular representation of raster memory.*

For each graphical element that is to appear on the page, the scan converter sets the values of the corresponding pixels. When the interpretation of the page description is complete, the pixel values in memory represent the appearance of the page. At this point, a raster output process can *render* this representation (make it visible) on a printed page or display screen.

Scan-converting a graphical shape, such as a rectangle or circle, entails determining which device pixels lie inside the shape and setting their values appropriately (for example, to black). Because the edges of a shape do not always fall precisely on the boundaries between pixels, some policy is required for deciding how to set the pixels along the edges. Scan-converting a glyph representing a text character is conceptually the same as scan-converting an arbitrary graphical shape. However, character glyphs are much more sensitive to legibility requirements and must meet more rigid objective and subjective measures of quality.

Rendering grayscale elements on a bilevel device is accomplished by a technique known as *halftoning*. The array of pixels is divided into small clusters according to some pattern (called the *halftone screen*). Within each cluster, some pixels are set to black and others to white in proportion to the level of gray desired at that location on the page. When viewed from a sufficient distance, the individual dots become imperceptible and the perceived result is a shade of gray. This enables a bilevel raster output device to reproduce shades of gray and to approximate natural images such as photographs. Some color devices use a similar technique.

2.2 Other General Properties

This section describes other notable general properties of PDF, aside from its imaging model.

2.2.1 Portability

PDF files are represented as sequences of 8-bit binary bytes. A PDF file is designed to be portable across all platforms and operating systems. The binary representation is intended to be generated, transported, and consumed directly, without translation between native character sets, end-of-line representations, or other conventions used on various platforms.

Any PDF file can also be represented in a form that uses only 7-bit ASCII (American Standard Code for Information Interchange) character codes. This is useful for the purpose of exposition, as in this book. However, this representation is not recommended for actual use, since it is less efficient than the normal binary representation. Regardless of which representation is used, PDF files must be transported and stored as binary files, not as text files. Inadvertent changes, such as conversion between text end-of-line conventions, will damage the file and may render it unusable.

2.2.2 Compression

To reduce file size, PDF supports a number of industry-standard compression filters:

- JPEG and (in PDF 1.5) JPEG2000 compression of color and grayscale images
- CCITT (Group 3 or Group 4), run-length, and (in PDF 1.4) JBIG2 compression of monochrome images
- LZW (Lempel-Ziv-Welch) and (beginning with PDF 1.2) Flate compression of text, graphics, and images

Using JPEG compression, color and grayscale images can be compressed by a factor of 10 or more. Effective compression of monochrome images depends on the compression filter used and the properties of the image, but reductions of 2:1 to 8:1 are common (or 20:1 to 50:1 for JBIG2 compression of an image of a page full of text). LZW or Flate compression of the content streams describing all other text and graphics in the document results in compression ratios of approximately 2:1. All of these compression filters produce binary data, which can be further converted to ASCII base-85 encoding if a 7-bit ASCII representation is required.

2.2.3 Font Management

Managing fonts is a fundamental challenge in document interchange. Generally, the receiver of a document must have the same fonts that were originally used to create it. If a different font is substituted, its character set, glyph shapes, and metrics may differ from those in the original font. This substitution can produce unexpected and unwanted results, such as lines of text extending into margins or overlapping with graphics.

PDF provides various means for dealing with font management:

- The original font programs can be embedded in the PDF file, which ensures the most predictable and dependable results. PDF supports various font formats, including Type 1, TrueType, OpenType, and CID-keyed fonts.
- To conserve space, a font subset can be embedded, containing just the glyph descriptions for those characters that are actually used in the document. Also, Type 1 fonts can be represented in a special compact format.
- PDF prescribes a set of 14 standard fonts that can be used without prior definition. These include four faces each of three Latin text typefaces (Courier, Helvetica*, and Times*), as well as two symbolic fonts (Symbol and ITC Zapf Dingbats®). These fonts, or suitable substitute fonts with the same metrics, are required to be available in all PDF consumer applications.
- A PDF file can refer by name to fonts that are not embedded in the PDF file. In this case, a PDF consumer can use those fonts if they are available in its environment. This approach suffers from the uncertainties noted above.
- A PDF file contains a *font descriptor* for each font that it uses. The font descriptor includes font metrics and style information, enabling an application to select or synthesize a suitable substitute font if necessary. Although the glyphs' shapes differ from those intended, their placement is accurate.

Font management is primarily concerned with producing the correct appearance of text—that is, the shape and placement of glyphs. However, it is sometimes necessary for a PDF application to extract the meaning of the text, represented in some standard information encoding such as Unicode. In some cases, this information can be deduced from the encoding used to represent the text in the PDF file. Otherwise, the PDF producer application should specify the mapping explicitly by including a special object, the ***ToUnicode*** *CMap*.

2.2.4 Single-Pass File Generation

Because of system limitations and efficiency considerations, it may be necessary or desirable for an application program to generate a PDF file in a single pass. For example, the program may have limited memory available or be unable to open temporary files. For this reason, PDF supports single-pass generation of files. Although some PDF objects must specify their length in bytes, a mechanism is provided allowing the length to follow the object in the PDF file. In addition, information such as the number of pages in the document can be written into the file after all pages have been generated.

A PDF file that is generated in a single pass is generally not ordered for most efficient viewing, particularly when accessing the contents of the file over a network. When generating a PDF file that is intended to be viewed many times, it is worthwhile to perform a second pass to optimize the order in which objects occur in the file. PDF specifies a particular file organization, *Linearized PDF*, which is documented in Appendix F. Other optimizations are also possible, such as detecting duplicated sequences of graphics objects and collapsing them to a single shared sequence that is specified only once.

2.2.5 Random Access

A PDF file should be thought of as a flattened representation of a data structure consisting of a collection of objects that can refer to each other in any arbitrary way. The order of the objects' occurrence in the PDF file has no semantic significance. In general, an application should process a PDF file by following references from object to object, rather than by processing objects sequentially. This is particularly important for interactive document viewing or for any application in which pages or other objects in the PDF file are accessed out of sequence.

To support such random access to individual objects, every PDF file contains a *cross-reference table* that can be used to locate and directly access pages and other important objects within the file. The cross-reference table is stored at the end of the file, allowing applications that generate PDF files in a single pass to store it easily and those that read PDF files to locate it easily. By using the cross-reference table, the time needed to locate a page or other object is nearly independent of the length of the document, allowing PDF documents containing hundreds or thousands of pages to be accessed efficiently.

2.2.6 Security

PDF has two security features that can be used, separately or together, in any document:

- The document can be *encrypted* so that only authorized users can access it. There is separate authorization for the owner of the document and for all other users; the users' access can be selectively restricted to allow only certain operations, such as viewing, printing, or editing.

- The document can be digitally *signed* to certify its authenticity. The signature may take many forms, including a document digest that has been encrypted with a public/private key, a biometric signature such as a fingerprint, and others. Any subsequent changes to a signed PDF file invalidate the signature.

2.2.7 Incremental Update

Applications may allow users to modify PDF documents. Users should not have to wait for the entire file—which can contain hundreds of pages or more—to be rewritten each time modifications to the document are saved. PDF allows modifications to be appended to a file, leaving the original data intact. The addendum appended when a file is incrementally updated contains only those objects that were actually added or modified, and includes an update to the cross-reference table. Incremental update allows an application to save modifications to a PDF document in an amount of time proportional to the size of the modification rather than the size of the file.

In addition, because the original contents of the document are still present in the file, it is possible to undo saved changes by deleting one or more addenda. The ability to recover the exact contents of an original document is critical when digital signatures have been applied and subsequently need to be verified.

2.2.8 Extensibility

PDF is designed to be extensible. Not only can new features be added, but applications based on earlier versions of PDF can behave reasonably when they encounter newer features that they do not understand. Appendix H describes how a PDF consumer application should behave in such cases.

Additionally, PDF provides means for applications to store their own private information in a PDF file. This information can be recovered when the file is imported by the same application, but it is ignored by other applications. Therefore, PDF can serve as an application's native file format while its documents can be viewed and printed by other applications. Application-specific data can be stored either as *marked content* annotating the graphics objects in a PDF content stream or as entirely separate objects unconnected with the PDF content.

2.3 Creating PDF

PDF files may be produced either directly by application programs or indirectly by conversion from other file formats or imaging models. As PDF documents and applications that process them become more prevalent, new ways of creating and using PDF will be invented.

Many applications can generate PDF files directly, and some can import them as well. This direct approach is preferable, since it gives the application access to the full capabilities of PDF, including the imaging model and the interactive and document interchange features. Alternatively, applications that do not generate PDF directly can produce PDF output indirectly. There are two principal indirect methods:

- The application describes its printable output by making calls to an application programming interface (API) such as GDI in Microsoft® Windows® or QuickDraw in the Apple Mac OS. A software component called a *printer driver* intercepts these calls and interprets them to generate output in PDF form.
- The application produces printable output directly in some other file format, such as PostScript, PCL, HPGL, or DVI, which is converted to PDF by a separate translation program.

Although these indirect strategies are often the easiest way to obtain PDF output from an existing application, the resulting PDF files may not make the best use of the high-level Adobe imaging model. This is because the information embodied in the application's API calls or in the intermediate output file often describes the desired results at too low a level. Any higher-level information maintained by the original application has been lost and is not available to the printer driver or translator.

Figures 2.1 and 2.2 show how Acrobat products support these indirect approaches. The Adobe PDF printer (Figure 2.1), available on the Windows and Mac OS platforms, acts as a printer driver, intercepting graphics and text operations generated by a running application program through the operating system's API. Instead of converting these operations into printer commands and transmitting them directly to a printer, the Adobe PDF printer converts them to equivalent PDF operators and embeds them in a PDF file. The result is a platform-independent file that can be viewed and printed by a PDF viewer application, such as Acrobat, running on any supported platform—even a different platform from the one on which the file was originally generated.

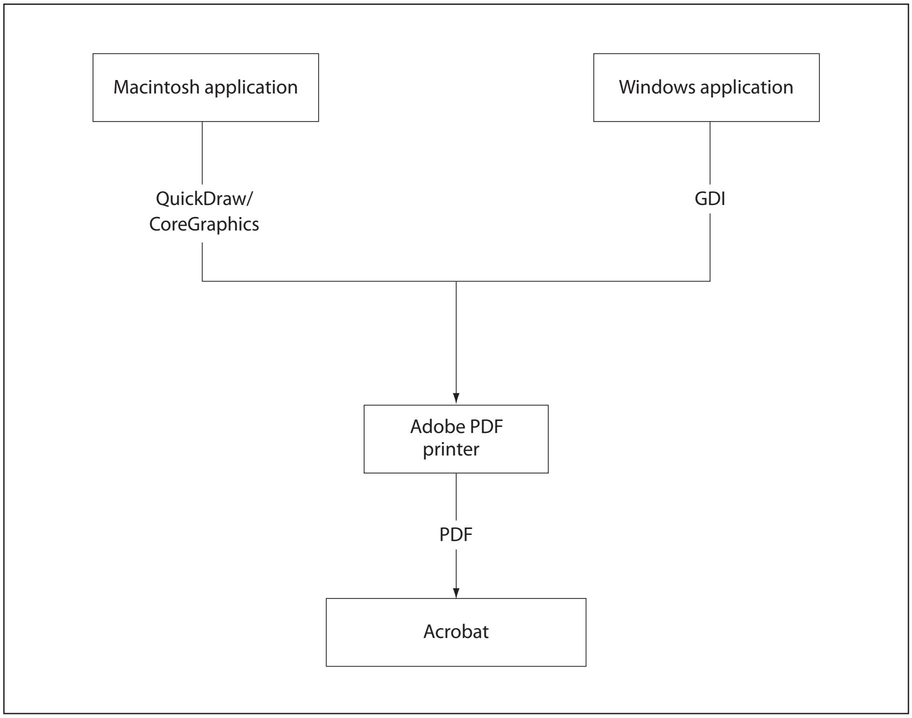

FIGURE 2.1 *Creating PDF files using the Adobe PDF printer*

Instead of describing their printable output through API calls, some applications produce PostScript page descriptions directly—either because of limitations in the QuickDraw or GDI imaging models or because the applications run on platforms such as DOS or UNIX®, where no system-level printer driver exists. PostScript files generated by such applications can be converted to PDF files using the Acrobat Distiller® application (see Figure 2.2). Because PostScript and PDF share the same Adobe imaging model, Distiller can preserve the exact graphical content of the PostScript file in the translation to PDF. Additionally, Distiller supports a PostScript language extension, called **pdfmark**, that allows the producing application to embed instructions in the PostScript file for creating hypertext links, logical structure, and other interactive and document interchange features of PDF. Again, the resulting PDF file can be viewed with a viewer application, such as Acrobat, on any supported platform.

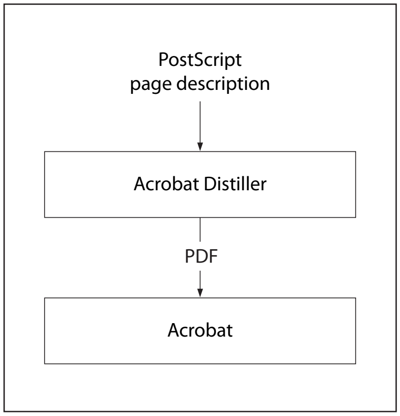

FIGURE 2.2 *Creating PDF files using Acrobat Distiller*

2.4 PDF and the PostScript Language

The PDF operators for setting the graphics state and painting graphics objects are similar to the corresponding operators in the PostScript language. Unlike PostScript, however, PDF is not a full-scale programming language; it trades reduced flexibility for improved efficiency and predictability. PDF therefore differs from PostScript in the following significant ways:

- PDF enforces a strictly defined file structure that allows an application to access parts of a document in arbitrary order.
- To simplify the processing of content streams, PDF does not include common programming language features such as procedures, variables, and control constructs.
- PDF files contain information such as font metrics to ensure viewing fidelity.
- A PDF file may contain additional information that is not directly connected with the imaging model, such as hypertext links for interactive viewing and logical structure information for document interchange.

Because of these differences, a PDF file generally cannot be transmitted directly to a PostScript output device for printing (although a few such devices do also support PDF directly). An application printing a PDF document to a PostScript device must follow these steps:

1. Insert *procedure sets* containing PostScript procedure definitions to implement the PDF operators.
2. Extract the content for each page. Each content stream is essentially the script portion of a traditional PostScript program using very specific procedures, such as **m** for **moveto** and **l** for **lineto**.
3. Decode compressed text, graphics, and image data as necessary. The compression filters used in PDF are compatible with those used in PostScript; they may or may not be supported, depending on the LanguageLevel of the target output device.
4. Insert any needed resources, such as fonts, into the PostScript file. These can be either the original fonts or suitable substitute fonts based on the font metrics in the PDF file. Fonts may need to be converted to a format that the PostScript interpreter recognizes, such as Type 1 or Type 42.
5. Put the information in the correct order. The result is a traditional PostScript program that fully represents the visual aspects of the document but no longer contains PDF elements such as hypertext links, annotations, and bookmarks.
6. Transmit the PostScript program to the output device.

CHAPTER 3

Syntax

This chapter covers everything about the syntax of PDF at the object, file, and document level. It sets the stage for subsequent chapters, which describe how the contents of a PDF file are interpreted as page descriptions, interactive navigational aids, and application-level logical structure.

PDF syntax is best understood by thinking of it in four parts, as shown in Figure 3.1:

- *Objects*. A PDF document is a data structure composed from a small set of basic types of data objects. Section 3.1, "Lexical Conventions," describes the character set used to write objects and other syntactic elements. Section 3.2, "Objects," describes the syntax and essential properties of the objects. Section 3.2.7, "Stream Objects," provides complete details of the most complex data type, the stream object.
- *File structure*. The PDF file structure determines how objects are stored in a PDF file, how they are accessed, and how they are updated. This structure is independent of the semantics of the objects. Section 3.4, "File Structure," describes the file structure. Section 3.5, "Encryption," describes a file-level mechanism for protecting a document's contents from unauthorized access.
- *Document structure*. The PDF document structure specifies how the basic object types are used to represent components of a PDF document: pages, fonts, annotations, and so forth. Section 3.6, "Document Structure," describes the overall document structure; later chapters address the detailed semantics of the components.
- *Content streams*. A PDF *content stream* contains a sequence of instructions describing the appearance of a page or other graphical entity. These instructions, while also represented as objects, are conceptually distinct from the objects that

represent the document structure and are described separately. Section 3.7, "Content Streams and Resources," discusses PDF content streams and their associated resources.

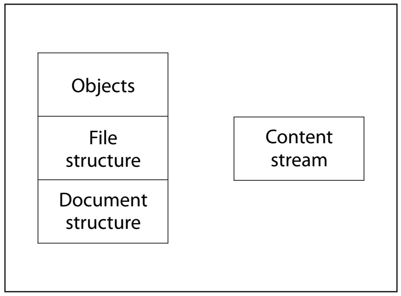

FIGURE 3.1 *PDF components*

In addition, this chapter describes some data structures, built from basic objects, that are so widely used that they can almost be considered basic object types in their own right. These objects are covered in Sections 3.8, "Common Data Structures"; 3.9, "Functions"; and 3.10, "File Specifications."

PDF's object and file syntax is also used as the basis for other file formats. These include the Forms Data Format (FDF), described in Section 8.6.6, "Forms Data Format," and the Portable Job Ticket Format (PJTF), described in Adobe Technical Note #5620, *Portable Job Ticket Format.*

3.1 Lexical Conventions

At the most fundamental level, a PDF file is a sequence of 8-bit bytes. These bytes can be grouped into *tokens* according to the syntax rules described below. One or more tokens are assembled to form higher-level syntactic entities, principally *objects*, which are the basic data values from which a PDF document is constructed.

PDF can be entirely represented using byte values corresponding to the visible printable subset of the ASCII character set, plus white space characters such as space, tab, carriage return, and line feed characters. ASCII is the American Standard Code for Information Interchange, a widely used convention for encoding a specific set of 128 characters as binary numbers. However, a PDF file is not restricted to the ASCII character set; it can contain arbitrary 8-bit bytes, subject to the following considerations:

- The tokens that delimit objects and that describe the structure of a PDF file are all written in the ASCII character set, as are all the reserved words and the names used as keys in standard dictionaries.
- The data values of certain types of objects—strings and streams—can be but need not be written entirely in ASCII. For the purpose of exposition (as in this book), ASCII representation is preferred. However, in actual practice, data that is naturally binary, such as sampled images, is represented directly in binary for compactness and efficiency.
- A PDF file containing binary data must be transported and stored by means that preserve all bytes of the file faithfully; that is, as a binary file rather than a text file. Such a file is not portable to environments that impose reserved character codes, maximum line lengths, end-of-line conventions, or other restrictions.

Note: *In this chapter, the term* character *is synonymous with* byte *and merely refers to a particular 8-bit value. This usage is entirely independent of any logical meaning that the value may have when it is treated as data in specific contexts, such as representing human-readable text or selecting a glyph from a font.*

3.1.1 Character Set

The PDF character set is divided into three classes, called *regular*, *delimiter*, and *white-space* characters. This classification determines the grouping of characters into tokens, except within strings, streams, and comments; different rules apply in those contexts.

White-space characters (see Table 3.1) separate syntactic constructs such as names and numbers from each other. All white-space characters are equivalent, except in comments, strings, and streams. In all other contexts, PDF treats any sequence of consecutive white-space characters as one character.

TABLE 3.1 White-space characters

DECIMAL	HEXADECIMAL	OCTAL	NAME
0	00	000	Null (NUL)
9	09	011	Tab (HT)
10	0A	012	Line feed (LF)
12	0C	014	Form feed (FF)
13	0D	015	Carriage return (CR)
32	20	040	Space (SP)

The carriage return (CR) and line feed (LF) characters, also called *newline characters*, are treated as *end-of-line* (EOL) markers. The combination of a carriage return followed immediately by a line feed is treated as one EOL marker. For the most part, EOL markers are treated the same as any other white-space characters. However, sometimes an EOL marker is required or recommended—that is, the following token must appear at the beginning of a line.

Note: *The examples in this book illustrate a recommended convention for arranging tokens into lines. However, the examples' use of white space for indentation is purely for clarity of exposition and is not recommended for practical use.*

The *delimiter characters* (,), <, >, [,], {, }, /, and % are special. They delimit syntactic entities such as strings, arrays, names, and comments. Any of these characters terminates the entity preceding it and is not included in the entity.

All characters except the white-space characters and delimiters are referred to as *regular characters*. These characters include 8-bit binary characters that are outside the ASCII character set. A sequence of consecutive regular characters comprises a single token.

Note: *PDF is case-sensitive; corresponding uppercase and lowercase letters are considered distinct.*