PDF 32000-1:2008

First Edition

2008-7-1

Document management — Portable document format — Part 1: PDF 1.7

Contents

Page

Foreword

On January 29, 2007, Adobe Systems Incorporated announced it's intention to release the full Portable Document Format (PDF) 1.7 specification to the American National Standard Institute (ANSI) and the Enterprise Content Management Association (AIIM), for the purpose of publication by the International Organization for Standardization (ISO).

PDF has become a de facto global standard for more secure and dependable information exchange since Adobe published the complete PDF specification in 1993. Both government and private industry have come to rely on PDF for the volumes of electronic records that need to be more securely and reliably shared, managed, and in some cases preserved for generations. Since 1995 Adobe has participated in various working groups that develop technical specifications for publication by ISO and worked within the ISO process to deliver specialized subsets of PDF as standards for specific industries and functions. Today, PDF for Archive (PDF/A) and PDF for Exchange (PDF/X) are ISO standards, and PDF for Engineering (PDF/E) and PDF for Universal Access (PDF/UA) are proposed standards. Additionally, PDF for Healthcare (PDF/H) is an AIIM proposed Best Practice Guide. AIIM serves as the administrator for PDF/A, PDF/E, PDF/UA and PDF/H.

In the spring of 2008 the ISO 32000 document was prepared by Adobe Systems Incorporated (based upon PDF Reference, sixth edition, Adobe Portable Document Format version 1.7, November 2006) and was reviewed, edited and adopted, under a special “fast-track procedure”, by Technical Committee ISO/TC 171, *Document management application*, Subcommittee SC 2, *Application issues*, in parallel with its approval by the ISO member bodies.

In January 2008, this ISO technical committee approved the final revised documentation for PDF 1.7 as the international standard ISO 32000-1. In July 2008 the ISO document was placed for sale on the ISO web site (http://www.iso.org).

This document you are now reading is a copy of the ISO 32000-1 standard. By agreement with ISO, Adobe Systems is allowed to offer this version of the ISO standard as a free PDF file on it's web site. It is not an official ISO document but the technical content is identical including the section numbering and page numbering.

Introduction

ISO 32000 specifies a digital form for representing documents called the Portable Document Format or usually referred to as PDF. PDF was developed and specified by Adobe Systems Incorporated beginning in 1993 and continuing until 2007 when this ISO standard was prepared. The Adobe Systems version PDF 1.7 is the basis for this ISO 32000 edition. The specifications for PDF are backward inclusive, meaning that PDF 1.7 includes all of the functionality previously documented in the Adobe PDF Specifications for versions 1.0 through 1.6. It should be noted that where Adobe removed certain features of PDF from their standard, they too are not contained herein.

The goal of PDF is to enable users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created or the environment in which they are viewed or printed. At the core of PDF is an advanced imaging model derived from the PostScript® page description language. This PDF Imaging Model enables the description of 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. Unlike Postscript, which is a programming language, PDF is based on a structured binary file format that is optimized for high performance in interactive viewing. PDF also includes objects, such as annotations and hypertext links, that are not part of the page content itself but are useful for interactive viewing and document interchange.

PDF files may be created natively in PDF form, converted from other electronic formats or digitized from paper, microform, or other hard copy format. Businesses, governments, libraries, archives and other institutions and individuals around the world use PDF to represent considerable bodies of important information.

Over the past fourteen years, aided by the explosive growth of the Internet, PDF has become widely used for the electronic exchange of documents. There are several specific applications of PDF that have evolved where limiting the use of some features of PDF and requiring the use of others, enhances the usefulness of PDF. ISO 32000 is an ISO standard for the full function PDF; the following standards are for more specialized uses. PDF/X (ISO 15930) is now the industry standard for the intermediate representation of printed material in electronic prepress systems for conventional printing applications. PDF/A (ISO 19005) is now the industry standard for the archiving of digital documents. PDF/E (ISO 24517) provides a mechanism for representing engineering documents and exchange of engineering data. 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, together with software for creating, viewing, printing and processing PDF files in a variety of ways, fulfils a set of requirements for electronic documents including:

- preservation of document fidelity independent 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,
- 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 for use with other file formats and applications, and
- electronic forms to gather data and integrate it with business systems.

The International Organization for Standardization draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning the creation, modification, display and processing of PDF files which are owned by the following parties:

- Adobe Systems Incorporated, 345 Park Avenue, San Jose, California,95110-2704, USA

ISO takes no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights has assured the ISO that they are willing to negotiate licenses under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO. Information may be obtained from those parties listed above.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO shall not be held responsible for identifying any or all such patent rights.

A repository of referenced documents has been established by AIIM (http://www.aiim.org/pdfrefdocs). Not all referenced documents can be found there because of copyright restrictions.

Document management — Portable document format —

Part 1:
PDF 1.7

IMPORTANT — The electronic file of this document contains colours which are considered to be useful for the correct understanding of the document. Users should therefore consider printing this document using a colour printer.

1 Scope

This International Standard specifies a digital form for representing electronic documents to enable users to exchange and view electronic documents independent of the environment in which they were created or the environment in which they are viewed or printed. It is intended for the developer of software that creates PDF files (conforming writers), software that reads existing PDF files and interprets their contents for display and interaction (conforming readers) and PDF products that read and/or write PDF files for a variety of other purposes (conforming products).

This standard does not specify the following:

- specific processes for converting paper or electronic documents to the PDF format;
- specific technical design, user interface or implementation or operational details of rendering;
- specific physical methods of storing these documents such as media and storage conditions;
- methods for validating the conformance of PDF files or readers;
- required computer hardware and/or operating system.

2 Conformance

2.1 General

Conforming PDF files shall adhere to all requirements of the ISO 32000-1 specification and a conforming file is not obligated to use any feature other than those explicitly required by ISO 32000-1.

NOTE 1 The proper mechanism by which a file can presumptively identify itself as being a PDF file of a given version level is described in 7.5.2, "File Header".

2.2 Conforming readers

A conforming reader shall comply with all requirements regarding reader functional behaviour specified in ISO 32000-1. The requirements of ISO 32000-1 with respect to reader behaviour are stated in terms of general functional requirements applicable to all conforming readers. ISO 32000-1 does not prescribe any specific technical design, user interface or implementation details of conforming readers. The rendering of conforming files shall be performed as defined by ISO 32000-1.

2.3 Conforming writers

A conforming writer shall comply with all requirements regarding the creation of PDF files as specified in ISO 32000-1. The requirements of ISO 32000-1 with respect to writer behaviour are stated in terms of general functional requirements applicable to all conforming writers and focus on the creation of conforming files. ISO 32000-1 does not prescribe any specific technical design, user interface or implementation details of conforming writers.

2.4 Conforming products

A conforming product shall comply with all requirements regarding the creation of PDF files as specified in ISO 32000-1 as well as comply with all requirements regarding reader functional behavior specified in ISO 32000-1.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 639-1:2002, *Codes for the representation of names of languages -- Part 1: Alpha-2 code.*

ISO 639-2:1998, *Codes for the representation of names of languages -- Part 2: Alpha-3 code.*

ISO 3166-1:2006, *Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes.*

ISO 3166-2:1998, *Codes for the representation of names of countries and their subdivisions -- Part 2: Country subdivision code.*

ISO/IEC 8824-1:2002, *Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO/IEC 10918-1:1994, *Digital Compression and Coding of Continuous-Tone Still Images* (informally known as the JPEG standard, for the Joint Photographic Experts Group, the ISO group that developed the standard).

ISO/IEC 15444-2:2004, *Information Technology—JPEG 2000 Image Coding System: Extensions.*

ISO/IEC 11544:1993/Cor 2:2001, *Information technology—Coded representation of picture and audio information—Progressive bi-level image compression* (JBIG2).

IEC/3WD 61966-2.1:1999, *Colour Measurement and Management in Multimedia Systems and Equipment, Part 2.1: Default RGB Colour Space—sRGB.*

ISO 15076-1:2005, *Image technology colour management - Architecture, profile format and data structure - Part 1:Based on ICC.1:2004-10.*

ISO 10646:2003, Information technology -- Universal Multiple-Octet Coded Character Set (UCS).

ISO/IEC 9541-1:1991, Information technology -- Font information interchange -- Part 1: Architecture.

ANSI X3.4-1986, Information Systems - Coded Sets 7-Bit American National Standard Code for Information Interchange (7-bit ASCII).

NOTE 1 The following documents can be found at AIIM at http://www.aiim.org/pdfrefdocs as well as at the Adobe Systems Incorporated Web Site http://www.adobe.com/go/pdf_ref_bibliography.

PDF Reference, Version 1.7, – 5th ed., (ISBN 0-321-30474-8), Adobe Systems Incorporated.

JavaScript for Acrobat API Reference, Version 8.0, (April 2007), Adobe Systems Incorporated.

Acrobat 3D JavaScript Reference, (April 2007), Adobe Systems Incorporated.

Adobe Glyph List, Version 2.0, (September 2002), Adobe Systems Incorporated.

OPI: Open Prepress Interface Specification 1.3, (September 1993), Adobe Systems Incorporated.

PDF Signature Build Dictionary Specification v.1.4, (March 2008), Adobe Systems Incorporated.

Adobe XML Architecture, Forms Architecture (XFA) Specification, version 2.5, (June 2007), Adobe Systems Incorporated*.*

Adobe XML Architecture, Forms Architecture (XFA) Specification, version 2.4, (September 2006), Adobe Systems Incorporated.

Adobe XML Architecture, Forms Architecture (XFA) Specification, version 2.2, (June 2005), Adobe Systems Incorporated.

Adobe XML Architecture, Forms Architecture (XFA) Specification, version 2.0, (October 2003), Adobe Systems Incorporated.

NOTE 2 Beginning with XFA 2.2, the XFA specification includes the Template Specification, the Config Specification, the XDP Specification, and all other XML specifications unique to the XML Forms Architecture (XFA).

Adobe XML Architecture, XML Data Package (XDP) Specification, version 2.0, (October 2003), Adobe Systems Incorporated.

Adobe XML Architecture, Template Specification, version 2.0, (October 2003), Adobe Systems Incorporated.

XML Forms Data Format Specification, version 2.0, (September 2007), Adobe Systems Incorporated.

XMP: Extensible Metadata Platform, (September 2005), Adobe Systems Incorporated.

TIFF Revision 6.0, Final, (June 1992), Adobe Systems Incorporated.

NOTE 3 The following Adobe Technical Notes can be found at the AIIM website at http://www.aiim.org/pdfnotes as well as at the Adobe Systems Incorporated Web Site (http://www.adobe.com) using the general search facility, entering the Technical Note number.

Technical Note #5004, Adobe Font Metrics File Format Specification, Version 4.1, (October 1998), Adobe Systems Incorporated.

NOTE 4 Adobe font metrics (AFM) files are available through the Type section of the ASN Web site.

Technical Note #5014, Adobe CMap and CID Font Files Specification, Version 1.0, (June 1993), Adobe Systems Incorporated.

Technical Note #5015, Type 1 Font Format Supplement, (May 1994), Adobe Systems Incorporated.

Technical Note #5078, Adobe-Japan1-4 Character Collection for CID-Keyed Fonts, (June 2004), Adobe Systems Incorporated.

Technical Note #5079, Adobe-GB1-4 Character Collection for CID-Keyed Fonts, (November 2000), Adobe Systems Incorporated.

Technical Note #5080, Adobe-CNS1-4 Character Collection for CID-Keyed Fonts, (May 2003), Adobe Systems Incorporated.

Technical Note #5087, Multiple Master Font Programs for the Macintosh, (February 1992), Adobe Systems Incorporated.

Technical Note #5088, Font Naming Issues, (April 1993), Adobe Systems Incorporated.

Technical Note #5092, CID-Keyed Font Technology Overview, (September 1994), Adobe Systems Incorporated.

Technical Note #5093, Adobe-Korea1-2 Character Collection for CID-Keyed Fonts, (May 2003), Adobe Systems Incorporated.

Technical Note #5094, Adobe CJKV Character Collections and CMaps for CID-Keyed Fonts, (June 2004), Adobe Systems Incorporated.

Technical Note #5097, Adobe-Japan2-0 Character Collection for CID-Keyed Fonts, (May 2003), Adobe Systems Incorporated.

Technical Note #5116, Supporting the DCT Filters in PostScript Level 2, (November 1992), Adobe Systems Incorporated.

Technical Note #5176, The Compact Font Format Specification, version 1.0, (December 2003), Adobe Systems Incorporated.

Technical Note #5177, The Type 2 Charstring Format, (December 2003), Adobe Systems Incorporated.

Technical Note #5411, ToUnicode Mapping File Tutorial, (May 2003), Adobe Systems Incorporated.

Technical Note #5620, Portable Job Ticket Format, Version 1.1, (April 1999), Adobe Systems Incorporated.

Technical Note #5660, Open Prepress Interface (OPI) Specification, Version 2.0, (January 2000), Adobe Systems Incorporated.

NOTE 5 The following documents are available as Federal Information Processing Standards Publications.

FIPS PUB 186-2, Digital Signature Standard, describes DSA signatures, (January 2000), Federal Information Processing Standards.

FIPS PUB 197, Advanced Encryption Standard (AES), (November 2001), Federal Information Processing Standards.

NOTE 6 The following documents are available as Internet Engineering Task Force RFCs.

RFC 1321, The MD5 Message-Digest Algorithm, (April 1992), Internet Engineering Task Force (IETF).

RFC 1738, Uniform Resource Locators, (December 1994), Internet Engineering Task Force (IETF).

RFC 1808, Relative Uniform Resource Locators, (June 1995), Internet Engineering Task Force (IETF).

RFC 1950, ZLIB Compressed Data Format Specification, Version 3.3, (May 1996), Internet Engineering Task Force (IETF).

RFC 1951, DEFLATE Compressed Data Format Specification, Version 1.3, (May 1996), Internet Engineering Task Force (IETF).

RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, (November 1996), Internet Engineering Task Force (IETF).

RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, (November 1996), Internet Engineering Task Force (IETF).

RFC 2083, PNG (Portable Network Graphics) Specification, Version 1.0, (March 1997), Internet Engineering Task Force (IETF).

RFC 2315, PKCS #7: Cryptographic Message Syntax, Version 1.5, (March 1998), Internet Engineering Task Force (IETF).

RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax, (August 1998), Internet Engineering Task Force (IETF).

RFC 2560, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP, (June 1999), Internet Engineering Task Force (IETF).

RFC 2616, Hypertext Transfer Protocol—HTTP/1.1, (June 1999), Internet Engineering Task Force (IETF).

RFC 2898, PKCS #5: Password-Based Cryptography Specification Version 2.0, (September 2000), Internet Engineering Task Force (IETF).

RFC 3066, Tags for the Identification of Languages, (January 2001), Internet Engineering Task Force (IETF).

RFC 3161, Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), (August 2001), Internet Engineering Task Force (IETF).

RFC 3174, US Secure Hash Algorithm 1 (SHA1), (September 2001), Internet Engineering Task Force (IETF).

RFC 3280, Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile, (April 2002), Internet Engineering Task Force (IETF).

NOTE 7 The following documents are available from other sources.

Adobe Type 1 Font Format., Version 1.1, (February 1993), Addison-Wesley, ISBN 0-201-57044-0.

OpenType Font Specification 1.4, December 2004, Microsoft.

*TrueType Reference Manual, (*December 2002), Apple Computer, Inc.

Standard ECMA-363, Universal 3D File Format, 1st Edition (U3D), (December 2004), Ecma International.

PANOSE Classification Metrics Guide, (February 1997), Hewlett-Packard Corporation.

ICC Characterization Data Registry, International Color Consortium (ICC).

Recommendations T.4 and T.6, Group 3 and Group 4 facsimile encoding, International Telecommunication Union (ITU).

TrueType 1.0 Font Files Technical Specification, Microsoft Corporation.

Client-Side JavaScript Reference, (May 1999), Mozilla Foundation.

The Unicode Standard, Version 4.0, Addison-Wesley, Boston, MA, 2003, Unicode Consortium.

Unicode Standard Annex #9, The Bidirectional Algorithm, Version 4.0.0, (April 2003), Unicode Consortium.

Unicode Standard Annex #14, Line Breaking Properties, Version 4.0.0, (April 2003), Unicode Consortium.

Unicode Standard Annex #29, Text Boundaries, Version 4.0.0, (March 2005), Unicode Consortium.

Extensible Markup Language (XML) 1.1, World Wide Web Consortium (W3C).

4 Terms and definitions

For the purposes of this document, these terms and definitions apply.

4.1
… (ellipsis)
An ellipsis is used within PDF examples to indicate omitted detail. Pairs of ellipses are also used to bracket comments, in italic, about such omitted detail.

4.2
8-bit value
(see byte)

4.3
array object
a one-dimensional collection of objects arranged sequentially and implicitly numbered starting at 0

4.4
ASCII
the American Standard Code for Information Interchange, a widely used convention for encoding a specific set of 128 characters as binary numbers defined in ANSI X3.4-1986

4.5
binary data
an ordered sequence of bytes

4.6
boolean objects
either the keyword **true** or the keyword **false**

4.7
byte
a group of 8 binary digits which collectively can be configured to represent one of 256 different values and various realizations of the 8 binary digits are widely used in today's electronic equipment

4.8
catalog
the primary dictionary object containing references directly or indirectly to all other objects in the document with the exception that there may be objects in the **trailer** that are not referred to by the **catalog**

4.9
character
numeric code representing an abstract symbol according to some defined character encoding rule

NOTE 1 There are three manifestations of characters in PDF, depending on context:

- A PDF file is represented as a sequence of 8-bit bytes, some of which are interpreted as character codes in the ASCII character set and some of which are treated as arbitrary binary data depending upon the context.
- The contents (data) of a string or stream object in some contexts are interpreted as character codes in the PDFDocEncoding or UTF-16 character set.
- The contents of a string within a PDF content stream in some situations are interpreted as character codes that select glyphs to be drawn on the page according to a character encoding that is associated with the text font.

4.10
character set
a defined set of symbols each assigned a unique character value

4.11
conforming reader
software application that is able to read and process PDF files that have been made in conformance with this specification and that itself conforms to requirements of conforming readers specified here [ISO 32000-1]

4.12
conforming product
software application that is both a conforming reader and a conforming writer

4.13
conforming writer
software application that is able to write PDF files that conform to this specification [ISO 32000-1]

4.14
content stream
stream object whose data consists of a sequence of instructions describing the graphical elements to be painted on a page

4.15
cross reference table
data structure that contains the byte offset start for each of the indirect objects within the file

4.16
developer
Any entity, including individuals, companies, non-profits, standards bodies, open source groups, etc., who are developing standards or software to use and extend ISO 32000-1.

4.17
dictionary object
an associative table containing pairs of objects, the first object being a name object serving as the key and the second object serving as the value and may be any kind of object including another dictionary

4.18
direct object
any object that has not been made into an indirect object

4.19
electronic document
electronic representation of a page-oriented aggregation of text, image and graphic data, and metadata useful to identify, understand and render that data, that can be reproduced on paper or displayed without significant loss of its information content

4.20
end-of-line marker (EOL marker)
one or two character sequence marking the end of a line of text, consisting of a CARRIAGE RETURN character (0Dh) or a LINE FEED character (0Ah) or a CARRIAGE RETURN followed immediately by a LINE FEED

4.21
FDF file
File conforming to the Forms Data Format containing form data or annotations that may be imported into a PDF file (see 12.7.7, “Forms Data Format”)

4.22
filter
an optional part of the specification of a stream object, indicating how the data in the stream should be decoded before it is used

4.23
font
identified collection of graphics that may be glyphs or other graphic elements [ISO 15930-4]

4.24
function
a special type of object that represents parameterized classes, including mathematical formulas and sampled representations with arbitrary resolution

4.25
glyph
recognizable abstract graphic symbol that is independent of any specific design [ISO/IEC 9541-1]

4.26
graphic state
the top of a push down stack of the graphics control parameters that define the current global framework within which the graphics operators execute

4.27
ICC profile
colour profile conforming to the ICC specification [ISO 15076-1:2005]

4.28
indirect object
an object that is labeled with a positive integer object number followed by a non-negative integer generation number followed by **obj** and having **endobj** after it

4.29
integer object
mathematical integers with an implementation specified interval centered at 0 and written as one or more decimal digits optionally preceded by a sign

4.30
name object
an atomic symbol uniquely defined by a sequence of characters introduced by a SOLIDUS (/), (2Fh) but the SOLIDUS is not considered to be part of the name

4.31
name tree
similar to a dictionary that associates keys and values but the keys in a name tree are strings and are ordered

4.32
null object
a single object of type null, denoted by the keyword **null**, and having a type and value that are unequal to those of any other object

4.33
number tree
similar to a dictionary that associates keys and values but the keys in a number tree are integers and are ordered

4.34
numeric object
either an integer object or a real object

4.35
object
a basic data structure from which PDF files are constructed and includes these types: array, Boolean, dictionary, integer, name, null, real, stream and string

4.36
object reference
an object value used to allow one object to refer to another; that has the form “<n> <m> R” where <n> is an indirect object number, <m> is its version number and R is the uppercase letter R

4.37
object stream
a stream that contains a sequence of PDF objects

4.38
PDF
Portable Document Format file format defined by this specification [ISO 32000-1]

4.39
real object
approximate mathematical real numbers, but with limited range and precision and written as one or more decimal digits with an optional sign and a leading, trailing, or embedded PERIOD (2Eh) (decimal point)

4.40
rectangle
a specific array object used to describe locations on a page and bounding boxes for a variety of objects and written as an array of four numbers giving the coordinates of a pair of diagonally opposite corners, typically in the form [ll_x ll_y ur_x ur_y] specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the rectangle, in that order

4.41
resource dictionary
associates resource names, used in content streams, with the resource objects themselves and organized into various categories (e.g., Font, ColorSpace, Pattern)

4.42
space character
text string character used to represent orthographic white space in text strings

NOTE 2 space characters include HORIZONTAL TAB (U+0009), LINE FEED (U+000A), VERTICAL TAB (U+000B), FORM FEED (U+000C), CARRIAGE RETURN (U+000D), SPACE (U+0020), NOBREAK SPACE (U+00A0), EN SPACE (U+2002), EM SPACE (U+2003), FIGURE SPACE (U+2007), PUNCTUATION SPACE (U+2008), THIN SPACE (U+2009), HAIR SPACE (U+200A), ZERO WIDTH SPACE (U+200B), and IDEOGRAPHIC SPACE (U+3000)

4.43
stream object
consists of a dictionary followed by zero or more bytes bracketed between the keywords stream and endstream

4.44
string object
consists of a series of bytes (unsigned integer values in the range 0 to 255) and the bytes are not integer objects, but are stored in a more compact form

4.45
web capture
refers to the process of creating PDF content by importing and possibly converting internet-based or locally-resident files. The files being imported may be any arbitrary format, such as HTML, GIF, JPEG, text, and PDF

4.46
white-space character
characters that separate PDF syntactic constructs such as names and numbers from each other; white space characters are HORIZONTAL TAB (09h), LINE FEED (0Ah), FORM FEED (0Ch), CARRIAGE RETURN (0Dh), SPACE (20h); (see Table 1 in 7.2.2, “Character Set”)

4.47
XFDF file
file conforming to the XML Forms Data Format 2.0 specification, which is an XML transliteration of Forms Data Format (FDF)

4.48
XMP packet
structured wrapper for serialized XML metadata that can be embedded in a wide variety of file formats

5 Notation

PDF operators, PDF keywords, the names of keys in PDF dictionaries, and other predefined names are written in bold sans serif font; words that denote operands of PDF operators or values of dictionary keys are written in italic sans serif font.

Token characters used to delimit objects and describe the structure of PDF files, as defined in 7.2, "Lexical Conventions", may be identified by their ANSI X3.4-1986 (ASCII 7-bit USA codes) character name written in upper case in bold sans serif font followed by a parenthetic two digit hexadecimal character value with the suffix “h”.

Characters in text streams, as defined by 7.9.2, "String Object Types", may be identified by their ANSI X3.4-1986 (ASCII 7-bit USA codes) character name written in uppercase in sans serif font followed by a parenthetic four digit hexadecimal character code value with the prefix “U+” as shown in EXAMPLE 1 in this clause.

EXAMPLE 1 **EN SPACE** (U+2002).

6 Version Designations

For the convenience of the reader, the PDF versions in which various features were introduced are provided informatively within this document. The first version of PDF was designated PDF 1.0 and was specified by Adobe Systems Incorporated in the PDF Reference 1.0 document published by Adobe and Addison Wesley. Since then, PDF has gone through seven revisions designated as: PDF 1.1, PDF 1.2, PDF 1.3, PDF 1.4, PDF 1.5, PDF 1.6 and PDF 1.7. All non-deprecated features defined in a previous PDF version were also included in the subsequent PDF version. Since ISO 32000-1 is a PDF version matching PDF 1.7, it is also suitable for interpretation of files made to conform with any of the PDF specifications 1.0 through 1.7. Throughout this specification in order to indicate at which point in the sequence of versions a feature was introduced, a notation with a PDF version number in parenthesis (e.g., *(PDF 1.3)*) is used. Thus if a feature is labelled with *(PDF 1.3)* it means that PDF 1.0, PDF 1.1 and PDF 1.2 were not specified to support this feature whereas all versions of PDF 1.3 and greater were defined to support it.

7 Syntax

7.1 General

This clause covers everything about the syntax of PDF at the object, file, and document level. It sets the stage for subsequent clauses, 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 considering it as four parts, as shown in Figure 1:

- *Objects*. A PDF document is a data structure composed from a small set of basic types of data objects. Sub-clause 7.2, "Lexical Conventions," describes the character set used to write objects and other syntactic elements. Sub-clause 7.3, "Objects," describes the syntax and essential properties of the objects. Sub-clause 7.3.8, "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. Sub-clause 7.5, "File Structure," describes the file structure. Sub-clause 7.6, "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. Sub-clause 7.7, "Document Structure," describes the overall document structure; later clauses 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. Sub-clause 7.8, "Content Streams and Resources," discusses PDF content streams and their associated resources.

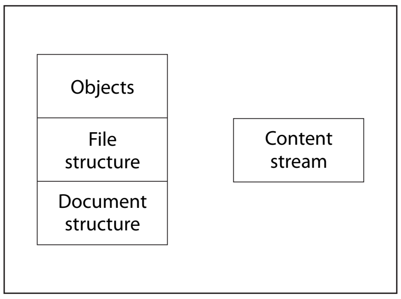


Figure 1 – PDF Components

In addition, this clause 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: 7.9, "Common Data Structures"; 7.10, "Functions"; and 7.11, "File Specifications."

NOTE Variants of PDF's object and file syntax are also used as the basis for other file formats. These include the Forms Data Format (FDF), described in 12.7.7, "Forms Data Format", and the Portable Job Ticket Format (PJTF), described in Adobe Technical Note #5620, *Portable Job Ticket Format*.

7.2 Lexical Conventions

7.2.1 General

At the most fundamental level, a PDF file is a sequence of bytes. These bytes can be grouped into *tokens* according to the syntax rules described in this sub-clause. 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.

A non-encrypted PDF can be entirely represented using byte values corresponding to the visible printable subset of the character set defined in ANSI X3.4-1986, plus white space characters. However, a PDF file is not restricted to the ASCII character set; it may contain arbitrary bytes, subject to the following considerations:

- The tokens that delimit objects and that describe the structure of a PDF file shall use the ASCII character set. In addition all the reserved words and the names used as keys in PDF standard dictionaries and certain types of arrays shall be defined using the ASCII character set.
- The data values of strings and streams objects may be written either entirely using the ASCII character set or entirely in binary data. In actual practice, data that is naturally binary, such as sampled images, is usually represented in binary for compactness and efficiency.
- A PDF file containing binary data shall be transported as a binary file rather than as a text file to insure that all bytes of the file are faithfully preserved.

NOTE 1 A binary file is not portable to environments that impose reserved character codes, maximum line lengths, end-of-line conventions, or other restrictions

NOTE 2 In this clause, the usage of the term character 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.

7.2.2 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. The rules defined in this sub-clause apply to all characters in the file except within strings, streams, and comments.

The *White-space characters* shown in Table 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 1 – White-space characters

Decimal	Hexadecimal	Octal	Name
0	00	000	Null (NUL)
9	09	011	HORIZONTAL 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 (0Dh) and LINE FEED (0Ah) characters, also called *newline characters*, shall be treated as *end-of-line* (EOL) markers. The combination of a CARRIAGE RETURN followed immediately by a LINE FEED shall be treated as one EOL marker. EOL markers may be treated the same as any other white-space characters. However, sometimes an EOL marker is required or recommended—that is, preceding a token that must appear at the beginning of a line.

NOTE The examples in this standard use a convention that arranges tokens into lines. However, the examples' use of white space for indentation is purely for clarity of exposition and need not be included in practical use.

The *delimiter characters* (,), <, >, [,], {, }, /, and % are special (LEFT PARENTHESIS (28h), RIGHT PARENTHESIS (29h), LESS-THAN SIGN (3Ch), GREATER-THAN SIGN (3Eh), LEFT SQUARE BRACKET (5Bh), RIGHT SQUARE BRACKET (5Dh), LEFT CURLY BRACE (7Bh), RIGHT CURLY BRACE (07Dh), SOLIDUS (2Fh) and PERCENT SIGN (25h), respectively). They delimit syntactic entities such as arrays, names, and comments. Any of these characters terminates the entity preceding it and is not included in the entity. Delimiter characters are allowed within the scope of a string when following the rules for composing strings; see 7.3.4.2, “Literal Strings”. The leading (of a string does delimit a preceding entity and the closing) of a string delimits the string’s end.

Table 2 – Delimiter characters

Glyph	Decimal	Hexadecimal	Octal	Name
(	40	28	50	LEFT PARENTHESIS
)	41	29	51	RIGHT PARENTHESIS
<	60	3C	60	LESS-THAN SIGN
>	62	3E	62	GREATER-THAN SIGN
[	91	5B	133	LEFT SQUARE BRACKET
]	93	5D	135	RIGHT SQUARE BRACKET
{	123	7B	173	LEFT CURLY BRACKET
}	125	7D	175	RIGHT CURLY BRACKET
/	47	2F	57	SOLIDUS
%	37	25	45	PERCENT SIGN

All characters except the white-space characters and delimiters are referred to as *regular characters*. These characters include bytes that are outside the ASCII character set. A sequence of consecutive regular characters comprises a single token. PDF is case-sensitive; corresponding uppercase and lowercase letters shall be considered distinct.

7.2.3 Comments

Any occurrence of the PERCENT SIGN (25h) outside a string or stream introduces a *comment*. The comment consists of all characters after the PERCENT SIGN and up to but not including the end of the line, including regular, delimiter, SPACE (20h), and HORZONTAL TAB characters (09h). A conforming reader shall ignore comments, and treat them as single white-space characters. That is, a comment separates the token preceding it from the one following it.

EXAMPLE The PDF fragment in this example is syntactically equivalent to just the tokens abc and 123.

```
abc% comment ( /%) blah blah blah
123
```

Comments (other than the %PDF–*n.m* and %%EOF comments described in 7.5, "File Structure") have no semantics. They are not necessarily preserved by applications that edit PDF files.

7.3 Objects

7.3.1 General

PDF includes eight basic types of objects: Boolean values, Integer and Real numbers, Strings, Names, Arrays, Dictionaries, Streams, and the null object.

Objects may be labelled so that they can be referred to by other objects. A labelled object is called an indirect object (see 7.3.10, "Indirect Objects").

Each object type, their method of creation and their proper referencing as indirect objects is described in 7.3.2, "Boolean Objects" through 7.3.10, "Indirect Objects."

7.3.2 Boolean Objects

Boolean objects represent the logical values of true and false. They appear in PDF files using the keywords **true** and **false**.

7.3.3 Numeric Objects

PDF provides two types of numeric objects: integer and real. *Integer objects* represent mathematical integers. *Real objects* represent mathematical real numbers. The range and precision of numbers may be limited by the internal representations used in the computer on which the conforming reader is running; Annex C gives these limits for typical implementations.

An integer shall be written as one or more decimal digits optionally preceded by a sign. The value shall be interpreted as a signed decimal integer and shall be converted to an integer object.

EXAMPLE 1 Integer objects

```
123    43445    +17    -98    0
```

A real value shall be written as one or more decimal digits with an optional sign and a leading, trailing, or embedded PERIOD (2Eh) (decimal point). The value shall be interpreted as a real number and shall be converted to a real object.

EXAMPLE 2 Real objects

```
34.5    -3.62    +123.6    4.    -.002    0.0
```

NOTE 1 A conforming writer shall not use the PostScript syntax for numbers with non-decimal radices (such as 16#FFFE) or in exponential format (such as 6.02E23).

NOTE 2 Throughout this standard, the term *number* refers to an object whose type may be either integer or real. Wherever a real number is expected, an integer may be used instead. For example, it is not necessary to write the number 1.0 in real format; the integer 1 is sufficient.

7.3.4 String Objects

7.3.4.1 General

A *string object* shall consist of a series of zero or more bytes. String objects are not integer objects, but are stored in a more compact format. The length of a string may be subject to implementation limits; see Annex C.

String objects shall be written in one of the following two ways:

- As a sequence of literal characters enclosed in parentheses ( ) (using LEFT PARENTHESIS (28h) and RIGHT PARENThESIS (29h)); see 7.3.4.2, "Literal Strings."

- As hexadecimal data enclosed in angle brackets < > (using LESS-THAN SIGN (3Ch) and GREATER-THAN SIGN (3Eh)); see 7.3.4.3, "Hexadecimal Strings."

NOTE In many contexts, conventions exist for the interpretation of the contents of a string value. This sub-clause defines only the basic syntax for writing a string as a sequence of bytes; conventions or rules governing the contents of strings in particular contexts are described with the definition of those particular contexts.

7.9.2, "String Object Types," describes the encoding schemes used for the contents of string objects.

7.3.4.2 Literal Strings

A *literal string* shall be written as an arbitrary number of characters enclosed in parentheses. Any characters may appear in a string except unbalanced parentheses (LEFT PARENHESIS (28h) and RIGHT PARENTHESIS (29h)) and the backslash (REVERSE SOLIDUS (5Ch)), which shall be treated specially as described in this sub-clause. Balanced pairs of parentheses within a string require no special treatment.

EXAMPLE 1 The following are valid literal strings:

```
(This is a string)
(Strings may contain newlines
and such.)
(Strings may contain balanced parentheses ( ) and
special characters (*!&}^% and so on).)
(The following is an empty string.)
()
(It has zero (0) length.)
```

Within a literal string, the REVERSE SOLIDUS is used as an escape character. The character immediately following the REVERSE SOLIDUS determines its precise interpretation as shown in Table 3. If the character following the REVERSE SOLIDUS is not one of those shown in Table 3, the REVERSE SOLIDUS shall be ignored.

Table 3 – Escape sequences in literal strings

Sequence	Meaning
\n	LINE FEED (0Ah) (LF)
\r	CARRIAGE RETURN (0Dh) (CR)
\t	HORIZONTAL TAB (09h) (HT)
\b	BACKSPACE (08h) (BS)
\f	FORM FEED (FF)
\(	LEFT PARENTHESIS (28h)
\)	RIGHT PARENTHESIS (29h)
\\\\	REVERSE SOLIDUS (5Ch) (Backslash)
\ddd	Character code *ddd* (octal)

A conforming writer may split a literal string across multiple lines. The REVERSE SOLIDUS (5Ch) (backslash character) at the end of a line shall be used to indicate that the string continues on the following line. A conforming reader shall disregard the REVERSE SOLIDUS and the end-of-line marker following it when reading the string; the resulting string value shall be identical to that which would be read if the string were not split.

EXAMPLE 2

```
(These \
two strings \
are the same.)
(These two strings are the same.)
```

An end-of-line marker appearing within a literal string without a preceding REVERSE SOLIDUS shall be treated as a byte value of (0Ah), irrespective of whether the end-of-line marker was a CARRIAGE RETURN (0Dh), a LINE FEED (0Ah), or both.

EXAMPLE 3
```
(This string has an end-of-line at the end of it.
)
(So does this one.\n)
```

The *ddd* escape sequence provides a way to represent characters outside the printable ASCII character set.

EXAMPLE 4
```
(This string contains \245two octal characters\307.)
```

The number *ddd* may consist of one, two, or three octal digits; high-order overflow shall be ignored. Three octal digits shall be used, with leading zeros as needed, if the next character of the string is also a digit.

EXAMPLE 5 the literal
```
(\0053)
```
denotes a string containing two characters, \005 (Control-E) followed by the digit 3, whereas both
```
(\053)
```
and
```
(\53)
```
denote strings containing the single character \053, a plus sign (+).

Since any 8-bit value may appear in a string (with proper escaping for REVERSE SOLIDUS (backslash) and unbalanced PARENTHESES) this \ddd notation provides a way to specify characters outside the ASCII character set by using ASCII characters only. However, any 8-bit value may appear in a string, represented either as itself or with the \ddd notation described.

When a document is encrypted (see 7.6, “Encryption”), all of its strings are encrypted; the encrypted string values contain arbitrary 8-bit values. When writing encrypted strings using the literal string form, the conforming writer shall follow the rules described. That is, the REVERSE SOLIDUS character shall be used as an escape to specify unbalanced PARENTHESES or the REVERSE SOLIDUS character itself. The REVERSE SOLIDUS may, but is not required, to be used to specify other, arbitrary 8-bit values.

7.3.4.3 Hexadecimal Strings

Strings may also be written in hexadecimal form, which is useful for including arbitrary binary data in a PDF file. A hexadecimal string shall be written as a sequence of hexadecimal digits (0–9 and either A–F or a–f) encoded as ASCII characters and enclosed within angle brackets (using LESS-THAN SIGN (3Ch) and GREATER-THAN SIGN (3Eh)).

EXAMPLE 1
```
<4E6F762073686D6F7A206B6120706F702E>
```

Each pair of hexadecimal digits defines one byte of the string. White-space characters (such as SPACE (20h), HORIZONTAL TAB (09h), CARRIAGE RETURN (0Dh), LINE FEED (0Ah), and FORM FEED (0Ch)) shall be ignored.

If the final digit of a hexadecimal string is missing—that is, if there is an odd number of digits—the final digit shall be assumed to be 0.

EXAMPLE 2
```
<901FA3>
```
is a 3-byte string consisting of the characters whose hexadecimal codes are 90, 1F, and A3, but
```
<901FA>
```
is a 3-byte string containing the characters whose hexadecimal codes are 90, 1F, and A0.

7.3.5 Name Objects

Beginning with PDF 1.2 a *name object* is an atomic symbol uniquely defined by a sequence of any characters (8-bit values) except null (character code 0). *Uniquely defined* means that any two name objects made up of the same sequence of characters denote the same object. *Atomic* means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not considered elements of the name.

When writing a name in a PDF file, a SOLIDUS (2Fh) (/) shall be used to introduce a name. The SOLIDUS is not part of the name but is a prefix indicating that what follows is a sequence of characters representing the name in the PDF file and shall follow these rules:

a) A NUMBER SIGN (23h) (#) in a name shall be written by using its 2-digit hexadecimal code (23), preceded by the NUMBER SIGN.

b) Any character in a name that is a regular character (other than NUMBER SIGN) shall be written as itself or by using its 2-digit hexadecimal code, preceded by the NUMBER SIGN.

c) Any character that is not a regular character shall be written using its 2-digit hexadecimal code, preceded by the NUMBER SIGN only.

NOTE 1 There is not a unique encoding of names into the PDF file because regular characters may be coded in either of two ways.

White space used as part of a name shall always be coded using the 2-digit hexadecimal notation and no white space may intervene between the SOLIDUS and the encoded name.

Regular characters that are outside the range EXCLAMATION MARK(21h) (!) to TILDE (7Eh) (~) should be written using the hexadecimal notation.

The token SOLIDUS (a slash followed by no regular characters) introduces a unique valid name defined by the empty sequence of characters.

NOTE 2 The examples shown in Table 4 and containing # are not valid literal names in PDF 1.0 or 1.1.

Table 4 – Examples of literal names

Syntax for Literal name	Resulting Name
/Name1	Name1
/ASomewhatLongerName	ASomewhatLongerName
/A;Name_With-Various***Characters?	A;Name_With-Various***Characters?
/1.2	1.2
/$$	$$
/@pattern	@pattern
/.notdef	.notdef
/lime#20Green	Lime Green
/paired#28#29parentheses	paired()parentheses
/The_Key_of_F#23_Minor	The_Key_of_F#_Minor
/A#42	AB

In PDF, literal names shall always be introduced by the SOLIDUS character (/), unlike keywords such as **true**, **false**, and **obj**.

NOTE 3 This standard follows a typographic convention of writing names without the leading SOLIDUS when they appear in running text and tables. For example, **Type** and **FullScreen** denote names that would actually be written in a PDF file (and in code examples in this standard) as **/Type** and **/FullScreen.**

The length of a name shall be subject to an implementation limit; see Annex C. The limit applies to the number of characters in the name's internal representation. For example, the name /A#20B has three characters (A, SPACE, B), not six.

As stated above, name objects shall be treated as atomic within a PDF file. Ordinarily, the bytes making up the name are never treated as text to be presented to a human user or to an application external to a conforming reader. However, occasionally the need arises to treat a name object as text, such as one that represents a font name (see the **BaseFont** entry in Table 111), a colorant name in a separation or DeviceN colour space, or a structure type (see 14.7.3, "Structure Types").

In such situations, the sequence of bytes (after expansion of NUMBER SIGN sequences, if any) should be interpreted according to UTF-8, a variable-length byte-encoded representation of Unicode in which the printable ASCII characters have the same representations as in ASCII. This enables a name object to represent text virtually in any natural language, subject to the implementation limit on the length of a name.

NOTE 4 PDF does not prescribe what UTF-8 sequence to choose for representing any given piece of externally specified text as a name object. In some cases, multiple UTF-8 sequences may represent the same logical text. Name objects defined by different sequences of bytes constitute distinct name objects in PDF, even though the UTF-8 sequences may have identical external interpretations.

7.3.6 Array Objects

An *array object* is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array's elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays. An array may have zero elements.

An array shall be written as a sequence of objects enclosed in SQUARE BRACKETS (using LEFT SQUARE BRACKET (5Bh) and RIGHT SQUARE BRACKET (5Dh)).

EXAMPLE
```
[549 3.14 false (Ralph) /SomeName]
```

PDF directly supports only one-dimensional arrays. Arrays of higher dimension can be constructed by using arrays as elements of arrays, nested to any depth.

7.3.7 Dictionary Objects

A *dictionary object* is an associative table containing pairs of objects, known as the dictionary's *entries*. The first element of each entry is the *key* and the second element is the *value*. The key shall be a name (unlike dictionary keys in PostScript, which may be objects of any type). The value may be any kind of object, including another dictionary. A dictionary entry whose value is **null** (see 7.3.9, "Null Object") shall be treated the same as if the entry does not exist. (This differs from PostScript, where **null** behaves like any other object as the value of a dictionary entry.) The number of entries in a dictionary shall be subject to an implementation limit; see Annex C. A dictionary may have zero entries.

The entries in a dictionary represent an associative table and as such shall be unordered even though an arbitrary order may be imposed upon them when written in a file. That ordering shall be ignored.

Multiple entries in the same dictionary shall not have the same key.

A dictionary shall be written as a sequence of key-value pairs enclosed in double angle brackets (<<…>>) (using LESS-THAN SIGNs (3Ch) and GREATER-THAN SIGNs (3Eh)).

EXAMPLE
```
<< /Type /Example
   /Subtype /DictionaryExample
   /Version 0.01
   /IntegerItem 12
   /StringItem (a string)
   /Subdictionary << /Item1 0.4
                     /Item2 true
                     /LastItem (not!)
                     /VeryLastItem (OK)
                  >>
>>
```

NOTE Do not confuse the double angle brackets with single angle brackets (< and >) (using LESS-THAN SIGN (3Ch) and GREATER-THAN SIGN (3Eh)), which delimit a hexadecimal string (see 7.3.4.3, "Hexadecimal Strings").

Dictionary objects are the main building blocks of a PDF document. They are commonly used to collect and tie together the attributes of a complex object, such as a font or a page of the document, with each entry in the dictionary specifying the name and value of an attribute. By convention, the **Type** entry of such a dictionary, if present, identifies the type of object the dictionary describes. In some cases, a **Subtype** entry (sometimes abbreviated **S**) may be used to further identify a specialized subcategory of the general type. The value of the **Type** or **Subtype** entry shall always be a name. For example, in a font dictionary, the value of the **Type** entry shall always be *Font*, whereas that of the **Subtype** entry may be *Type1*, *TrueType*, or one of several other values.

The value of the **Type** entry can almost always be inferred from context. The value of an entry in a page's font resource dictionary, for example, shall be a font object; therefore, the **Type** entry in a font dictionary serves primarily as documentation and as information for error checking. The **Type** entry shall not be required unless so stated in its description; however, if the entry is present, it shall have the correct value. In addition, the value of the **Type** entry in any dictionary, even in private data, shall be either a name defined in this standard or a registered name; see Annex E for details.

7.3.8 Stream Objects

7.3.8.1 General

A *stream object*, like a string object, is a sequence of bytes. Furthermore, a stream may be of unlimited length, whereas a string shall be subject to an implementation limit. For this reason, objects with potentially large amounts of data, such as images and page descriptions, shall be represented as streams.

NOTE 1 This sub-clause describes only the syntax for writing a stream as a sequence of bytes. The context in which a stream is referenced determines what the sequence of bytes represent.

A stream shall consist of a dictionary followed by zero or more bytes bracketed between the keywords **stream** (followed by newline) and **endstream**:

EXAMPLE
dictionary
stream
…*Zero or more bytes*…
endstream

All streams shall be indirect objects (see 7.3.10, "Indirect Objects") and the stream dictionary shall be a direct object. The keyword **stream** that follows the stream dictionary shall be followed by an end-of-line marker consisting of either a CARRIAGE RETURN and a LINE FEED or just a LINE FEED, and not by a CARRIAGE RETURN alone. The sequence of bytes that make up a stream lie between the end-of-line marker following the **stream** keyword and the **endstream** keyword; the stream dictionary specifies the exact number of bytes. There should be an end-of-line marker after the data and before **endstream**; this marker shall not be included in the stream length. There shall not be any extra bytes, other than white space, between endstream and endobj.

Alternatively, beginning with PDF 1.2, the bytes may be contained in an external file, in which case the stream dictionary specifies the file, and any bytes between **stream** and **endstream** shall be ignored by a conforming reader.

NOTE 2 Without the restriction against following the keyword **stream** by a CARRIAGE RETURN alone, it would be impossible to differentiate a stream that uses CARRIAGE RETURN as its end-of-line marker and has a LINE FEED as its first byte of data from one that uses a CARRIAGE RETURN–LINE FEED sequence to denote end-of-line.

Table 5 lists the entries common to all stream dictionaries; certain types of streams may have additional dictionary entries, as indicated where those streams are described. The optional entries regarding *filters* for the stream indicate whether and how the data in the stream shall be transformed (decoded) before it is used. Filters are described further in 7.4, "Filters."

7.3.8.2 Stream Extent

Every stream dictionary shall have a **Length** entry that indicates how many bytes of the PDF file are used for the stream's data. (If the stream has a filter, **Length** shall be the number of bytes of *encoded* data.) In addition, most filters are defined so that the data shall be self-limiting; that is, they use an encoding scheme in which an explicit *end-of-data* (EOD) marker delimits the extent of the data. Finally, streams are used to represent many objects from whose attributes a length can be inferred. All of these constraints shall be consistent.

EXAMPLE An image with 10 rows and 20 columns, using a single colour component and 8 bits per component, requires exactly 200 bytes of image data. If the stream uses a filter, there shall be enough bytes of encoded data in the PDF file to produce those 200 bytes. An error occurs if **Length** is too small, if an explicit EOD marker occurs too soon, or if the decoded data does not contain 200 bytes.

It is also an error if the stream contains too much data, with the exception that there may be an extra end-of-line marker in the PDF file before the keyword **endstream**.

Table 5 – Entries common to all stream dictionaries

Key	Type	Value
Length	integer	*(Required)* The number of bytes from the beginning of the line following the keyword **stream** to the last byte just before the keyword **endstream**. (There may be an additional EOL marker, preceding **endstream**, that is not included in the count and is not logically part of the stream data.) See 7.3.8.2, "Stream Extent", for further discussion.
Filter	name or array	*(Optional)* The name of a filter that shall be applied in processing the stream data found between the keywords **stream** and **endstream**, or an array of zero, one or several names. Multiple filters shall be specified in the order in which they are to be applied.
DecodeParms	dictionary or array	*(Optional)* A parameter dictionary or an array of such dictionaries, used by the filters specified by **Filter**. If there is only one filter and that filter has parameters, **DecodeParms** shall be set to the filter's parameter dictionary unless all the filter's parameters have their default values, in which case the **DecodeParms** entry may be omitted. If there are multiple filters and any of the filters has parameters set to nondefault values, **DecodeParms** shall be an array with one entry for each filter: either the parameter dictionary for that filter, or the null object if that filter has no parameters (or if all of its parameters have their default values). If none of the filters have parameters, or if all their parameters have default values, the **DecodeParms** entry may be omitted.
F	file specification	*(Optional; PDF 1.2)* The file containing the stream data. If this entry is present, the bytes between **stream** and **endstream** shall be ignored. However, the **Length** entry should still specify the number of those bytes (usually, there are no bytes and **Length** is 0). The filters that are applied to the file data shall be specified by **FFilter** and the filter parameters shall be specified by **FDecodeParms**.
FFilter	name or array	*(Optional; PDF 1.2)* The name of a filter to be applied in processing the data found in the stream's external file, or an array of zero, one or several such names. The same rules apply as for **Filter**.
FDecodeParms	dictionary or array	*(Optional; PDF 1.2)* A parameter dictionary, or an array of such dictionaries, used by the filters specified by **FFilter**. The same rules apply as for **DecodeParms**.

Table 5 – Entries common to all stream dictionaries (continued)

Key	Type	Value
DL	integer	*(Optional; PDF 1.5)* A non-negative integer representing the number of bytes in the decoded (defiltered) stream. It can be used to determine, for example, whether enough disk space is available to write a stream to a file. This value shall be considered a hint only; for some stream filters, it may not be possible to determine this value precisely.

7.3.9 Null Object

The *null object* has a type and value that are unequal to those of any other object. There shall be only one object of type null, denoted by the keyword **null**. An indirect object reference (see 7.3.10, "Indirect Objects") to a nonexistent object shall be treated the same as a null object. Specifying the null object as the value of a dictionary entry (7.3.7, "Dictionary Objects") shall be equivalent to omitting the entry entirely.

7.3.10 Indirect Objects

Any object in a PDF file may be labelled as an *indirect object*. This gives the object a unique *object identifier* by which other objects can refer to it (for example, as an element of an array or as the value of a dictionary entry). The object identifier shall consist of two parts:

- A positive integer *object number*. Indirect objects may be numbered sequentially within a PDF file, but this is not required; object numbers may be assigned in any arbitrary order.

- A non-negative integer *generation number*. In a newly created file, all indirect objects shall have generation numbers of 0. Nonzero generation numbers may be introduced when the file is later updated; see sub-clauses 7.5.4, "Cross-Reference Table" and 7.5.6, "Incremental Updates."

Together, the combination of an object number and a generation number shall uniquely identify an indirect object.

The definition of an indirect object in a PDF file shall consist of its object number and generation number (separated by white space), followed by the value of the object bracketed between the keywords **obj** and **endobj**.

EXAMPLE 1 Indirect object definition

```
12 0 obj
    (Brillig)
endobj
```

Defines an indirect string object with an object number of 12, a generation number of 0, and the value Brillig.

The object may be referred to from elsewhere in the file by an *indirect reference.* Such indirect references shall consist of the object number, the generation number, and the keyword **R** (with white space separating each part):

```
12 0 R
```

Beginning with PDF 1.5, indirect objects may reside in object streams (see 7.5.7, "Object Streams"). They are referred to in the same way; however, their definition shall not include the keywords **obj** and **endobj**, and their generation number shall be zero.

An indirect reference to an undefined object shall not be considered an error by a conforming reader; it shall be treated as a reference to the null object.

EXAMPLE 2 If a file contains the indirect reference 17 0 R but does not contain the corresponding definition then the indirect reference is considered to refer to the null object.

Except were documented to the contrary any object value may be a direct or an indirect reference; the semantics are equivalent.

EXAMPLE 3 The following shows the use of an indirect object to specify the length of a stream. The value of the stream's Length entry is an integer object that follows the stream in the file. This allows applications that generate PDF in a single pass to defer specifying the stream's length until after its contents have been generated.

```
7  0  obj
    <<  /Length  8 0 R  >>                         % An indirect reference to object 8
stream
    BT
        /F1  12  Tf
        72  712  Td
        (A stream with an indirect length)  Tj
    ET
endstream
endobj

8  0  obj
    77                                             % The length of the preceding stream
endobj
```

7.4 Filters

7.4.1 General

Stream filters are introduced in 7.3.8, "Stream Objects." An option when reading stream data is to decode it using a filter to produce the original non-encoded data. Whether to do so and which decoding filter or filters to use may be specified in the stream dictionary.

EXAMPLE 1 If a stream dictionary specifies the use of an ASCIIHexDecode filter, an application reading the data in that stream should transform the ASCII hexadecimal-encoded data in that stream in order to obtain the original binary data.

A conforming writer may encode data in a stream (for example, data for sampled images) to compress it or to convert it to a portable ASCII representation (or both). A conforming reader shall invoke the corresponding decoding filter or filters to convert the information back to its original form.

The filter or filters for a stream shall be specified by the **Filter** entry in the stream's dictionary (or the **FFilter** entry if the stream is external). Filters may be cascaded to form a *pipeline* that passes the stream through two or more decoding transformations in sequence. For example, data encoded using LZW and ASCII base-85 encoding (in that order) shall be decoded using the following entry in the stream dictionary:

EXAMPLE 2 /Filter [/ASCII85Decode /LZWDecode]

Some filters may take parameters to control how they operate. These optional parameters shall be specified by the **DecodeParms** entry in the stream's dictionary (or the **FDecodeParms** entry if the stream is external).

PDF supports a standard set of filters that fall into two main categories:

- *ASCII filters* enable decoding of arbitrary binary data that has been encoded as ASCII text (see 7.2, "Lexical Conventions," for an explanation of why this type of encoding might be useful).

- *Decompression filters* enable decoding of data that has been compressed. The compressed data shall be in binary format, even if the original data is ASCII text.

NOTE 1 ASCII filters serve no useful purpose in a PDF file that is encrypted; see 7.6, “Encryption”.

NOTE 2 Compression is particularly valuable for large sampled images, since it reduces storage requirements and transmission time. Some types of compression are lossy, meaning that some data is lost during the encoding, resulting in a loss of quality when the data is decompressed. Compression in which no loss of data occurs is called lossless. Though somehow obvious it might be worth pointing out that lossy compression can only be applied to sampled image data (and only certain types of lossy compression for certain types of images). Lossless compression on the other hand can be used for any kind of stream.

The standard filters are summarized in Table 6, which also indicates whether they accept any optional parameters. The following sub-clauses describe these filters and their parameters (if any) in greater detail, including specifications of encoding algorithms for some filters.

Table 6 – Standard filters

FILTER name	Parameters	Description
ASCIIHexDecode	no	Decodes data encoded in an ASCII hexadecimal representation, reproducing the original binary data.
ASCII85Decode	no	Decodes data encoded in an ASCII base-85 representation, reproducing the original binary data.
LZWDecode	yes	Decompresses data encoded using the LZW (Lempel-Ziv-Welch) adaptive compression method, reproducing the original text or binary data.
FlateDecode	yes	*(PDF 1.2)* Decompresses data encoded using the zlib/deflate compression method, reproducing the original text or binary data.
RunLengthDecode	no	Decompresses data encoded using a byte-oriented run-length encoding algorithm, reproducing the original text or binary data (typically monochrome image data, or any data that contains frequent long runs of a single byte value).
CCITTFaxDecode	yes	Decompresses data encoded using the CCITT facsimile standard, reproducing the original data (typically monochrome image data at 1 bit per pixel).
JBIG2Decode	yes	*(PDF 1.4)* Decompresses data encoded using the JBIG2 standard, reproducing the original monochrome (1 bit per pixel) image data (or an approximation of that data).
DCTDecode	yes	Decompresses data encoded using a DCT (discrete cosine transform) technique based on the JPEG standard, reproducing image sample data that approximates the original data.
JPXDecode	no	*(PDF 1.5)* Decompresses data encoded using the wavelet-based JPEG2000 standard, reproducing the original image data.
Crypt	yes	*(PDF 1.5)* Decrypts data encrypted by a security handler, reproducing the data as it was before encryption.

EXAMPLE 3 The following example shows a stream, containing the marking instructions for a page, that was compressed using the LZW compression method and then encoded in ASCII base-85 representation.

```
1 0 obj
   << /Length 534
      /Filter [/ASCII85Decode /LZWDecode]
   >>
```

```
stream
J..)6T`?p&<!J9%_[umg"B7/Z7KNXbN'S+,*Q/&"OLT'F
LIDK#!n`$"<Atdi`\Vn%b%)&'cA*VnK\CJY(sF>c!Jnl@
RM]WM;jjH6Gnc75idkL5]+cPZKEBPWdR>FF(kj1_R%W_d
&/jS!;iuad7h?[L-F$+]]0A3Ck*$I0KZ?;<)CJtqi65Xb
Vc3\n5ua:Q/=0$W<#N3U;H,MQKqfg1?:lUpR;6oN[C2E4
ZNr8Udn.'p+?#X+1>0Kuk$bCDF/(3fL5]Oq)^kJZ!C2H1
'TO]Rl?Q:&'<5&iP!$Rq;BXRecDN[IJB`,)o8XJOSJ9sD
S]hQ;Rj@!ND)bD_q&C\g:inYC%)&u#:u,M6Bm%IY!Kb1+
":aAa'S`ViJglLb8<W9k6Yl\\0McJQkDeLWdPN?9A'jX*
al>iG1p&i;eVoK&juJHs9%;Xomop"5KatWRT"JQ#qYuL,
JD?M$0QP)lKn06l1apKDC@\qJ4B!!(5m+j.7F790m(Vj8
8l8Q:_CZ(Gm1%X\N1&u!FKHMB~>
endstream
endobj
```

EXAMPLE 4 The following shows the same stream without any filters applied to it. (The stream's contents are explained in 7.8.2, "Content Streams," and the operators used there are further described in clause 9, "Text".)

```
1 0 obj
   << /Length 568 >>
stream
2 J
BT
/F1 12 Tf
0 Tc
0 Tw
72.5 712 TD
[(Unfiltered streams can be read easily) 65 (, )] TJ
0 -14 TD
[(b) 20 (ut generally tak) 10 (e more space than \311)] TJ
T* (compressed streams.) Tj
0 -28 TD
[(Se) 25 (v) 15 (eral encoding methods are a) 20 (v) 25 (ailable in PDF) 80 (.)] TJ
0 -14 TD
(Some are used for compression and others simply) Tj
T* [(to represent binary data in an ) 55 (ASCII format.)] TJ
T* (Some of the compression filters are \
suitable ) Tj
T* (for both data and images, while others are \
suitable only ) Tj
T* (for continuous-tone images.) Tj
ET
endstream
endobj
```

7.4.2 ASCIIHexDecode Filter

The **ASCIIHexDecode** filter decodes data that has been encoded in ASCII hexadecimal form. ASCII hexadecimal encoding and ASCII base-85 encoding (7.4.3, "ASCII85Decode Filter") convert binary data, such as image data or previously compressed data, to 7-bit ASCII characters.

NOTE ASCII base-85 encoding is preferred to ASCII hexadecimal encoding. Base-85 encoding is preferred because it is more compact: it expands the data by a factor of 4:5, compared with 1:2 for ASCII hexadecimal encoding.

The **ASCIIHexDecode** filter shall produce one byte of binary data for each pair of ASCII hexadecimal digits (0–9 and A–F or a–f). All white-space characters (see 7.2, "Lexical Conventions") shall be ignored. A GREATER-THAN SIGN (3Eh) indicates EOD. Any other characters shall cause an error. If the filter encounters the EOD marker after reading an odd number of hexadecimal digits, it shall behave as if a 0 (zero) followed the last digit.

7.4.3 ASCII85Decode Filter

The **ASCII85Decode** filter decodes data that has been encoded in ASCII base-85 encoding and produces binary data. The following paragraphs describe the process for encoding binary data in ASCII base-85; the **ASCII85Decode** filter reverses this process.

The ASCII base-85 encoding shall use the ASCII characters ! through u ((21h) - (75h)) and the character z (7Ah), with the 2-character sequence ~> (7Eh)(3Eh) as its EOD marker. The **ASCII85Decode** filter shall ignore all white-space characters (see 7.2, "Lexical Conventions"). Any other characters, and any character sequences that represent impossible combinations in the ASCII base-85 encoding shall cause an error.

Specifically, ASCII base-85 encoding shall produce 5 ASCII characters for every 4 bytes of binary data. Each group of 4 binary input bytes, (b_1 b_2 b_3 b_4), shall be converted to a group of 5 output bytes, (c_1 c_2 c_3 c_4 c_5), using the relation

$$(b_1 \times 256^3) + (b_2 \times 256^2) + (b_3 \times 256^1) + b_4 =$$
$$(c_1 \times 85^4) + (c_2 \times 85^3) + (c_3 \times 85^2) + (c_4 \times 85^1) + c$$

In other words, 4 bytes of binary data shall be interpreted as a base-256 number and then shall be converted to a base-85 number. The five bytes of the base-85 number shall then be converted to ASCII characters by adding 33 (the ASCII code for the character !) to each. The resulting encoded data shall contain only printable ASCII characters with codes in the range 33 (!) to 117 (u). As a special case, if all five bytes are 0, they shall be represented by the character with code 122 (z) instead of by five exclamation points (!!!!!).

If the length of the data to be encoded is not a multiple of 4 bytes, the last, partial group of 4 shall be used to produce a last, partial group of 5 output characters. Given *n* (1, 2, or 3) bytes of binary data, the encoder shall first append 4 - *n* zero bytes to make a complete group of 4. It shall encode this group in the usual way, but shall not apply the special z case. Finally, it shall write only the first *n* + 1 characters of the resulting group of 5. These characters shall be immediately followed by the ~> EOD marker.

The following conditions shall never occur in a correctly encoded byte sequence:

- The value represented by a group of 5 characters is greater than 2^{32} - 1.
- A z character occurs in the middle of a group.
- A final partial group contains only one character.

7.4.4 LZWDecode and FlateDecode Filters

7.4.4.1 General

The **LZWDecode** and (*PDF 1.2*) **FlateDecode** filters have much in common and are discussed together in this sub-clause. They decode data that has been encoded using the LZW or Flate data compression method, respectively:

- LZW (Lempel-Ziv-Welch) is a variable-length, adaptive compression method that has been adopted as one of the standard compression methods in the *Tag Image File Format* (TIFF) standard. For details on LZW encoding see 7.4.4.2, "Details of LZW Encoding."
- The Flate method 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. It is fully defined in Internet RFCs 1950, ZLIB Compressed Data Format Specification, and 1951, DEFLATE Compressed Data Format Specification (see the Bibliography).

Both of these methods compress either binary data or ASCII text but (like all compression methods) always produce binary data, even if the original data was text.

The LZW and Flate compression methods can discover and exploit many patterns in the input data, whether the data is text or images. As described later, both filters support optional transformation by a *predictor function*, which improves the compression of sampled image data.

NOTE 1 Because of its cascaded adaptive Huffman coding, Flate-encoded output is usually much more compact than LZW-encoded output for the same input. Flate and LZW decoding speeds are comparable, but Flate encoding is considerably slower than LZW encoding.

NOTE 2 Usually, both Flate and LZW encodings compress their input substantially. However, in the worst case (in which no pair of adjacent bytes appears twice), Flate encoding expands its input by no more than 11 bytes or a factor of 1.003 (whichever is larger), plus the effects of algorithm tags added by PNG predictors. For LZW encoding, the best case (all zeros) provides a compression approaching 1365 : 1 for long files, but the worst-case expansion is at least a factor of 1.125, which can increase to nearly 1.5 in some implementations, plus the effects of PNG tags as with Flate encoding.

7.4.4.2 Details of LZW Encoding

Data encoded using the LZW compression method shall consist of a sequence of codes that are 9 to 12 bits long. Each code shall represent a single character of input data (0–255), a clear-table marker (256), an EOD marker (257), or a table entry representing a multiple-character sequence that has been encountered previously in the input (258 or greater).

Initially, the code length shall be 9 bits and the LZW table shall contain only entries for the 258 fixed codes. As encoding proceeds, entries shall be appended to the table, associating new codes with longer and longer sequences of input characters. The encoder and the decoder shall maintain identical copies of this table.

Whenever both the encoder and the decoder independently (but synchronously) realize that the current code length is no longer sufficient to represent the number of entries in the table, they shall increase the number of bits per code by 1. The first output code that is 10 bits long shall be the one following the creation of table entry 511, and similarly for 11 (1023) and 12 (2047) bits. Codes shall never be longer than 12 bits; therefore, entry 4095 is the last entry of the LZW table.

The encoder shall execute the following sequence of steps to generate each output code:

a) Accumulate a sequence of one or more input characters matching a sequence already present in the table. For maximum compression, the encoder looks for the longest such sequence.

b) Emit the code corresponding to that sequence.

c) Create a new table entry for the first unused code. Its value is the sequence found in step (a) followed by the next input character.

EXAMPLE 1 Suppose the input consists of the following sequence of ASCII character codes:
45 45 45 45 45 65 45 45 45 66

Starting with an empty table, the encoder proceeds as shown in Table 7.

Table 7 – Typical LZW encoding sequence

Input sequence	Output code	Code added to table	Sequence represented by new code
–	256 (clear-table)	–	–
45	45	258	45 45
45 45	258	259	45 45 45

Table 7 – Typical LZW encoding sequence (continued)

Input sequence	Output code	Code added to table	Sequence represented by new code
45 45	258	260	45 45 65
65	65	261	65 45
45 45 45	259	262	45 45 45 66
66	66	–	–
–	257 (EOD)	–	–

Codes shall be packed into a continuous bit stream, high-order bit first. This stream shall then be divided into bytes, high-order bit first. Thus, codes may straddle byte boundaries arbitrarily. After the EOD marker (code value 257), any leftover bits in the final byte shall be set to 0.

In the example above, all the output codes are 9 bits long; they would pack into bytes as follows (represented in hexadecimal):

EXAMPLE 2 80 0B 60 50 22 0C 0C 85 01

To adapt to changing input sequences, the encoder may at any point issue a clear-table code, which causes both the encoder and the decoder to restart with initial tables and a 9-bit code length. The encoder shall begin by issuing a clear-table code. It shall issue a clear-table code when the table becomes full; it may do so sooner.

7.4.4.3 LZWDecode and FlateDecode Parameters

The **LZWDecode** and **FlateDecode** filters shall accept optional parameters to control the decoding process.

NOTE Most of these parameters are related to techniques that reduce the size of compressed sampled images (rectangular arrays of colour values, described in 8.9, "Images"). For example, image data typically changes very little from sample to sample. Therefore, subtracting the values of adjacent samples (a process called differencing), and encoding the differences rather than the raw sample values, can reduce the size of the output data. Furthermore, when the image data contains several colour components (red-green-blue or cyan-magenta-yellow-black) per sample, taking the difference between the values of corresponding components in adjacent samples, rather than between different colour components in the same sample, often reduces the output data size.

Table 8 shows the parameters that may optionally be specified for **LZWDecode** and **FlateDecode** filters. Except where otherwise noted, all values supplied to the decoding filter for any optional parameters shall match those used when the data was encoded.

Table 8 – Optional parameters for LZWDecode and FlateDecode filters

Key	Type	Value
Predictor	integer	A code that selects the predictor algorithm, if any. If the value of this entry is 1, the filter shall assume that the normal algorithm was used to encode the data, without prediction. If the value is greater than 1, the filter shall assume that the data was differenced before being encoded, and **Predictor** selects the predictor algorithm. For more information regarding **Predictor** values greater than 1, see 7.4.4.4, "LZW and Flate Predictor Functions." Default value: 1.
Colors	integer	*(May be used only if **Predictor** is greater than 1)* The number of interleaved colour components per sample. Valid values are 1 to 4 (*PDF 1.0*) and 1 or greater (*PDF 1.3*). Default value: 1.

Table 8 – Optional parameters for LZWDecode and FlateDecode filters (continued)

Key	Type	Value
BitsPerComponent	integer	*(May be used only if **Predictor** is greater than 1)* The number of bits used to represent each colour component in a sample. Valid values are 1, 2, 4, 8, and (*PDF 1.5*) 16. Default value: 8.
Columns	integer	*(May be used only if **Predictor** is greater than 1)* The number of samples in each row. Default value: 1.
EarlyChange	integer	(***LZWDecode** only)* An indication of when to increase the code length. If the value of this entry is 0, code length increases shall be postponed as long as possible. If the value is 1, code length increases shall occur one code early. This parameter is included because LZW sample code distributed by some vendors increases the code length one code earlier than necessary. Default value: 1.

7.4.4.4 LZW and Flate Predictor Functions

LZW and Flate encoding compress more compactly if their input data is highly predictable. One way of increasing the predictability of many continuous-tone sampled images is to replace each sample with the difference between that sample and a predictor function applied to earlier neighboring samples. If the predictor function works well, the postprediction data clusters toward 0.

PDF supports two groups of predictor functions. The first, the *TIFF* group, consists of the single function that is Predictor 2 in the TIFF 6.0 specification.

NOTE 1 (In the TIFF 6.0 specification, Predictor 2 applies only to LZW compression, but here it applies to Flate compression as well.) TIFF Predictor 2 predicts that each colour component of a sample is the same as the corresponding colour component of the sample immediately to its left.

The second supported group of predictor functions, the *PNG* group, consists of the filters of the World Wide Web Consortium’s Portable Network Graphics recommendation, documented in Internet RFC 2083, PNG (Portable Network Graphics) Specification (see the Bibliography).

The term predictors is used here instead of filters to avoid confusion.

There are five basic PNG predictor algorithms (and a sixth that chooses the optimum predictor function separately for each row).

Table 9 – PNG predictor algorithms

PNG Predictor Algorithms	Description
None	No prediction
Sub	Predicts the same as the sample to the left
Up	Predicts the same as the sample above
Average	Predicts the average of the sample to the left and the sample above
Paeth	A nonlinear function of the sample above, the sample to the left, and the sample to the upper left

The predictor algorithm to be used, if any, shall be indicated by the **Predictor** filter parameter (see Table 8), whose value shall be one of those listed in Table 10.

For **LZWDecode** and **FlateDecode**, a **Predictor** value greater than or equal to 10 shall indicate that a PNG predictor is in use; the specific predictor function used shall be explicitly encoded in the incoming data. The value of **Predictor** supplied by the decoding filter need not match the value used when the data was encoded if they are both greater than or equal to 10.

Table 10 – Predictor values

Value	Meaning
1	No prediction (the default value)
2	TIFF Predictor 2
10	PNG prediction (on encoding, PNG None on all rows)
11	PNG prediction (on encoding, PNG Sub on all rows)
12	PNG prediction (on encoding, PNG Up on all rows)
13	PNG prediction (on encoding, PNG Average on all rows)
14	PNG prediction (on encoding, PNG Paeth on all rows)
15	PNG prediction (on encoding, PNG optimum)

The two groups of predictor functions have some commonalities. Both make the following assumptions:

- Data shall be presented in order, from the top row to the bottom row and, within a row, from left to right.
- A row shall occupy a whole number of bytes, rounded up if necessary.
- Samples and their components shall be packed into bytes from high-order to low-order bits.
- All colour components of samples outside the image (which are necessary for predictions near the boundaries) shall be 0.

The predictor function groups also differ in significant ways:

- The postprediction data for each PNG-predicted row shall begin with an explicit algorithm tag; therefore, different rows can be predicted with different algorithms to improve compression. TIFF Predictor 2 has no such identifier; the same algorithm applies to all rows.
- The TIFF function group shall predict each colour component from the prior instance of that component, taking into account the number of bits per component and components per sample. In contrast, the PNG function group shall predict each byte of data as a function of the corresponding byte of one or more previous image samples, regardless of whether there are multiple colour components in a byte or whether a single colour component spans multiple bytes.

NOTE 2 This can yield significantly better speed at the cost of somewhat worse compression.

7.4.5 RunLengthDecode Filter

The **RunLengthDecode** filter decodes data that has been encoded in a simple byte-oriented format based on run length. The encoded data shall be a sequence of *runs*, where each run shall consist of a *length* byte followed by 1 to 128 bytes of data. If the *length* byte is in the range 0 to 127, the following *length* + 1 (1 to 128) bytes shall be copied literally during decompression. If *length* is in the range 129 to 255, the following single byte shall be copied 257 - *length* (2 to 128) times during decompression. A *length* value of 128 shall denote EOD.

NOTE The compression achieved by run-length encoding depends on the input data. In the best case (all zeros), a compression of approximately 64 : 1 is achieved for long files. The worst case (the hexadecimal sequence 00 alternating with FF) results in an expansion of 127 : 128.

7.4.6 CCITTFaxDecode Filter

The **CCITTFaxDecode** filter decodes image data that has been encoded using either Group 3 or Group 4 CCITT facsimile (fax) encoding.

NOTE 1 CCITT encoding is designed to achieve efficient compression of monochrome (1 bit per pixel) image data at relatively low resolutions, and so is useful only for bitmap image data, not for colour images, grayscale images, or general data.

NOTE 2 The CCITT encoding standard is defined by the International Telecommunications Union (ITU), formerly known as the Comité Consultatif International Téléphonique et Télégraphique (International Coordinating Committee for Telephony and Telegraphy). The encoding algorithm is not described in detail in this standard but can be found in ITU Recommendations T.4 and T.6 (see the Bibliography). For historical reasons, we refer to these documents as the CCITT standard.

CCITT encoding is bit-oriented, not byte-oriented. Therefore, in principle, encoded or decoded data need not end at a byte boundary. This problem shall be dealt with in the following ways:

- Unencoded data shall be treated as complete scan lines, with unused bits inserted at the end of each scan line to fill out the last byte. This approach is compatible with the PDF convention for sampled image data.
- Encoded data shall ordinarily be treated as a continuous, unbroken bit stream. The **EncodedByteAlign** parameter (described in Table 11) may be used to cause each encoded scan line to be filled to a byte boundary.

NOTE 3 Although this is not prescribed by the CCITT standard and fax machines never do this, some software packages find it convenient to encode data this way.

- When a filter reaches EOD, it shall always skip to the next byte boundary following the encoded data.

The filter shall not perform any error correction or resynchronization, except as noted for the **DamagedRowsBeforeError** parameter in Table 11.

Table 11 lists the optional parameters that may be used to control the decoding. Except where noted otherwise, all values supplied to the decoding filter by any of these parameters shall match those used when the data was encoded.

Table 11 – Optional parameters for the CCITTFaxDecode filter

Key	Type	Value
K	integer	A code identifying the encoding scheme used: <0 Pure two-dimensional encoding (Group 4) $=0$ Pure one-dimensional encoding (Group 3, 1-D) >0 Mixed one- and two-dimensional encoding (Group 3, 2-D), in which a line encoded one-dimensionally may be followed by at most **K** − 1 lines encoded two-dimensionally The filter shall distinguish among negative, zero, and positive values of **K** to determine how to interpret the encoded data; however, it shall not distinguish between different positive **K** values. Default value: 0.

Table 11 – Optional parameters for the CCITTFaxDecode filter (continued)

Key	Type	Value
EndOfLine	boolean	A flag indicating whether end-of-line bit patterns shall be present in the encoding. The **CCITTFaxDecode** filter shall always accept end-of-line bit patterns. If EndOfLine is **true** end-of-line bit patterns shall be present.Default value: **false**.
EncodedByteAlign	boolean	A flag indicating whether the filter shall expect extra 0 bits before each encoded line so that the line begins on a byte boundary. If **true**, the filter shall skip over encoded bits to begin decoding each line at a byte boundary. If **false**, the filter shall not expect extra bits in the encoded representation. Default value: **false**.
Columns	integer	The width of the image in pixels. If the value is not a multiple of 8, the filter shall adjust the width of the unencoded image to the next multiple of 8 so that each line starts on a byte boundary. Default value: 1728.
Rows	integer	The height of the image in scan lines. If the value is 0 or absent, the image's height is not predetermined, and the encoded data shall be terminated by an end-of-block bit pattern or by the end of the filter's data. Default value: 0.
EndOfBlock	boolean	A flag indicating whether the filter shall expect the encoded data to be terminated by an end-of-block pattern, overriding the **Rows** parameter. If **false**, the filter shall stop when it has decoded the number of lines indicated by **Rows** or when its data has been exhausted, whichever occurs first. The end-of-block pattern shall be the CCITT end-of-facsimile-block (EOFB) or return-to-control (RTC) appropriate for the **K** parameter. Default value: **true**.
BlackIs1	boolean	A flag indicating whether 1 bits shall be interpreted as black pixels and 0 bits as white pixels, the reverse of the normal PDF convention for image data. Default value: **false**.
DamagedRowsBeforeError	integer	The number of damaged rows of data that shall be tolerated before an error occurs. This entry shall apply only if **EndOfLine** is **true** and **K** is non-negative. Tolerating a damaged row shall mean locating its end in the encoded data by searching for an **EndOfLine** pattern and 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.

NOTE 4 The compression achieved using CCITT encoding depends on the data, as well as on the value of various optional parameters. For Group 3 one-dimensional encoding, in the best case (all zeros), each scan line compresses to 4 bytes, and the compression factor depends on the length of a scan line. If the scan line is 300 bytes long, a compression ratio of approximately 75:1 is achieved. The worst case, an image of alternating ones and zeros, produces an expansion of 2:9.

7.4.7 JBIG2Decode Filter

The **JBIG2Decode** filter *(PDF 1.4)* decodes monochrome (1 bit per pixel) image data that has been encoded using JBIG2 encoding.

NOTE 1 JBIG stands for the Joint Bi-Level Image Experts Group, a group within the International Organization for Standardization (ISO) that developed the format. JBIG2 is the second version of a standard originally released as JBIG1.

JBIG2 encoding, which provides for both lossy and lossless compression, is useful only for monochrome images, not for colour images, grayscale images, or general data. The algorithms used by the encoder, and the details of the format, are not described here. See ISO/IEC 11544 published standard for the current JBIG2 specification. Additional information can be found through the Web site for the JBIG and JPEG (Joint Photographic Experts Group) committees at <http://www.jpeg.org>.

In general, JBIG2 provides considerably better compression than the existing CCITT standard (discussed in 7.4.6, "CCITTFaxDecode Filter"). The compression it achieves depends strongly on the nature of the image. Images of pages containing text in any language compress particularly well, with typical compression ratios of 20:1 to 50:1 for a page full of text.

The JBIG2 encoder shall build a table of unique symbol bitmaps found in the image, and other symbols found later in the image shall be matched against the table. Matching symbols shall be replaced by an index into the table, and symbols that fail to match shall be added to the table. The table itself shall be compressed using other means.

NOTE 2 This method results in high compression ratios for documents in which the same symbol is repeated often, as is typical for images created by scanning text pages. It also results in high compression of white space in the image, which does not need to be encoded because it contains no symbols.

While best compression is achieved for images of text, the JBIG2 standard also includes algorithms for compressing regions of an image that contain dithered halftone images (for example, photographs).

The JBIG2 compression method may also be used for encoding multiple images into a single JBIG2 bit stream.

NOTE 3 Typically, these images are scanned pages of a multiple-page document. Since a single table of symbol bitmaps is used to match symbols across multiple pages, this type of encoding can result in higher compression ratios than if each of the pages had been individually encoded using JBIG2.

In general, an image may be specified in PDF as either an *image XObject* or an *inline image* (as described in 8.9, "Images"); however, the **JBIG2Decode** filter shall not be used with inline images.

This filter addresses both single-page and multiple-page JBIG2 bit streams by representing each JBIG2 page as a PDF image, as follows:

- The filter shall use the embedded file organization of JBIG2. (The details of this and the other types of file organization are provided in an annex of the ISO specification.) The optional 2-byte combination (marker) mentioned in the specification shall not be used in PDF. JBIG2 bit streams in random-access organization should be converted to the embedded file organization. Bit streams in sequential organization need no reorganization, except for the mappings described below.

- The JBIG2 file header, end-of-page segments, and end-of-file segment shall not be used in PDF. These should be removed before the PDF objects described below are created.

- The image XObject to which the **JBIG2Decode** filter is applied shall contain all segments that are associated with the JBIG2 page represented by that image; that is, all segments whose segment page association field contains the page number of the JBIG2 page represented by the image. In the image XObject, however, the segment's page number should always be 1; that is, when each such segment is written to the XObject, the value of its segment page association field should be set to 1.

- If the bit stream contains global segments (segments whose segment page association field contains 0), these segments shall be placed in a separate PDF stream, and the filter parameter listed in Table 12 should refer to that stream. The stream can be shared by multiple image XObjects whose JBIG2 encodings use the same global segments.

Table 12 – Optional parameter for the JBIG2Decode filter

Key	Type	Value
JBIG2Globals	stream	A stream containing the JBIG2 global (page 0) segments. Global segments shall be placed in this stream even if only a single JBIG2 image XObject refers to it.

EXAMPLE 1 The following shows an image that was compressed using the JBIG2 compression method and then encoded in ASCII hexadecimal representation. Since the JBIG2 bit stream contains global segments, these segments are placed in a separate PDF stream, as indicated by the JBIG2Globals filter parameter.

```
5 0 obj
    << /Type /XObject
        /Subtype /Image
        /Width 52
        /Height 66
        /ColorSpace /DeviceGray
        /BitsPerComponent 1
        /Length 224
        /Filter [/ASCIIHexDecode /JBIG2Decode]
        /DecodeParms [null << /JBIG2Globals 6 0 R >>]
    >>
stream
000000013000010000001300000034000000420000000000
00000040000000000000206200001000000 1e000000340000
00420000000000000000200100000000231db51ce51ffac>
endstream
endobj

6 0 obj
    << /Length 126
        /Filter /ASCIIHexDecode
    >>
stream
0000000000010000000032000003fffdff02fefefe000000
01000000012ae225aea9a5a538b4d9999c5c8e56ef0f872
7f2b53d4e37ef795cc5506dffac>
endstream
endobj
```

The JBIG2 bit stream for this example is as follows:

EXAMPLE 2

```
97 4A 42 32 0D 0A 1A 0A 01 00 00 00 01 00 00 00 00 00 01 00 00 00 00 32
00 00 03 FF FD FF 02 FE FE FE 00 00 00 01 00 00 00 01 2A E2 25 AE A9 A5
A5 38 B4 D9 99 9C 5C 8E 56 EF 0F 87 27 F2 B5 3D 4E 37 EF 79 5C C5 50 6D
FF AC 00 00 00 01 30 00 01 00 00 00 13 00 00 00 34 00 00 00 42 00 00 00
00 00 00 00 00 40 00 00 00 00 00 02 06 20 00 01 00 00 00 1E 00 00 00 34
00 00 00 42 00 00 00 00 00 00 00 00 02 00 10 00 00 00 02 31 DB 51 CE 51
FF AC 00 00 00 03 31 00 01 00 00 00 00 00 00 00 04 33 01 00 00 00 00
```

This bit stream is made up of the following parts (in the order listed):

a) The JBIG2 file header

```
97 4A 42 32 0D 0A 1A 0A 01 00 00 00 01
```

Since the JBIG2 file header shall not used in PDF, this header is not placed in the JBIG2 stream object and is discarded.

b) The first JBIG2 segment (segment 0)—in this case, the symbol dictionary segment

```
00 00 00 00 00 01 00 00 00 00 32 00 00 03 FF FD FF 02 FE FE FE 00 00 00
01 00 00 00 01 2A E2 25 AE A9 A5 A5 38 B4 D9 99 9C 5C 8E 56 EF 0F 87
27 F2 B5 3D 4E 37 EF 79 5C C5 50 6D FF AC
```

This is a global segment (segment page association = 0) and so shall be placed in the **JBIG2Globals** stream.

c) The page information segment

```
00 00 00 01 30 00 01 00 00 00 13 00 00 00 34 00 00 00 42 00 00 00 00
00 00 00 00 40 00 00
```

and the immediate text region segment

```
00 00 00 02 06 20 00 01 00 00 00 1E 00 00 00 34 00 00 00 42 00 00 00
00 00 00 00 00 02 00 10 00 00 00 02 31 DB 51 CE 51 FF AC
```

These two segments constitute the contents of the JBIG2 page and shall be placed in the PDF XObject representing this image.

d) The end-of-page segment

```
00 00 00 03 31 00 01 00 00 00 00
```

and the end-of-file segment

```
00 00 00 04 33 01 00 00 00 00
```

Since these segments shall not be used in PDF, they are discarded.

The resulting PDF image object, then, contains the page information segment and the immediate text region segment and refers to a **JBIG2Globals** stream that contains the symbol dictionary segment.

7.4.8 DCTDecode Filter

The **DCTDecode** filter decodes grayscale or colour image data that has been encoded in the JPEG baseline format. See Adobe Technical Note #5116 for additional information about the use of JPEG “markers.”

NOTE 1 JPEG stands for the Joint Photographic Experts Group, a group within the International Organization for Standardization that developed the format; DCT stands for discrete cosine transform, the primary technique used in the encoding.

JPEG encoding is a lossy compression method, designed specifically for compression of sampled continuous-tone images and not for general data compression.

Data to be encoded using JPEG shall consist of a stream of image samples, each consisting of one, two, three, or four colour components. The colour component values for a particular sample shall appear consecutively. Each component value shall occupy a byte.

During encoding, several parameters shall control the algorithm and the information loss. The values of these parameters, which include the dimensions of the image and the number of components per sample, are entirely under the control of the encoder and shall be stored in the encoded data. **DCTDecode** may obtain the parameter values it requires directly from the encoded data. However, in one instance, the parameter need not be present in the encoded data but shall be specified in the filter parameter dictionary; see Table 13.

NOTE 2 The details of the encoding algorithm are not presented here but are in the ISO standard and in JPEG: Still Image Data Compression Standard, by Pennebaker and Mitchell (see the Bibliography). Briefly, the JPEG algorithm breaks an image up into blocks that are 8 samples wide by 8 samples high. Each colour component in an image is treated separately. A two-dimensional DCT is performed on each block. This operation produces 64 coefficients, which are then quantized. Each coefficient may be quantized with a different step size. It is this quantization that results in the loss of information in the JPEG algorithm. The quantized coefficients are then compressed.

Table 13 – Optional parameter for the DCTDecode filter

Key	Type	Value
ColorTransform	integer	(*Optional*) A code specifying the transformation that shall be performed on the sample values: 0 No transformation. 1 If the image has three colour components, *RGB* values shall be transformed to *YUV* before encoding and from *YUV* to *RGB* after decoding. If the image has four components, *CMYK* values shall be transformed to *YUVK* before encoding and from *YUVK* to *CMYK* after decoding. This option shall be ignored if the image has one or two colour components. If the encoding algorithm has inserted the Adobe-defined marker[a] code in the encoded data indicating the ColorTransform value, then the colours shall be transformed, or not, after the DCT decoding has been performed according to the value provided in the encoded data and the value of this dictionary entry shall be ignored. If the Adobe-defined marker code in the encoded data indicating the ColorTransform value is not present then the value specified in this dictionary entry will be used. If the Adobe-defined marker code in the encoded data indicating the ColorTransform value is not present and this dictionary entry is not present in the filter dictionary then the default value of ColorTransform shall be 1 if the image has three components and 0 otherwise.

[a] Parameters that control the decoding process as well as other metadata is embedded within the encoded data stream using a notation referred to as “markers”. When it defined the use of JPEG images within PostScript data streams, Adobe System Incorporated defined a particular set of rules pertaining to which markers are to be recognized, which are to be ignored and which are considered errors. A specific Adobe-defined marker was also introduced. The exact rules for producing and consuming DCT encoded data within PostScript are provide in Adobe Technical Note #5116 (reference). PDF DCT Encoding shall exactly follow those rules established by Adobe for PostScript.

NOTE 3 The encoding algorithm can reduce the information loss by making the step size in the quantization smaller at the expense of reducing the amount of compression achieved by the algorithm. The compression achieved by the JPEG algorithm depends on the image being compressed and the amount of loss that is acceptable. In general, a compression of 15:1 can be achieved without perceptible loss of information, and 30:1 compression causes little impairment of the image.

NOTE 4 Better compression is often possible for colour spaces that treat luminance and chrominance separately than for those that do not. The RGB-to-YUV conversion provided by the filters is one attempt to separate luminance and chrominance; it conforms to CCIR recommendation 601-1. Other colour spaces, such as the CIE 1976 L*a*b* space, may also achieve this objective. The chrominance components can then be compressed more than the luminance by using coarser sampling or quantization, with no degradation in quality.

In addition to the baseline JPEG format, beginning with PDF 1.3, the **DCTDecode** filter shall support the progressive JPEG extension. This extension does not add any entries to the **DCTDecode** parameter dictionary; the distinction between baseline and progressive JPEG shall be represented in the encoded data.

NOTE 5 There is no benefit to using progressive JPEG for stream data that is embedded in a PDF file. Decoding progressive JPEG is slower and consumes more memory than baseline JPEG. The purpose of this feature is to enable a stream to refer to an external file whose data happens to be already encoded in progressive JPEG.

7.4.9 JPXDecode Filter

The **JPXDecode** filter *(PDF 1.5)* decodes data that has been encoded using the JPEG2000 compression method, an ISO standard for the compression and packaging of image data.

NOTE 1 JPEG2000 defines a wavelet-based method for image compression that gives somewhat better size reduction than other methods such as regular JPEG or CCITT. Although the filter can reproduce samples that are losslessly compressed.

This filter shall only be applied to image XObjects, and not to inline images (see 8.9, "Images"). It is suitable both for images that have a single colour component and for those that have multiple colour components. The colour components in an image may have different numbers of bits per sample. Any value from 1 to 38 shall be allowed.

NOTE 2 From a single JPEG2000 data stream, multiple versions of an image may be decoded. These different versions form progressions along four degrees of freedom: sampling resolution, colour depth, band, and location. For example, with a resolution progression, a thumbnail version of the image may be decoded from the data, followed by a sequence of other versions of the image, each with approximately four times as many samples (twice the width times twice the height) as the previous one. The last version is the full-resolution image.

NOTE 3 Viewing and printing applications may gain performance benefits by using the resolution progression. If the full-resolution image is densely sampled, the application may be able to select and decode only the data making up a lower-resolution version, thereby spending less time decoding. Fewer bytes need be processed, a particular benefit when viewing files over the Web. The tiling structure of the image may also provide benefits if only certain areas of an image need to be displayed or printed.

NOTE 4 Information on these progressions is encoded in the data; no decode parameters are needed to describe them. The decoder deals with any progressions it encounters to deliver the correct image data. Progressions that are of no interest may simply have performance consequences.

The JPEG2000 specifications define two widely used formats, JP2 and JPX, for packaging the compressed image data. JP2 is a subset of JPX. These packagings contain all the information needed to properly interpret the image data, including the colour space, bits per component, and image dimensions. In other words, they are complete descriptions of images (as opposed to image data that require outside parameters for correct interpretation). The **JPXDecode** filter shall expect to read a full JPX file structure—either internal to the PDF file or as an external file.

NOTE 5 To promote interoperability, the specifications define a subset of JPX called JPX baseline (of which JP2 is also a subset). The complete details of the baseline set of JPX features are contained in ISO/IEC 15444-2, Information Technology—JPEG 2000 Image Coding System: Extensions (see the Bibliography). See also <http://www.jpeg.org/jpeg2000/>.

Data used in PDF image XObjects shall be limited to the JPX baseline set of features, except for enumerated colour space 19 (CIEJab). In addition, enumerated colour space 12 (CMYK), which is part of JPX but not JPX baseline, shall be supported in a PDF.

A JPX file describes a collection of *channels* that are present in the image data. A channel may have one of three types:

- An *ordinary* channel contains values that, when decoded, shall become samples for a specified colour component.
- An *opacity* channel provides samples that shall be interpreted as raw opacity information.
- A *premultiplied opacity* channel shall provide samples that have been multiplied into the colour samples of those channels with which it is associated.

Opacity and premultiplied opacity channels shall be associated with specific colour channels. There shall not be more than one opacity channel (of either type) associated with a given colour channel.

EXAMPLE It is possible for one opacity channel to apply to the red samples and another to apply to the green and blue colour channels of an RGB image.

NOTE 6 The method by which the opacity information is to be used is explicitly not specified, although one possible method shows a normal blending mode.

In addition to using opacity channels for describing transparency, JPX files also have the ability to specify chroma-key transparency. A single colour may be specified by giving an array of values, one value for each colour channel. Any image location that matches this colour shall be considered to be completely transparent.

Images in JPX files may have one of the following colour spaces:

- A predefined colour space, chosen from a list of *enumerated colour spaces*. (Two of these are actually families of spaces and parameters are included.)
- A restricted ICC profile. These are the only sorts of ICC profiles that are allowed in JP2 files.
- An input ICC profile of any sort defined by ICC-1.
- A *vendor-defined* colour space.

More than one colour space may be specified for an image, with each space being tagged with a precedence and an approximation value that indicates how well it represents the preferred colour space. In addition, the image's colour space may serve as the foundation for a palette of colours that are selected using samples coming from the image's data channels: the equivalent of an **Indexed** colour space in PDF.

There are other features in the JPX format beyond describing a simple image. These include provisions for describing layering and giving instructions on composition, specifying simple animation, and including generic XML metadata (along with JPEG2000-specific schemas for such data). Relevant metadata should be replicated in the image dictionary's **Metadata** stream in XMP format (see 14.3.2, "Metadata Streams").

When using the **JPXDecode** filter with image XObjects, the following changes to and constraints on some entries in the image dictionary shall apply (see 8.9.5, "Image Dictionaries" for details on these entries):

- **Width** and **Height** shall match the corresponding width and height values in the JPEG2000 data.
- **ColorSpace** shall be optional since JPEG2000 data contain colour space specifications. If present, it shall determine how the image samples are interpreted, and the colour space specifications in the JPEG2000 data shall be ignored. The number of colour channels in the JPEG2000 data shall match the number of components in the colour space; a conforming writer shall ensure that the samples are consistent with the colour space used.
- Any colour space other than **Pattern** may be specified. If an **Indexed** colour space is used, it shall be subject to the PDF limit of 256 colours. If the colour space does not match one of JPX's enumerated colour spaces (for example, if it has two colour components or more than four), it should be specified as a vendor colour space in the JPX data.
- If **ColorSpace** is not present in the image dictionary, the colour space information in the JPEG2000 data shall be used. A JPEG2000 image within a PDF shall have one of: the baseline JPX colorspaces; or enumerated colorspace 19 (CIEJab) or enumerated colorspace 12 (CMYK); or at least one ICC profile that is valid within PDF. Conforming PDF readers shall support the JPX baseline set of enumerated colour spaces; they shall also be responsible for dealing with the interaction between the colour spaces and the bit depth of samples.
- If multiple colour space specifications are given in the JPEG2000 data, a conforming reader should attempt to use the one with the highest precedence and best approximation value. If the colour space is given by an unsupported ICC profile, the next lower colour space, in terms of precedence and approximation value, shall be used. If no supported colour space is found, the colour space used shall be DeviceGray, DeviceRGB, or DeviceCMYK, depending on the whether the number of channels in the JPEG2000 data is 1,3, or 4.
- **SMaskInData** specifies whether soft-mask information packaged with the image samples shall be used (see 11.6.5.3, "Soft-Mask Images"); if it is, the **SMask** entry shall not be present. If **SMaskInData** is nonzero, there shall be only one opacity channel in the JPEG2000 data and it shall apply to all colour channels.
- **Decode** shall be ignored, except in the case where the image is treated as a mask; that is, when **ImageMask** is **true**. In this case, the JPEG2000 data shall provide a single colour channel with 1-bit samples.

7.4.10 Crypt Filter

The **Crypt** filter *(PDF 1.5)* allows the document-level security handler (see 7.6, "Encryption") to determine which algorithms should be used to decrypt the input data. The **Name** parameter in the decode parameters dictionary for this filter (see Table 14) shall specify which of the named crypt filters in the document (see 7.6.5, "Crypt Filters") shall be used. The Crypt filter shall be the first filter in the Filter array entry.

Table 14 – Optional parameters for Crypt filters

Key	Type	Value
Type	name	*(Optional)* If present, shall be **CryptFilterDecodeParms** for a **Crypt** filter decode parameter dictionary.
Name	name	*(Optional)* The name of the crypt filter that shall be used to decrypt this stream. The name shall correspond to an entry in the **CF** entry of the encryption dictionary (see Table 20) or one of the standard crypt filters (see Table 26). Default value: **Identity**.

In addition, the decode parameters dictionary may include entries that are private to the security handler. Security handlers may use information from both the crypt filter decode parameters dictionary and the crypt filter dictionaries (see Table 25) when decrypting data or providing a key to decrypt data.

NOTE When adding private data to the decode parameters dictionary, security handlers should name these entries in conformance with the PDF name registry (see Annex E).

If a stream specifies a crypt filter, then the security handler does not apply "Algorithm 1: Encryption of data using the RC4 or AES algorithms" in 7.6.2, "General Encryption Algorithm," to the key prior to decrypting the stream. Instead, the security handler shall decrypt the stream using the key as is. Sub-clause 7.4, "Filters," explains how a stream specifies filters.

7.5 File Structure

7.5.1 General

This sub-clause describes how objects are organized in a PDF file for efficient random access and incremental update. A basic conforming PDF file shall be constructed of following four elements (see Figure 2):

- A one-line *header* identifying the version of the PDF specification to which the file conforms
- A *body* containing the objects that make up the document contained in the file
- A *cross-reference table* containing information about the indirect objects in the file
- A *trailer* giving the location of the cross-reference table and of certain special objects within the body of the file

This initial structure may be modified by later updates, which append additional elements to the end of the file; see 7.5.6, "Incremental Updates," for details.

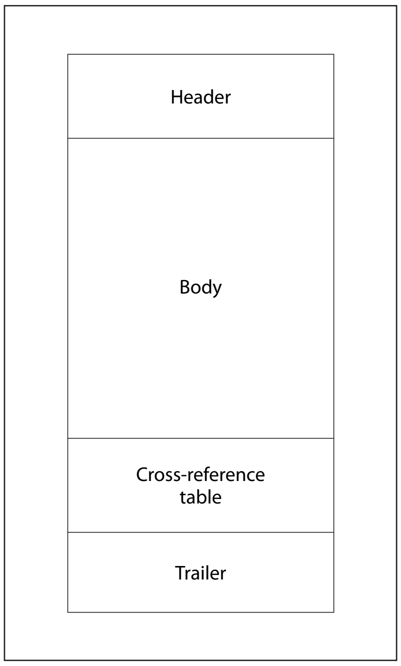


Figure 2 – Initial structure of a PDF file

As a matter of convention, the tokens in a PDF file are arranged into lines; see 7.2, "Lexical Conventions." Each line shall be terminated by an end-of-line (EOL) marker, which may be a CARRIAGE RETURN (0Dh), a LINE FEED (0Ah), or both. PDF files with binary data may have arbitrarily long lines.

NOTE To increase compatibility with compliant programs that process PDF files, lines that are not part of stream object data are limited to no more than 255 characters, with one exception. Beginning with PDF 1.3, the Contents string of a signature dictionary (see 12.8, "Digital Signatures") is not subject to the restriction on line length.

The rules described here are sufficient to produce a basic conforming PDF file. However, additional rules apply to organizing a PDF file to enable efficient incremental access to a document's components in a network environment. This form of organization, called *Linearized PDF*, is described in Annex F.

7.5.2 File Header

The first line of a PDF file shall be a *header* consisting of the 5 characters %PDF– followed by a version number of the form 1.N, where N is a digit between 0 and 7.

A conforming reader shall accept files with any of the following headers:

```
%PDF–1.0
%PDF–1.1
%PDF–1.2
%PDF–1.3
%PDF–1.4
%PDF–1.5
%PDF–1.6
%PDF–1.7
```

Beginning with PDF 1.4, the **Version** entry in the document's catalog dictionary (located via the **Root** entry in the file's trailer, as described in 7.5.5, "File Trailer"), if present, shall be used instead of the version specified in the Header.

NOTE This allows a conforming writer to update the version using an incremental update (see 7.5.6, "Incremental Updates").

Under some conditions, a conforming reader may be able to process PDF files conforming to a later version than it was designed to accept. New PDF features are often introduced in such a way that they can safely be ignored by a conforming reader that does not understand them (see I.2, "PDF Version Numbers").

This part of ISO 32000 defines the Extensions entry in the document’s catalog dictionary. If present, it shall identify any developer-defined extensions that are contained in this PDF file. See 7.12, “Extensions Dictionary”.

If a PDF file contains binary data, as most do (see 7.2, "Lexical Conventions"), the header line shall be immediately followed by a comment line containing at least four binary characters—that is, characters whose codes are 128 or greater. This ensures proper behaviour of file transfer applications that inspect data near the beginning of a file to determine whether to treat the file’s contents as text or as binary.

7.5.3 File Body

The *body* of a PDF file shall consist of a sequence of indirect objects representing the contents of a document. The objects, which are of the basic types described in 7.3, "Objects," represent components of the document such as fonts, pages, and sampled images. Beginning with PDF 1.5, the body can also contain object streams, each of which contains a sequence of indirect objects; see 7.5.7, "Object Streams."

7.5.4 Cross-Reference Table

The *cross-reference table* contains information that permits random access to indirect objects within the file so that the entire file need not be read to locate any particular object. The table shall contain a one-line entry for each indirect object, specifying the byte offset of that object within the body of the file. (Beginning with PDF 1.5, some or all of the cross-reference information may alternatively be contained in cross-reference streams; see 7.5.8, "Cross-Reference Streams.")

NOTE 1 The cross-reference table is the only part of a PDF file with a fixed format, which permits entries in the table to be accessed randomly.

The table comprises one or more *cross-reference sections*. Initially, the entire table consists of a single section (or two sections if the file is linearized; see Annex F). One additional section shall be added each time the file is incrementally updated (see 7.5.6, "Incremental Updates").

Each cross-reference section shall begin with a line containing the keyword **xref**. Following this line shall be one or more *cross-reference subsections*, which may appear in any order. For a file that has never been incrementally updated, the cross-reference section shall contain only one subsection, whose object numbering begins at 0.

NOTE 2 The subsection structure is useful for incremental updates, since it allows a new cross-reference section to be added to the PDF file, containing entries only for objects that have been added or deleted.

Each cross-reference subsection shall contain entries for a contiguous range of object numbers. The subsection shall begin with a line containing two numbers separated by a SPACE (20h), denoting the object number of the first object in this subsection and the number of entries in the subsection.

EXAMPLE 1 The following line introduces a subsection containing five objects numbered consecutively from 28 to 32.

```
28 5
```

A given object number shall not have an entry in more than one subsection within a single section.

Following this line are the cross-reference entries themselves, one per line. Each entry shall be exactly 20 bytes long, including the end-of-line marker. There are two kinds of cross-reference entries: one for objects that are in use and another for objects that have been deleted and therefore are free. Both types of entries have similar basic formats, distinguished by the keyword **n** (for an in-use entry) or **f** (for a free entry). The format of an in-use entry shall be:

nnnnnnnnnn ggggg **n** *eol*

where:

nnnnnnnnnn shall be a 10-digit byte offset in the decoded stream

ggggg shall be a 5-digit generation number

n shall be a keyword identifying this as an in-use entry

eol shall be a 2-character end-of-line sequence

The byte offset in the decoded stream shall be a 10-digit number, padded with leading zeros if necessary, giving the number of bytes from the beginning of the file to the beginning of the object. It shall be separated from the generation number by a single SPACE. The generation number shall be a 5-digit number, also padded with leading zeros if necessary. Following the generation number shall be a single SPACE, the keyword **n**, and a 2-character end-of-line sequence consisting of one of the following: SP CR, SP LF, or CR LF. Thus, the overall length of the entry shall always be exactly 20 bytes.

The cross-reference entry for a free object has essentially the same format, except that the keyword shall be **f** instead of **n** and the interpretation of the first item is different:

nnnnnnnnnn ggggg **f** *eol*

where:

nnnnnnnnnn shall be the 10-digit object number of the next free object

ggggg shall be a 5-digit generation number

f shall be a keyword identifying this as a free entry

eol shall be a 2-character end-of-line sequence

There are two ways an entry may be a member of the free entries list. Using the basic mechanism the free entries in the cross-reference table may form a linked list, with each free entry containing the object number of the next. The first entry in the table (object number 0) shall always be free and shall have a generation number of 65,535; it is shall be the head of the linked list of free objects. The last free entry (the tail of the linked list) links back to object number 0. Using the second mechanism, the table may contain other free entries that link back to object number 0 and have a generation number of 65,535, even though these entries are not in the linked list itself.

Except for object number 0, all objects in the cross-reference table shall initially have generation numbers of 0. When an indirect object is deleted, its cross-reference entry shall be marked free and it shall be added to the linked list of free entries. The entry's generation number shall be incremented by 1 to indicate the generation number to be used the next time an object with that object number is created. Thus, each time the entry is reused, it is given a new generation number. The maximum generation number is 65,535; when a cross-reference entry reaches this value, it shall never be reused.

The cross-reference table (comprising the original cross-reference section and all update sections) shall contain one entry for each object number from 0 to the maximum object number defined in the file, even if one or more of the object numbers in this range do not actually occur in the file.

EXAMPLE 2 The following shows a cross-reference section consisting of a single subsection with six entries: four that are in use (objects number 1, 2, 4, and 5) and two that are free (objects number 0 and 3). Object number 3 has been deleted, and the next object created with that object number is given a generation number of 7.

```
xref
0 6
0000000003 65535 f
0000000017 00000 n
0000000081 00000 n
0000000000 00007 f
0000000331 00000 n
0000000409 00000 n
```

EXAMPLE 3 The following shows a cross-reference section with four subsections, containing a total of five entries. The first subsection contains one entry, for object number 0, which is free. The second subsection contains one entry, for object number 3, which is in use. The third subsection contains two entries, for objects number 23 and 24, both of which are in use. Object number 23 has been reused, as can be seen from the fact that it has a generation number of 2. The fourth subsection contains one entry, for object number 30, which is in use.

```
xref
0 1
0000000000 65535 f
3 1
0000025325 00000 n
23 2
0000025518 00002 n
0000025635 00000 n
30 1
0000025777 00000 n
```

See H.7, "Updating Example", for a more extensive example of the structure of a PDF file that has been updated several times.

7.5.5 File Trailer

The *trailer* of a PDF file enables a conforming reader to quickly find the cross-reference table and certain special objects. Conforming readers should read a PDF file from its end. The last line of the file shall contain only the end-of-file marker, **%%EOF**. The two preceding lines shall contain, one per line and in order, the keyword **startxref** and the byte offset in the decoded stream from the beginning of the file to the beginning of the **xref** keyword in the last cross-reference section. The **startxref** line shall be preceded by the *trailer dictionary*, consisting of the keyword **trailer** followed by a series of key-value pairs enclosed in double angle brackets (<<…>>) (using LESS-THAN SIGNs (3Ch) and GREATER-THAN SIGNs (3Eh)). Thus, the trailer has the following overall structure:

trailer
<< *key*$_1$ *value*$_1$
key$_2$ *value*$_2$
…
key$_n$ *value*$_n$
>>
startxref
Byte_offset_of_last_cross-reference_section
%%EOF

Table 15 lists the contents of the trailer dictionary.

Table 15 – Entries in the file trailer dictionary

Key	Type	Value
Size	integer	*(Required; shall not be an indirect reference)* The total number of entries in the file's cross-reference table, as defined by the combination of the original section and all update sections. Equivalently, this value shall be 1 greater than the highest object number defined in the file. Any object in a cross-reference section whose number is greater than this value shall be ignored and defined to be missing by a conforming reader.
Prev	integer	*(Present only if the file has more than one cross-reference section; shall be an indirect reference)* The byte offset in the decoded stream from the beginning of the file to the beginning of the previous cross-reference section.
Root	dictionary	*(Required; shall be an indirect reference)* The catalog dictionary for the PDF document contained in the file (see 7.7.2, "Document Catalog").
Encrypt	dictionary	*(Required if document is encrypted; PDF 1.1)* The document's encryption dictionary (see 7.6, "Encryption").
Info	dictionary	*(Optional; shall be an indirect reference)* The document's information dictionary (see 14.3.3, "Document Information Dictionary").
ID	array	*(Required if an* ***Encrypt*** *entry is present; optional otherwise; PDF 1.1)* An array of two byte-strings constituting a file identifier (see 14.4, "File Identifiers") for the file. If there is an ***Encrypt*** entry this array and the two byte-strings shall be direct objects and shall be unencrypted. NOTE 1 Because the **ID** entries are not encrypted it is possible to check the **ID** key to assure that the correct file is being accessed without decrypting the file. The restrictions that the string be a direct object and not be encrypted assure that this is possible. NOTE 2 Although this entry is optional, its absence might prevent the file from functioning in some workflows that depend on files being uniquely identified. NOTE 3 The values of the **ID** strings are used as input to the encryption algorithm. If these strings were indirect, or if the **ID** array were indirect, these strings would be encrypted when written. This would result in a circular condition for a reader: the **ID** strings must be decrypted in order to use them to decrypt strings, including the **ID** strings themselves. The preceding restriction prevents this circular condition.

NOTE Table 19 defines an additional entry, **XRefStm**, that appears only in the trailer of hybrid-reference files, described in 7.5.8.4, "Compatibility with Applications That Do Not Support Compressed Reference Streams."

EXAMPLE This example shows a trailer for a file that has never been updated (as indicated by the absence of a Prev entry in the trailer dictionary).

```
trailer
   << /Size 22
      /Root 2 0 R
      /Info 1 0 R
      /ID [   <81b14aafa313db63dbd6f981e49f94f4>
              <81b14aafa313db63dbd6f981e49f94f4>
         ]
   >>
startxref
18799
%%EOF
```

7.5.6 Incremental Updates

The contents of a PDF file can be updated incrementally without rewriting the entire file. When updating a PDF file incrementally, changes shall be appended to the end of the file, leaving its original contents intact.

NOTE 1 The main advantage to updating a file in this way is that small changes to a large document can be saved quickly. There are additional advantages:

In certain contexts, such as when editing a document across an HTTP connection or using OLE embedding (a Windows-specific technology), a conforming writer cannot overwrite the contents of the original file. Incremental updates may be used to save changes to documents in these contexts.

NOTE 2 The resulting file has the structure shown in Figure 3. A complete example of an updated file is shown in H.7, "Updating Example".

A cross-reference section for an incremental update shall contain entries only for objects that have been changed, replaced, or deleted. Deleted objects shall be left unchanged in the file, but shall be marked as deleted by means of their cross-reference entries. The added trailer shall contain all the entries except the Prev entry (if present) from the previous trailer, whether modified or not. In addition, the added trailer dictionary shall contain a **Prev** entry giving the location of the previous cross-reference section (see Table 15). Each trailer shall be terminated by its own end-of-file (%%EOF) marker.

NOTE 3 As shown in Figure 3, a file that has been updated several times contains several trailers. Because updates are appended to PDF files, a file may have several copies of an object with the same object identifier (object number and generation number).

EXAMPLE Several copies of an object can occur if a text annotation (see 12.5, "Annotations") is changed several times and the file is saved between changes. Because the text annotation object is not deleted, it retains the same object number and generation number as before. The updated copy of the object is included in the new update section added to the file.

The update's cross-reference section shall include a byte offset to this new copy of the object, overriding the old byte offset contained in the original cross-reference section. When a conforming reader reads the file, it shall build its cross-reference information in such a way that the most recent copy of each object shall be the one accessed from the file.

In versions of PDF 1.4 or later a conforming writer may use the **Version** entry in the document's catalog dictionary (see 7.7.2, "Document Catalog") to override the version specified in the header. A conforming writer may also need to update the Extensions dictionary, see 7.12, “Extensions Dictionary”, if the update either deleted or added developer-defined extensions.

NOTE 4 The version entry enables the version to be altered when performing an incremental update.

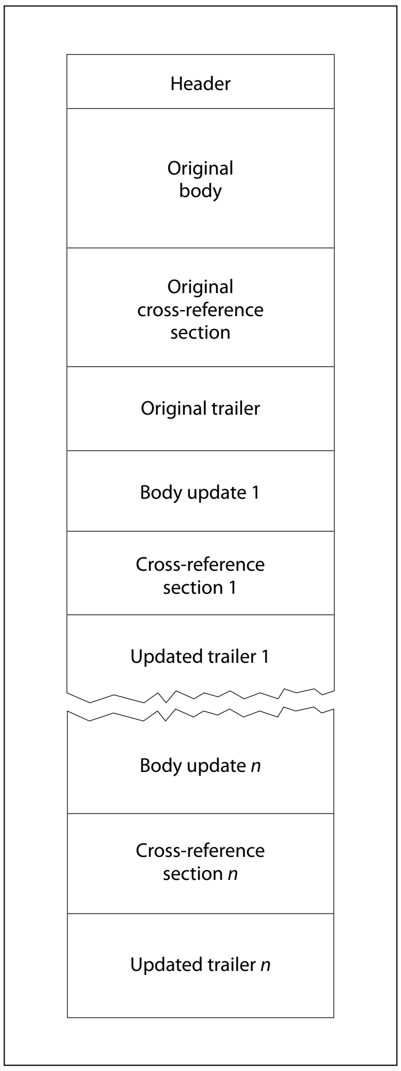


Figure 3 – Structure of an updated PDF file

7.5.7 Object Streams

An *object stream*, is a stream object in which a sequence of indirect objects may be stored, as an alternative to their being stored at the outermost file level.

NOTE 1 Object streams are first introduced in PDF 1.5. The purpose of object streams is to allow indirect objects other than streams to be stored more compactly by using the facilities provided by stream compression filters.

NOTE 2 The term "compressed object" is used regardless of whether the stream is actually encoded with a compression filter.

The following objects shall not be stored in an object stream:

- Stream objects
- Objects with a generation number other than zero
- A document's encryption dictionary (see 7.6, "Encryption")
- An object representing the value of the **Length** entry in an object stream dictionary

- In linearized files (see Annex F), the document catalog, the linearization dictionary, and page objects shall not appear in an object stream.

NOTE 3 Indirect references to objects inside object streams use the normal syntax: for example, 14 0 R. Access to these objects requires a different way of storing cross-reference information; see 7.5.8, "Cross-Reference Streams." Use of compressed objects requires a PDF 1.5 conforming reader. However, compressed objects can be stored in a manner that a PDF 1.4 conforming reader can ignore.

In addition to the regular keys for streams shown in Table 5, the stream dictionary describing an object stream contains the following entries:

Table 16 – Additional entries specific to an object stream dictionary

key	type	description
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **ObjStm** for an object stream.
N	integer	*(Required)* The number of indirect objects stored in the stream.
First	integer	*(Required)* The byte offset in the decoded stream of the first compressed object.
Extends	stream	*(Optional)* A reference to another object stream, of which the current object stream shall be considered an extension. Both streams are considered part of a *collection* of object streams (see below). A given collection consists of a set of streams whose **Extends** links form a directed acyclic graph.

A conforming writer determines which objects, if any, to store in object streams.

EXAMPLE 1 It can be useful to store objects having common characteristics together, such as “fonts on page 1,” or “Comments for draft #3.” These objects are known as a *collection*.

NOTE 4 To avoid a degradation of performance, such as would occur when downloading and decompressing a large object stream to access a single compressed object, the number of objects in an individual object stream should be limited. This may require a group of object streams to be linked as a collection, which can be done by means of the **Extends** entry in the object stream dictionary.

NOTE 5 **Extends** may also be used when a collection is being updated to include new objects. Rather than modifying the original object stream, which could entail duplicating much of the stream data, the new objects can be stored in a separate object stream. This is particularly important when adding an update section to a document.

The stream data in an object stream shall contain the following items:

- **N** pairs of integers separated by white space, where the first integer in each pair shall represent the object number of a compressed object and the second integer shall represent the byte offset in the decoded stream of that object, relative to the first object stored in the object stream, the value of the stream's first entry. The offsets shall be in increasing order.

NOTE 6 There is no restriction on the order of objects in the object stream; in particular, the objects need not be stored in object-number order.

- The value of the **First** entry in the stream dictionary shall be the byte offset in the decoded stream of the first object.

- The **N** objects are stored consecutively. Only the object values are stored in the stream; the **obj** and **endobj** keywords shall not be used.

NOTE 7 A compressed dictionary or array may contain indirect references.

An object in an object stream shall not consist solely of an object reference.

EXAMPLE 2 3 0 R

In an encrypted file (i.e., entire object stream is encrypted), strings occurring anywhere in an object stream shall not be separately encrypted.

A conforming writer shall store the first object immediately after the last byte offset. A conforming reader shall rely on the **First** entry in the stream dictionary to locate the first object.

An object stream itself, like any stream, shall be an indirect object, and therefore, there shall be an entry for it in a cross-reference table or cross-reference stream (see 7.5.8, "Cross-Reference Streams"), although there might not be any references to it (of the form 243 0 R).

The generation number of an object stream and of any compressed object shall be zero. If either an object stream or a compressed object is deleted and the object number is freed, that object number shall be reused only for an ordinary (uncompressed) object other than an object stream. When new object streams and compressed objects are created, they shall always be assigned new object numbers, not old ones taken from the free list.

EXAMPLE 3 The following shows three objects (two fonts and a font descriptor) as they would be represented in a PDF 1.4 or earlier file, along with a cross-reference table.

```
11 0 obj
    << /Type /Font
        /Subtype /TrueType
        ...other entries...
        /FontDescriptor 12 0 R
    >>
endobj

12 0 obj
    << /Type /FontDescriptor
        /Ascent 891
        ...other entries...
        /FontFile2 22 0 R
    >>
endobj

13 0 obj
    << /Type /Font
        /Subtype /Type0
        ...other entries...
        /ToUnicode 10 0 R
    >>
endobj

...

xref
0 32
0000000000 65535 f
... cross-reference entries for objects 1 through 10 ...
0000001434 00000 n
0000001735 00000 n
0000002155 00000 n
... cross-reference entries for objects 14 and on ...
trailer
    << /Size 32
        /Root ...
    >>
```

NOTE 8 For readability, the object stream has been shown unencoded. In a real PDF 1.5 file, Flate encoding would typically be used to gain the benefits of compression.

EXAMPLE 4 The following shows the same objects from the previous example stored in an object stream in a PDF 1.5 file, along with a cross-reference stream.

The cross-reference stream (see 7.5.8, "Cross-Reference Streams") contains entries for the fonts (objects 11 and 13) and the descriptor (object 12), which are compressed objects in an object stream. The first field of these entries is the entry type (2), the second field is the number of the object stream (15), and the third field is the position within the sequence of objects in the object stream (0, 1, and 2). The cross-reference stream also contains a type 1 entry for the object stream itself.

```
15 0 obj                              % The object stream
   << /Type /ObjStm
      /Length 1856
      /N 3                            % The number of objects in the stream
      /First 24                       % The byte offset in the decoded stream of the first object
% The object numbers and offsets of the objects, relative to the first are shown on the first line of
%  the stream (i.e., 11 0 12 547 13 665).
   >>
stream
   11 0 12 547 13 665
   << /Type /Font
      /Subtype /TrueType
      ...other keys...
      /FontDescriptor 12 0 R
   >>

   << /Type /FontDescriptor
      /Ascent 891
      ...other keys...
      /FontFile2 22 0 R
   >>
   << /Type /Font
      /Subtype /Type0
      ...other keys...
      /ToUnicode 10 0 R
   >>
...
endstream
endobj

99 0 obj                       % The cross-reference stream
   << /Type /XRef
      /Index [0 32]                   % This section has one subsection with 32 objects
      /W [1 2 2]                      % Each entry has 3 fields: 1, 2 and 2 bytes in width,
                                      % respectively
      /Filter /ASCIIHexDecode         % For readability in this example
      /Size 32
      ...
   >>
stream
   00 0000 FFFF
   ... cross-references for objects 1 through 10 ...
   02 000F 0000
   02 000F 0001
   02 000F 0002
   ... cross-reference for object 14 ...
   01 BA5E 0000
   ...
endstream
endobj

startxref
   54321
%%EOF
```

NOTE 9 The number 54321 in Example 4 is the offset for object 99 0.

7.5.8 Cross-Reference Streams

7.5.8.1 General

Beginning with PDF 1.5, cross-reference information may be stored in a *cross-reference stream* instead of in a cross-reference table. Cross-reference streams provide the following advantages:

- A more compact representation of cross-reference information
- The ability to access compressed objects that are stored in object streams (see 7.5.7, "Object Streams") and to allow new cross-reference entry types to be added in the future

Cross-reference streams are stream objects (see 7.3.8, "Stream Objects"), and contain a dictionary and a data stream. Each cross-reference stream contains the information equivalent to the cross-reference table (see 7.5.4, "Cross-Reference Table") and trailer (see 7.5.5, "File Trailer") for one cross-reference section.

EXAMPLE In this example, the trailer dictionary entries are stored in the stream dictionary, and the cross-reference table entries are stored as the stream data.

```
... objects ...

12 0 obj                 % Cross-reference stream
   << /Type /XRef        % Cross-reference stream dictionary
      /Size ...
      /Root ...
   >>
stream
   ... Stream data containing cross-reference information ...
endstream
endobj

... more objects ...

startxref
byte_offset_of_cross-reference_stream (points to object 12)
%%EOF
```

The value following the **startxref** keyword shall be the offset of the cross-reference stream rather than the **xref** keyword. For files that use cross-reference streams entirely (that is, files that are not hybrid-reference files; see 7.5.8.4, "Compatibility with Applications That Do Not Support Compressed Reference Streams"), the keywords **xref** and **trailer** shall no longer be used. Therefore, with the exception of the **startxref** *address* **%%EOF** segment and comments, a file may be entirely a sequence of objects.

In linearized files (see F.3, "Linearized PDF Document Structure"), the document catalog, the linearization dictionary, and page objects shall not appear in an object stream.

7.5.8.2 Cross-Reference Stream Dictionary

Cross-reference streams may contain the entries shown in Table 17 in addition to the entries common to all streams (Table 5) and trailer dictionaries (Table 15). Since some of the information in the cross-reference stream is needed by the conforming reader to construct the index that allows indirect references to be resolved, the entries in cross-reference streams shall be subject to the following restrictions:

- The values of all entries shown in Table 17 shall be direct objects; indirect references shall not be permitted. For arrays (the **Index** and **W** entries), all of their elements shall be direct objects as well. If the stream is encoded, the **Filter** and **DecodeParms** entries in Table 5 shall also be direct objects.
- Other cross-reference stream entries not listed in Table 17 may be indirect; in fact, some (such as **Root** in Table 15) shall be indirect.

- The cross-reference stream shall not be encrypted and strings appearing in the cross-reference stream dictionary shall not be encrypted. It shall not have a **Filter** entry that specifies a **Crypt** filter (see 7.4.10, "Crypt Filter").

Table 17 – Additional entries specific to a cross-reference stream dictionary

key	type	description
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **XRef** for a cross-reference stream.
Size	integer	*(Required)* The number one greater than the highest object number used in this section or in any section for which this shall be an update. It shall be equivalent to the **Size** entry in a trailer dictionary.
Index	array	*(Optional)* An array containing a pair of integers for each subsection in this section. The first integer shall be the first object number in the subsection; the second integer shall be the number of entries in the subsection The array shall be sorted in ascending order by object number. Subsections cannot overlap; an object number may have at most one entry in a section. Default value: [0 **Size**].
Prev	integer	*(Present only if the file has more than one cross-reference stream; not meaningful in hybrid-reference files; see 7.5.8.4, "Compatibility with Applications That Do Not Support Compressed Reference Streams")* The byte offset in the decoded stream from the beginning of the file to the beginning of the previous cross-reference stream. This entry has the same function as the **Prev** entry in the trailer dictionary (Table 15).
W	array	*(Required)* An array of integers representing the size of the fields in a single cross-reference entry. Table 18 describes the types of entries and their fields. For PDF 1.5, **W** always contains three integers; the value of each integer shall be the number of bytes (in the decoded stream) of the corresponding field. EXAMPLE [1 2 1] means that the fields are one byte, two bytes, and one byte, respectively. A value of zero for an element in the **W** array indicates that the corresponding field shall not be present in the stream, and the default value shall be used, if there is one. If the first element is zero, the type field shall not be present, and shall default to type 1. The sum of the items shall be the total length of each entry; it can be used with the **Index** array to determine the starting position of each subsection. Different cross-reference streams in a PDF file may use different values for *W*.

7.5.8.3 Cross-Reference Stream Data

Each entry in a cross-reference stream shall have one or more fields, the first of which designates the entry's type (see Table 18). In PDF 1.5 through PDF 1.7, only types 0, 1, and 2 are allowed. Any other value shall be interpreted as a reference to the null object, thus permitting new entry types to be defined in the future.

The fields are written in increasing order of field number; the length of each field shall be determined by the corresponding value in the **W** entry (see Table 17). Fields requiring more than one byte are stored with the high-order byte first.

Table 18 – Entries in a cross-reference stream

Type	Field	Description
0	1	The type of this entry, which shall be 0. Type 0 entries define the linked list of free objects (corresponding to **f** entries in a cross-reference table).
	2	The object number of the next free object.
	3	The generation number to use if this object number is used again.
1	1	The type of this entry, which shall be 1. Type 1 entries define objects that are in use but are not compressed (corresponding to **n** entries in a cross-reference table).
	2	The byte offset of the object, starting from the beginning of the file.
	3	The generation number of the object. Default value: 0.
2	1	The type of this entry, which shall be 2. Type 2 entries define compressed objects.
	2	The object number of the object stream in which this object is stored. (The generation number of the object stream shall be implicitly 0.)
	3	The index of this object within the object stream.

Like any stream, a cross-reference stream shall be an indirect object. Therefore, an entry for it shall exist in either a cross-reference stream (usually itself) or in a cross-reference table (in hybrid-reference files; see 7.5.8.4, "Compatibility with Applications That Do Not Support Compressed Reference Streams").

7.5.8.4 Compatibility with Applications That Do Not Support Compressed Reference Streams

Readers designed only to support versions of PDF before PDF 1.5, and hence do not support cross-reference streams, cannot access objects that are referenced by cross-reference streams. If a file uses cross-reference streams exclusively, it cannot be opened by such readers.

However, it is possible to construct a file called a *hybrid-reference* file that is readable by readers designed only to support versions of PDF before PDF 1.5. Such a file contains objects referenced by standard cross-reference tables in addition to objects in object streams that are referenced by cross-reference streams.

In these files, the trailer dictionary may contain, in addition to the entry for trailers shown in Table 15, an entry, as shown in Table 19. This entry may be ignored by readers designed only to support versions of PDF before PDF 1.5, which therefore have no access to entries in the cross-reference stream the entry refers to.

Table 19 – Additional entries in a hybrid-reference file’s trailer dictionary

Key	Type	Value
XRefStm	integer	*(Optional)* The byte offset in the decoded stream from the beginning of the file of a cross-reference stream.

The **Size** entry of the trailer shall be large enough to include all objects, including those defined in the cross-reference stream referenced by the **XRefStm** entry. However, to allow random access, a main cross-reference section shall contain entries for all objects numbered 0 through **Size** - 1 (see 7.5.4, "Cross-Reference Table"). Therefore, the **XRefStm** entry shall not be used in the trailer dictionary of the main cross-reference section but only in an update cross-reference section.

When a conforming reader opens a hybrid-reference file, objects with entries in cross-reference streams are not hidden. When the conforming reader searches for an object, if an entry is not found in any given standard cross-reference section, the search shall proceed to a cross-reference stream specified by the **XRefStm** entry before looking in the previous cross-reference section (the **Prev** entry in the trailer).

Hidden objects, therefore, have two cross-reference entries. One is in the cross-reference stream. The other is a free entry in some previous section, typically the section referenced by the **Prev** entry. A conforming reader shall look in the cross-reference stream first, shall find the object there, and shall ignore the free entry in the previous section. A reader designed only to support versions of PDF before PDF 1.5 ignores the cross-reference stream and looks in the previous section, where it finds the free entry. The free entry shall have a next-generation number of 65535 so that the object number shall not be reused.

There are limitations on which objects in a hybrid-reference file can be hidden without making the file appear invalid to readers designed only to support versions of PDF before PDF 1.5. In particular, the root of the PDF file and the document catalog (see 7.7.2, "Document Catalog") shall not be hidden, nor any object that is *visible from the root*. Such objects can be determined by starting from the root and working recursively:

- In any dictionary that is visible, direct objects shall be visible. The value of any required key-value pair shall be visible.
- In any array that is visible, every element shall be visible.
- Resource dictionaries in content streams shall be visible. Although a resource dictionary is not required, strictly speaking, the content stream to which it is attached is assumed to contain references to the resources.

In general, the objects that may be hidden are optional objects specified by indirect references. A conforming reader can resolve those references by processing the cross-reference streams. In a reader designed only to support versions of PDF before PDF 1.5, the objects appear to be free, and the references shall be treated as references to the null object.

EXAMPLE 1 The **Outlines** entry in the catalog dictionary is optional. Therefore, its value may be an indirect reference to a hidden object. A reader designed only to support versions of PDF before PDF 1.5 treats it as a reference to the null object, which is equivalent to having omitted the entry entirely; a conforming reader recognizes it.

If the value of the **Outlines** entry is an indirect reference to a visible object, the entire outline tree shall be visible because nodes in the outline tree contain required pointers to other nodes.

Items that shall be visible include the entire page tree, fonts, font descriptors, and width tables. Objects that may be hidden in a hybrid-reference file include the structure tree, the outline tree, article threads, annotations, destinations, Web Capture information, and page labels,.

EXAMPLE 2 In this example, an **ASCIIHexDecode** filter is specified to make the format and contents of the cross-reference stream readable.

This example shows a hybrid-reference file containing a main cross-reference section and an update cross-reference section with an **XRefStm** entry that points to a cross-reference stream (object 11), which in turn has references to an object stream (object 2).

In this example, the catalog (object 1) contains an indirect reference (3 0 R) to the root of the structure tree. The search for the object starts at the update cross-reference table, which has no objects in it. The search proceeds depending on the version of the conforming reader.

One choice for a reader designed only to support versions of PDF before PDF 1.5 is to continue the search by following the **Prev** pointer to the main cross-reference table. That table defines object 3 as a free object, which is treated as the **null** object. Therefore, the entry is considered missing, and the document has no structure tree.

Another choice for a conforming reader, is to continue the search by following the **XRefStm** pointer to the cross-reference stream (object 11). It defines object 3 as a compressed object, stored at index 0 in the object stream (2 0 obj). Therefore, the document has a structure tree.

```
1 0 obj                                % The document root, at offset 23.
    << /Type /Catalog
        /StructTreeRoot 3 0 R
        …
    >>
endobj

12 0 obj
…
endobj
…
99 0 obj
…
endobj

% The main xref section, at offset 2664 is next with entries for objects 0-99.
% Objects 2 through 11 are marked free and objects 12, 13 and 99 are marked in use.
xref
0 100
0000000002 65535 f
0000000023 00000 n
0000000003 65535 f
0000000004 65535 f
0000000005 65535 f
0000000006 65535 f
0000000007 65535 f
0000000008 65535 f
0000000009 65535 f
0000000010 65535 f
0000000011 65535 f
0000000000 65535 f
0000000045 00000 n
0000000179 00000 n
… cross-reference entries for objects 14 through 98 ...
0000002201 00000 n
trailer
    << /Size 100
        /Root 1 0 R
        /ID …
    >>
% The main xref section starts at offset 2664.
startxref
    2664
%%EOF

2 0 obj                                % The object stream, at offset 3722
    << /Length ...
        /N 8                           % This stream contains 8 objects.
        /First 47                      % The stream-offset of the first object
    >>
stream
    3 0 4 50 5 72 … the numbers and stream-offsets of the remaining 5 objects followed by dictionary
objects 3-5 ...
    << /Type /StructTreeRoot
        /K 4 0 R
        /RoleMap 5 0 R
```

```
            /ClassMap 6 0 R
            /ParentTree 7 0 R
            /ParentTreeNextKey 8
        >>
        << /S /Workbook
            /P 8 0 R
            /K 9 0 R
        >>
        << /Workbook /Div
            /Worksheet /Sect
            /TextBox /Figure
            /Shape /Figure
        >>
        … definitions for objects 6 through 10 ...
    endstream
    endobj

    11 0 obj                          % The cross-reference stream, at offset 4899
        << /Type /XRef
            /Index [2 10]             % This stream contains entries for objects 2 through 11
            /Size 100
            /W [1 2 1]                % The byte-widths of each field
            /Filter /ASCIIHexDecode   % For readability only
            …
        >>
    stream
        01 0E8A 0
        02 0002 00
        02 0002 01
        02 0002 02
        02 0002 03
        02 0002 04
        02 0002 05
        02 0002 06
        02 0002 07
        01 1323 0
    endstream
    endobj
    % The entries above are for: object 2 (0x0E8A = 3722), object 3 (in object stream 2, index 0),
    % object 4 (in object stream 2, index 1) ... object 10 (in object stream 2, index 7),
    % object 11 (0x1323 = 4899).

    % The update xref section starting at offset 5640.  There are no entries in this section.
    xref
    0 0
    trailer
        << /Size 100
            /Prev 2664                % Offset of previous xref section
            /XRefStm 4899
            /Root 1 0 R
            /ID …
        >>
    startxref
    5640
    %%EOF
```

The previous example illustrates several other points:

- The object stream is unencoded and the cross-reference stream uses an ASCII hexadecimal encoding for clarity. In practice, both streams should be Flate-encoded. PDF comments shall not be included in a cross-reference table or in cross-reference streams.

- The hidden objects, 2 through 11, are numbered consecutively. In practice, hidden objects and other free items in a cross-reference table need not be linked in ascending order until the end.

- The update cross-reference table need not contain any entries. A conforming writer that uses the hybrid-reference format creates the main cross-reference table, the update cross-reference table, and the cross-reference stream at the same time. Objects 12 and 13, for example, are not compressed. They might have entries in the update table. Since objects 2 and 11, the object stream and the cross-reference stream, are not compressed, they might also be defined in the update table. Since they are part of the hidden section, however, it makes sense to define them in the cross-reference stream.

- The update cross-reference section shall appear at the end of the file, but otherwise, there are no ordering restrictions on any of the objects or on the main cross-reference section. However, a file that uses both the hybrid-reference format and the linearized format has ordering requirements (see Annex F).

7.6 Encryption

7.6.1 General

A PDF document can be *encrypted (PDF 1.1)* to protect its contents from unauthorized access. Encryption applies to all strings and streams in the document's PDF file, with the following exceptions:

- The values for the ID entry in the trailer

- Any strings in an Encrypt dictionary

- Any strings that are inside streams such as content streams and compressed object streams, which themselves are encrypted

Encryption is not applied to other object types such as integers and boolean values, which are used primarily to convey information about the document's structure rather than its contents. Leaving these values unencrypted allows random access to the objects within a document, whereas encrypting the strings and streams protects the document's contents.

When a PDF stream object (see 7.3.8, "Stream Objects") refers to an external file, the stream's contents shall not be encrypted, since they are not part of the PDF file itself. However, if the contents of the stream are embedded within the PDF file (see 7.11.4, "Embedded File Streams"), they shall be encrypted like any other stream in the file. Beginning with PDF 1.5, embedded files can be encrypted in an otherwise unencrypted document (see 7.6.5, "Crypt Filters").

Encryption-related information shall be stored in a document's *encryption dictionary*, which shall be the value of the **Encrypt** entry in the document's trailer dictionary (see Table 15). The absence of this entry from the trailer dictionary means that a conforming reader shall consider the document to be not encrypted. The entries shown in Table 20 are common to all encryption dictionaries.

The encryption dictionary's **Filter** entry identifies the file's *security handler*, a software module that implements various aspects of the encryption process and controls access to the contents of the encrypted document. PDF specifies a standard password-based security handler that all conforming readers shall support, but conforming readers can optionally provide additional security handlers of their own.

The **SubFilter** entry specifies the syntax of the encryption dictionary contents. It allows interoperability between handlers; that is, a document can be decrypted by a handler other than the preferred one (the **Filter** entry) if they both support the format specified by **SubFilter**.

The **V** entry, in specifying which algorithm to use, determines the length of the encryption key, on which the encryption (and decryption) of data in a PDF file shall be based. For **V** values 2 and 3, the **Length** entry specifies the exact length of the encryption key. In PDF 1.5, a value of 4 for **V** permits the security handler to use its own encryption and decryption algorithms and to specify *crypt filters* to use on specific streams (see 7.6.5, "Crypt Filters").

The remaining contents of the encryption dictionary shall be determined by the security handler and may vary from one handler to another. Entries for the standard security handler are described in 7.6.3, "Standard

Security Handler." Entries for public-key security handlers are described in 7.6.4, "Public-Key Security Handlers."

Table 20 – Entries common to all encryption dictionaries

Key	Type	Value
Filter	name	*(Required)* The name of the preferred *security handler* for this document. It shall be the name of the security handler that was used to encrypt the document. If **SubFilter** is not present, only this security handler shall be used when opening the document. If it is present, a conforming reader can use any security handler that implements the format specified by **SubFilter**. **Standard** shall be the name of the built-in password-based security handler. Names for other security handlers may be registered by using the procedure described in Annex E.
SubFilter	name	*(Optional; PDF 1.3)* A name that completely specifies the format and interpretation of the contents of the encryption dictionary. It allows security handlers other than the one specified by **Filter** to decrypt the document. If this entry is absent, other security handlers shall not decrypt the document. NOTE This entry was introduced in PDF 1.3 to support the use of public-key cryptography in PDF files (see 7.6.4, "Public-Key Security Handlers"); however, it was not incorporated into the *PDF Reference* until the fourth edition *(PDF 1.5)*.
V	number	*(Optional)* A code specifying the algorithm to be used in encrypting and decrypting the document: 0 An algorithm that is undocumented. This value shall not be used. 1 "Algorithm 1: Encryption of data using the RC4 or AES algorithms" in 7.6.2, "General Encryption Algorithm," with an encryption key length of 40 bits; see below. 2 *(PDF 1.4)* "Algorithm 1: Encryption of data using the RC4 or AES algorithms" in 7.6.2, "General Encryption Algorithm," but permitting encryption key lengths greater than 40 bits. 3 *(PDF 1.4)* An unpublished algorithm that permits encryption key lengths ranging from 40 to 128 bits. This value shall not appear in a conforming PDF file. 4 *(PDF 1.5)* The security handler defines the use of encryption and decryption in the document, using the rules specified by the **CF**, **StmF**, and **StrF** entries. The default value if this entry is omitted shall be 0, but when present should be a value of 1 or greater.
Length	integer	*(Optional; PDF 1.4; only if V is 2 or 3)* The length of the encryption key, in bits. The value shall be a multiple of 8, in the range 40 to 128. Default value: 40.
CF	dictionary	*(Optional; meaningful only when the value of V is 4; PDF 1.5)* A dictionary whose keys shall be crypt filter names and whose values shall be the corresponding crypt filter dictionaries (see Table 25). Every crypt filter used in the document shall have an entry in this dictionary, except for the standard crypt filter names (see Table 26). The conforming reader shall ignore entries in CF dictionary with the keys equal to those listed in Table 26 and use properties of the respective standard crypt filters.
StmF	name	*(Optional; meaningful only when the value of V is 4; PDF 1.5)* The name of the crypt filter that shall be used by default when decrypting streams. The name shall be a key in the **CF** dictionary or a standard crypt filter name specified in Table 26. All streams in the document, except for cross-reference streams (see 7.5.8, "Cross-Reference Streams") or streams that have a **Crypt** entry in their **Filter** array (see Table 6), shall be decrypted by the security handler, using this crypt filter. Default value: **Identity**.

Table 20 – Entries common to all encryption dictionaries (continued)

Key	Type	Value
StrF	name	*(Optional; meaningful only when the value of V is 4; PDF 1.5)* The name of the crypt filter that shall be used when decrypting all strings in the document. The name shall be a key in the **CF** dictionary or a standard crypt filter name specified in Table 26. Default value: **Identity**.
EFF	name	*(Optional; meaningful only when the value of V is 4; PDF 1.6)* The name of the crypt filter that shall be used when encrypting embedded file streams that do not have their own crypt filter specifier; it shall correspond to a key in the **CF** dictionary or a standard crypt filter name specified in Table 26. This entry shall be provided by the security handler. Conforming writers shall respect this value when encrypting embedded files, except for embedded file streams that have their own crypt filter specifier. If this entry is not present, and the embedded file stream does not contain a crypt filter specifier, the stream shall be encrypted using the default stream crypt filter specified by **StmF**.

Unlike strings within the body of the document, those in the encryption dictionary shall be direct objects. The contents of the encryption dictionary shall not be encrypted (the algorithm specified by the **V** entry). Security handlers shall be responsible for encrypting any data in the encryption dictionary that they need to protect.

NOTE Conforming writers have two choices if the encryption methods and syntax provided by PDF are not sufficient for their needs: they can provide an alternate security handler or they can encrypt whole PDF documents themselves, not making use of PDF security.

7.6.2 General Encryption Algorithm

One of the following algorithms shall be used when encrypting data in a PDF file:

- A proprietary encryption algorithm known as RC4. RC4 is a symmetric stream cipher: the same algorithm shall be used for both encryption and decryption, and the algorithm does not change the length of the data. RC4 is a copyrighted, proprietary algorithm of RSA Security, Inc. Independent software vendors may be required to license RC4 to develop software that encrypts or decrypts PDF documents. For further information, visit the RSA Web site at <http://www.rsasecurity.com> or send e-mail to <products@rsasecurity.com>.

- The AES (Advanced Encryption Standard) algorithm (beginning with PDF 1.6). AES is a symmetric block cipher: the same algorithm shall be used for both encryption and decryption, and the length of the data when encrypted is rounded up to a multiple of the block size, which is fixed to always be 16 bytes, as specified in FIPS 197, *Advanced Encryption Standard (AES)*; see the Bibliography).

  Strings and streams encrypted with AES shall use a padding scheme that is described in Internet RFC 2898, *PKCS #5: Password-Based Cryptography Specification Version 2.0*; see the Bibliography. For an original message length of M, the pad shall consist of 16 - (M mod 16) bytes whose value shall also be 16 - (M mod 16).

EXAMPLE A 9-byte message has a pad of 7 bytes, each with the value 0x07. The pad can be unambiguously removed to determine the original message length when decrypting. Note that the pad is present when M is evenly divisible by 16; it contains 16 bytes of 0x10.

PDF's standard encryption methods also make use of the MD5 message-digest algorithm for key generation purposes (described in Internet RFC 1321, *The MD5 Message-Digest Algorithm*; see the Bibliography).

The encryption of data in a PDF file shall be based on the use of an *encryption key* computed by the security handler. Different security handlers compute the encryption key using their own mechanisms. Regardless of how the key is computed, its use in the encryption of data shall always be the same (see "Algorithm 1:

Encryption of data using the RC4 or AES algorithms"). Because the RC4 algorithm and AES algorithms are symmetric, this same sequence of steps shall be used both to encrypt and to decrypt data.

Algorithms in 7.6, “Encryption” are uniquely numbered within that clause in a manner that maintains compatibility with previous documentation.

Algorithm 1: Encryption of data using the RC4 or AES algorithms

a) Obtain the object number and generation number from the object identifier of the string or stream to be encrypted (see 7.3.10, "Indirect Objects"). If the string is a direct object, use the identifier of the indirect object containing it.

b) For all strings and streams without crypt filter specifier; treating the object number and generation number as binary integers, extend the original *n*-byte encryption key to *n* + 5 bytes by appending the low-order 3 bytes of the object number and the low-order 2 bytes of the generation number in that order, low-order byte first. (*n* is 5 unless the value of **V** in the encryption dictionary is greater than 1, in which case *n* is the value of **Length** divided by 8.)

   If using the AES algorithm, extend the encryption key an additional 4 bytes by adding the value “sAlT”, which corresponds to the hexadecimal values 0x73, 0x41, 0x6C, 0x54. (This addition is done for backward compatibility and is not intended to provide additional security.)

c) Initialize the MD5 hash function and pass the result of step (b) as input to this function.

d) Use the first (*n* + 5) bytes, up to a maximum of 16, of the output from the MD5 hash as the key for the RC4 or AES symmetric key algorithms, along with the string or stream data to be encrypted.

   If using the AES algorithm, the Cipher Block Chaining (CBC) mode, which requires an initialization vector, is used. The block size parameter is set to 16 bytes, and the initialization vector is a 16-byte random number that is stored as the first 16 bytes of the encrypted stream or string.

   The output is the encrypted data to be stored in the PDF file.

Stream data shall be encrypted after applying all stream encoding filters and shall be decrypted before applying any stream decoding filters. The number of bytes to be encrypted or decrypted shall be given by the **Length** entry in the stream dictionary. Decryption of strings (other than those in the encryption dictionary) shall be done after escape-sequence processing and hexadecimal decoding as appropriate to the string representation described in 7.3.4, "String Objects."

7.6.3 Standard Security Handler

7.6.3.1 General

PDF’s standard security handler shall allow *access permissions* and up to two passwords to be specified for a document: an *owner password* and a *user password*. An application’s decision to encrypt a document shall be based on whether the user creating the document specifies any passwords or access restrictions.

EXAMPLE A conforming writer may have a security settings dialog box that the user can invoke before saving the PDF file.

If passwords or access restrictions are specified, the document shall be encrypted, and the permissions and information required to validate the passwords shall be stored in the encryption dictionary. Documents in which only file attachments are encrypted shall use the same password as the *user* and *owner* password.

NOTE 1 A conforming writer may also create an encrypted document without any user interaction if it has some other source of information about what passwords and permissions to use.

If a user attempts to open an encrypted document that has a user password, the conforming reader shall first try to authenticate the encrypted document using the padding string defined in 7.6.3.3, "Encryption Key Algorithm" (default user password):

- If this authentication attempt is successful, the conforming reader may open, decrypt and display the document on the screen.
- If this authentication attempt fails, the application should prompt for a password. Correctly supplying either password (*owner* or *user* password) should enable the user to open the document, decrypt it, and display it on the screen.

Whether additional operations shall be allowed on a decrypted document depends on which password (if any) was supplied when the document was opened and on any access restrictions that were specified when the document was created:

- Opening the document with the correct *owner* password should allow full (owner) access to the document. This unlimited access includes the ability to change the document's passwords and access permissions.
- Opening the document with the correct *user* password (or opening a document with the default password) should allow additional operations to be performed according to the user access permissions specified in the document's encryption dictionary.

Access permissions shall be specified in the form of flags corresponding to the various operations, and the set of operations to which they correspond shall depend on the security handler's revision number (also stored in the encryption dictionary). If the security handler's revision number is 2 or greater, the operations to which user access can be controlled shall be as follows:

- Modifying the document's contents
- Copying or otherwise extracting text and graphics from the document, including extraction for accessibility purposes (that is, to make the contents of the document accessible through assistive technologies such as screen readers or Braille output devices; see 14.9, "Accessibility Support"**.**
- Adding or modifying text annotations (see 12.5.6.4, "Text Annotations") and interactive form fields (see 12.7, "Interactive Forms")
- Printing the document

If the security handler's revision number is 3 or greater, user access to the following operations shall be controlled more selectively:

- Filling in forms (that is, filling in existing interactive form fields) and signing the document (which amounts to filling in existing signature fields, a type of interactive form field).
- Assembling the document: inserting, rotating, or deleting pages and creating navigation elements such as bookmarks or thumbnail images (see 12.3, "Document-Level Navigation").
- Printing to a representation from which a faithful digital copy of the PDF content could be generated. Disallowing such printing may result in degradation of output quality.

In addition, security handlers of revisions 3 and greater shall enable the extraction of text and graphics (in support of accessibility to users with disabilities or for other purposes) to be controlled separately.

If a security handler of revision 4 is specified, the standard security handler shall support crypt filters (see 7.6.5, "Crypt Filters"). The support shall be limited to the **Identity** crypt filter (see Table 26) and crypt filters named **StdCF** whose dictionaries contain a **CFM** value of **V2** or **AESV2** and an **AuthEvent** value of **DocOpen**. Public-Key security handlers in this case shall use crypt filters named **DefaultCryptFilter** when all document content is encrypted, and shall use crypt filters named **DefEmbeddedFile** when file attachments only are encrypted in place of **StdCF** name. This nomenclature shall not be used as indicator of the type of the security handler or encryption.

Once the document has been opened and decrypted successfully, a conforming reader technically has access to the entire contents of the document. There is nothing inherent in PDF encryption that enforces the document permissions specified in the encryption dictionary. Conforming readers shall respect the intent of the document creator by restricting user access to an encrypted PDF file according to the permissions contained in the file.

NOTE 2 PDF 1.5 introduces a set of access permissions that do not require the document to be encrypted (see 12.8.4, "Permissions"). This enables limited access to a document when a user is not be able to respond to a prompt for a password. For example, there may be conforming readers that do not have a person running them such as printing off-line or on a server.

7.6.3.2 Standard Encryption Dictionary

Table 21 shows the encryption dictionary entries for the standard security handler (in addition to those in Table 20).

Table 21 – Additional encryption dictionary entries for the standard security handler

Key	Type	Value
R	number	*(Required)* A number specifying which revision of the standard security handler shall be used to interpret this dictionary: 2 if the document is encrypted with a **V** value less than 2 (see Table 20) and does not have any of the access permissions set to 0 (by means of the **P** entry, below) that are designated “Security handlers of revision 3 or greater” in Table 22 3 if the document is encrypted with a **V** value of 2 or 3, or has any “Security handlers of revision 3 or greater” access permissions set to 0 4 if the document is encrypted with a **V** value of 4
O	string	*(Required)* A 32-byte string, based on both the owner and user passwords, that shall be used in computing the encryption key and in determining whether a valid owner password was entered. For more information, see 7.6.3.3, "Encryption Key Algorithm," and 7.6.3.4, "Password Algorithms."
U	string	*(Required)* A 32-byte string, based on the user password, that shall be used in determining whether to prompt the user for a password and, if so, whether a valid user or owner password was entered. For more information, see 7.6.3.4, "Password Algorithms."
P	integer	*(Required)* A set of flags specifying which operations shall be permitted when the document is opened with user access (see Table 22).
EncryptMetadata	boolean	*(Optional; meaningful only when the value of **V** is 4; PDF 1.5)* Indicates whether the document-level metadata stream (see 14.3.2, "Metadata Streams") shall be encrypted. Conforming products should respect this value. Default value: **true**.

The values of the **O** and **U** entries in this dictionary shall be used to determine whether a password entered when the document is opened is the correct owner password, user password, or neither.

The value of the **P** entry shall be interpreted as an unsigned 32-bit quantity containing a set of flags specifying which access permissions shall be granted when the document is opened with user access. Table 22 shows the meanings of these flags. Bit positions within the flag word shall be numbered from 1 (low-order) to 32 (high-order). A 1 bit in any position shall enable the corresponding access permission. Which bits shall be meaningful, and in some cases how they shall be interpreted, shall depend on the security handler’s revision number (specified in the encryption dictionary’s **R** entry).

Conforming readers shall ignore all flags other than those at bit positions 3, 4, 5, 6, 9, 10, 11, and 12.

NOTE PDF integer objects can be interpreted as binary values in a signed twos-complement form. Since all the reserved high-order flag bits in the encryption dictionary's **P** value are required to be 1, the integer value ***P*** shall be specified as a negative integer. For example, assuming revision 2 of the security handler, the value -44 permits printing and copying but disallows modifying the contents and annotations.

Table 22 – User access permissions

Bit position	**Meaning**
3	*(Security handlers of revision 2)* Print the document. *(Security handlers of revision 3 or greater)* Print the document (possibly not at the highest quality level, depending on whether bit 12 is also set).
4	Modify the contents of the document by operations other than those controlled by bits 6, 9, and 11.
5	*(Security handlers of revision 2)* Copy or otherwise extract text and graphics from the document, including extracting text and graphics (in support of accessibility to users with disabilities or for other purposes). *(Security handlers of revision 3 or greater)* Copy or otherwise extract text and graphics from the document by operations other than that controlled by bit 10.
6	Add or modify text annotations, fill in interactive form fields, and, if bit 4 is also set, create or modify interactive form fields (including signature fields).
9	*(Security handlers of revision 3 or greater)* Fill in existing interactive form fields (including signature fields), even if bit 6 is clear.
10	*(Security handlers of revision 3 or greater)* Extract text and graphics (in support of accessibility to users with disabilities or for other purposes).
11	*(Security handlers of revision 3 or greater)* Assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images), even if bit 4 is clear.
12	*(Security handlers of revision 3 or greater)* Print the document to a representation from which a faithful digital copy of the PDF content could be generated. When this bit is clear (and bit 3 is set), printing is limited to a low-level representation of the appearance, possibly of degraded quality.

7.6.3.3 Encryption Key Algorithm

As noted earlier, one function of a security handler is to generate an encryption key for use in encrypting and decrypting the contents of a document. Given a password string, the standard security handler computes an encryption key as shown in "Algorithm 2: Computing an encryption key".

Algorithm 2: Computing an encryption key

a) Pad or truncate the password string to exactly 32 bytes. If the password string is more than 32 bytes long, use only its first 32 bytes; if it is less than 32 bytes long, pad it by appending the required number of additional bytes from the beginning of the following padding string:

```
< 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08
  2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >
```

That is, if the password string is *n* bytes long, append the first 32 - *n* bytes of the padding string to the end of the password string. If the password string is empty (zero-length), meaning there is no user password, substitute the entire padding string in its place.

b) Initialize the MD5 hash function and pass the result of step (a) as input to this function.

c) Pass the value of the encryption dictionary's **O** entry to the MD5 hash function. ("Algorithm 3: Computing the encryption dictionary's O (owner password) value" shows how the **O** value is computed.)

d) Convert the integer value of the **P** entry to a 32-bit unsigned binary number and pass these bytes to the MD5 hash function, low-order byte first.

e) Pass the first element of the file's file identifier array (the value of the **ID** entry in the document's trailer dictionary; see Table 15) to the MD5 hash function.

NOTE The first element of the ID array generally remains the same for a given document. However, in some situations, conforming writers may regenerate the ID array if a new generation of a document is created. Security handlers are encouraged not to rely on the ID in the encryption key computation.

f) *(Security handlers of revision 4 or greater)* If document metadata is not being encrypted, pass 4 bytes with the value 0xFFFFFFFF to the MD5 hash function.

g) Finish the hash.

h) *(Security handlers of revision 3 or greater)* Do the following 50 times: Take the output from the previous MD5 hash and pass the first *n* bytes of the output as input into a new MD5 hash, where *n* is the number of bytes of the encryption key as defined by the value of the encryption dictionary's **Length** entry.

i) Set the encryption key to the first *n* bytes of the output from the final MD5 hash, where *n* shall always be 5 for security handlers of revision 2 but, for security handlers of revision 3 or greater, shall depend on the value of the encryption dictionary's **Length** entry.

This algorithm, when applied to the user password string, produces the encryption key used to encrypt or decrypt string and stream data according to "Algorithm 1: Encryption of data using the RC4 or AES algorithms" in 7.6.2, "General Encryption Algorithm." Parts of this algorithm are also used in the algorithms described below.

7.6.3.4 Password Algorithms

In addition to the encryption key, the standard security handler shall provide the contents of the encryption dictionary (Table 20 and Table 21). The values of the **Filter**, **V**, **Length**, **R**, and **P** entries are straightforward, but the computation of the **O** (owner password) and **U** (user password) entries requires further explanation. The algorithms 3 through 7 that follow show how the values of the owner password and user password entries shall be computed (with separate versions of the latter depending on the revision of the security handler).

Algorithm 3: Computing the encryption dictionary's O (owner password) value

a) Pad or truncate the owner password string as described in step (a) of "Algorithm 2: Computing an encryption key". If there is no owner password, use the user password instead.

b) Initialize the MD5 hash function and pass the result of step (a) as input to this function.

c) *(Security handlers of revision 3 or greater)* Do the following 50 times: Take the output from the previous MD5 hash and pass it as input into a new MD5 hash.

d) Create an RC4 encryption key using the first *n* bytes of the output from the final MD5 hash, where *n* shall always be 5 for security handlers of revision 2 but, for security handlers of revision 3 or greater, shall depend on the value of the encryption dictionary's **Length** entry.

e) Pad or truncate the user password string as described in step (a) of "Algorithm 2: Computing an encryption key".

f) Encrypt the result of step (e), using an RC4 encryption function with the encryption key obtained in step (d).

g) *(Security handlers of revision 3 or greater)* Do the following 19 times: Take the output from the previous invocation of the RC4 function and pass it as input to a new invocation of the function; use an encryption key generated by taking each byte of the encryption key obtained in step (d) and performing an XOR (exclusive or) operation between that byte and the single-byte value of the iteration counter (from 1 to 19).

h) Store the output from the final invocation of the RC4 function as the value of the **O** entry in the encryption dictionary.

Algorithm 4: Computing the encryption dictionary's U (user password) value (Security handlers of revision 2)

a) Create an encryption key based on the user password string, as described in "Algorithm 2: Computing an encryption key".

b) Encrypt the 32-byte padding string shown in step (a) of "Algorithm 2: Computing an encryption key", using an RC4 encryption function with the encryption key from the preceding step.

c) Store the result of step (b) as the value of the **U** entry in the encryption dictionary.

Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or greater)

a) Create an encryption key based on the user password string, as described in "Algorithm 2: Computing an encryption key".

b) Initialize the MD5 hash function and pass the 32-byte padding string shown in step (a) of "Algorithm 2: Computing an encryption key" as input to this function.

c) Pass the first element of the file's file identifier array (the value of the **ID** entry in the document's trailer dictionary; see Table 15) to the hash function and finish the hash.

d) Encrypt the 16-byte result of the hash, using an RC4 encryption function with the encryption key from step (a).

e) Do the following 19 times: Take the output from the previous invocation of the RC4 function and pass it as input to a new invocation of the function; use an encryption key generated by taking each byte of the original encryption key obtained in step (a) and performing an XOR (exclusive or) operation between that byte and the single-byte value of the iteration counter (from 1 to 19).

f) Append 16 bytes of arbitrary padding to the output from the final invocation of the RC4 function and store the 32-byte result as the value of the **U** entry in the encryption dictionary.

NOTE The standard security handler uses the algorithms 6 and 7 that follow, to determine whether a supplied password string is the correct user or owner password. Note too that algorithm 6 can be used to determine whether a document's user password is the empty string, and therefore whether to suppress prompting for a password when the document is opened.

Algorithm 6: Authenticating the user password

a) Perform all but the last step of "Algorithm 4: Computing the encryption dictionary's U (user password) value (Security handlers of revision 2)" or "Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or greater)" using the supplied password string.

b) If the result of step (a) is equal to the value of the encryption dictionary's **U** entry (comparing on the first 16 bytes in the case of security handlers of revision 3 or greater), the password supplied is the correct user password. The key obtained in step (a) (that is, in the first step of "Algorithm 4: Computing the encryption dictionary's U (user password) value (Security handlers of revision 2)" or "Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or greater)") shall be used to decrypt the document.

Algorithm 7: Authenticating the owner password

a) Compute an encryption key from the supplied password string, as described in steps (a) to (d) of "Algorithm 3: Computing the encryption dictionary's O (owner password) value".

b) *(Security handlers of revision 2 only)* Decrypt the value of the encryption dictionary's **O** entry, using an RC4 encryption function with the encryption key computed in step (a).

   (Security handlers of revision 3 or greater) Do the following 20 times: Decrypt the value of the encryption dictionary's **O** entry (first iteration) or the output from the previous iteration (all subsequent iterations), using an RC4 encryption function with a different encryption key at each iteration. The key shall be generated by taking the original key (obtained in step (a)) and performing an XOR (exclusive or) operation between each byte of the key and the single-byte value of the iteration counter (from 19 to 0).

c) The result of step (b) purports to be the user password. Authenticate this user password using "Algorithm 6: Authenticating the user password". If it is correct, the password supplied is the correct owner password.

7.6.4 Public-Key Security Handlers

7.6.4.1 General

Security handlers may use *public-key* encryption technology to encrypt a document (or strings and streams within a document). When doing so, specifying one or more lists of recipients, where each list has its own unique access permissions may be done. Only specified recipients shall open the encrypted document or content, unlike the standard security handler, where a password determines access. The permissions defined for public-key security handlers are shown in Table 24 in 7.6.4.2, "Public-Key Encryption Dictionary".

Public-key security handlers use the industry standard Public Key Cryptographic Standard Number 7 (PKCS#7) binary encoding syntax to encode recipient list, decryption key, and access permission information. The PKCS#7 specification is in Internet RFC 2315, *PKCS #7: Cryptographic Message Syntax, Version 1.5* (see the Bibliography).

When encrypting the data, each recipient's X.509 public key certificate (as described in ITU-T Recommendation X.509; see the Bibliography) shall be available. When decrypting the data, the conforming reader shall scan the recipient list for which the content is encrypted and shall attempt to find a match with a certificate that belongs to the user. If a match is found, the user requires access to the corresponding private key, which may require authentication, possibly using a password. Once access is obtained, the private key shall be used to decrypt the encrypted data.

7.6.4.2 Public-Key Encryption Dictionary

Encryption dictionaries for public-key security handlers contain the common entries shown in Table 20, whose values are described above. In addition, they may contain the entry shown in Table 23 as described below.

The **Filter** entry shall be the name of a public-key security handler.

NOTE Examples of existing security handlers that support public-key encryption are **Entrust.PPKEF**, **Adobe.PPKLite**, and **Adobe.PubSec**. This handler will be the preferred handler when encrypting the document.

Permitted values of the **SubFilter** entry for use with conforming public-key security handlers are **adbe.pkcs7.s3**, **adbe.pkcs7.s4**, which shall be used when not using crypt filters (see 7.6.5, "Crypt Filters") and **adbe.pkcs7.s5**, which shall be used when using crypt filters.

The **CF**, **StmF**, and **StrF** entries may be present when **SubFilter** is **adbe.pkcs7.s5.**

Table 23 – Additional encryption dictionary entries for public-key security handlers

Key	Type	Value
Recipients	array	*(Required when **SubFilter** is **adbe.pkcs7.s3** or **adbe.pkcs7.s4**; PDF 1.3)* An array of byte-strings, where each string is a PKCS#7 object listing recipients who have been granted equal access rights to the document. The data contained in the PKCS#7 object shall include both a cryptographic key that shall be used to decrypt the encrypted data and the access permissions (see Table 24) that apply to the recipient list. There shall be only one PKCS#7 object per unique set of access permissions; if a recipient appears in more than one list, the permissions used shall be those in the first matching list. When ***SubFilter*** *is* ***adbe.pkcs7.s5***, recipient lists shall be specified in the crypt filter dictionary; see Table 27.
P	integer	(*Required*) A set of flags specifying which operations shall be permitted when the document is opened with user access. If bit 2 is set to 1, all other bits are ignored and all operations are permitted. If bit 2 is set to 0, permission for operations are based on the values of the remaining flags defined in Table 24.

The value of the **P** entry shall be interpreted as an unsigned 32-bit quantity containing a set of flags specifying which access permissions shall be granted when the document is opened with user access. Table 24 shows the meanings of these flags. Bit positions within the flag word shall be numbered from 1 (low-order) to 32 (high-order). A 1 bit in any position shall enable the corresponding access permission.

Conforming readers shall ignore all flags other than those at bit positions 2, 3, 4, 5, 6, 9, 10, 11, and 12.

Table 24 – Public-Key security handler user access permissions

Bit position	Meaning
2	When set permits change of encryption and enables all other permissions.
3	Print the document (possibly not at the highest quality level, depending on whether bit 12 is also set).
4	Modify the contents of the document by operations other than those controlled by bits 6, 9, and 11.
5	Copy or otherwise extract text and graphics from the document by operations other than that controlled by bit 10.
6	Add or modify text annotations, fill in interactive form fields, and, if bit 4 is also set, create or modify interactive form fields (including signature fields).
9	Fill in existing interactive form fields (including signature fields), even if bit 6 is clear.
10	Extract text and graphics (in support of accessibility to users with disabilities or for other purposes).

Table 24 – Public-Key security handler user access permissions (continued)

Bit position	Meaning
11	Assemble the document (insert, rotate, or delete pages and create bookmarks or thumbnail images), even if bit 4 is clear.
12	Print the document to a representation from which a faithful digital copy of the PDF content could be generated. When this bit is clear (and bit 3 is set), printing is limited to a low-level representation of the appearance, possibly of degraded quality.

7.6.4.3 Public-Key Encryption Algorithms

Figure 4 illustrates how PKCS#7 objects shall be used when encrypting PDF files. A PKCS#7 object is designed to encapsulate and encrypt what is referred to as the *enveloped data*.

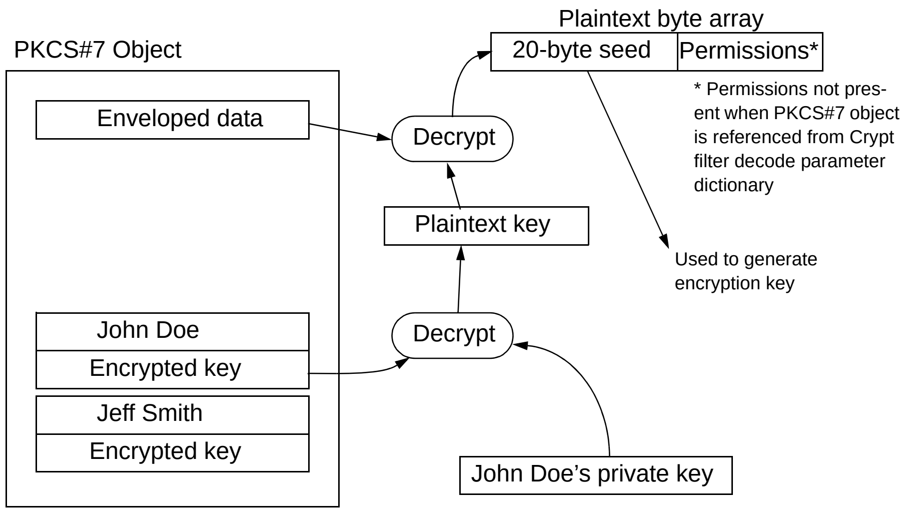


Figure 4 – Public-key encryption algorithm

The enveloped data in the PKCS#7 object contains keying material that shall be used to decrypt the document (or individual strings or streams in the document, when crypt filters are used; see 7.6.5, "Crypt Filters"). A key shall be used to encrypt (and decrypt) the enveloped data. This key (the *plaintext key* in Figure 4) shall be encrypted for each recipient, using that recipient's public key, and shall be stored in the PKCS#7 object (as the *encrypted key* for each recipient). To decrypt the document, that key shall be decrypted using the recipient's private key, which yields a decrypted (plaintext) key. That key, in turn, shall be used to decrypt the enveloped data in the PKCS#7 object, resulting in a byte array that includes the following information:

- A 20-byte seed that shall be used to create the encryption key that is used by "Algorithm 1: Encryption of data using the RC4 or AES algorithms". The seed shall be a unique random number generated by the security handler that encrypted the document.
- A 4-byte value defining the permissions, least significant byte first. See Table 24 for the possible permission values.
- When **SubFilter** is **adbe.pkcs7.s3**, the relevant permissions shall be only those specified for revision 2 of the standard security handler.
- For **adbe.pkcs7.s4**, security handlers of revision 3 permissions shall apply.
- For **adbe.pkcs7.s5**, which supports the use of crypt filters, the permissions shall be the same as **adbe.pkcs7.s4** when the crypt filter is referenced from the **StmF** or **StrF** entries of the encryption

dictionary. When referenced from the **Crypt** filter decode parameter dictionary of a stream object (see Table 14), the 4 bytes of permissions shall be absent from the enveloped data.

The algorithms that shall be used to encrypt the enveloped data in the PKCS#7 object are: RC4 with key lengths up to 256-bits, DES, Triple DES, RC2 with key lengths up to 128 bits, 128-bit AES in Cipher Block Chaining (CBC) mode, 192-bit AES in CBC mode, 256-bit AES in CBC mode. The PKCS#7 specification is in Internet RFC 2315, *PKCS #7: Cryptographic Message Syntax, Version 1.5* (see the Bibliography).

The encryption key used by "Algorithm 1: Encryption of data using the RC4 or AES algorithms" shall be calculated by means of an SHA-1 message digest operation that digests the following data, in order:

a) The 20 bytes of seed

b) The bytes of each item in the **Recipients** array of PKCS#7 objects in the order in which they appear in the array

c) 4 bytes with the value 0xFF if the key being generated is intended for use in document-level encryption and the document metadata is being left as plaintext

The first *n*/8 bytes of the resulting digest shall be used as the encryption key, where *n* is the bit length of the encryption key.

7.6.5 Crypt Filters

PDF 1.5 introduces *crypt filters,* which provide finer granularity control of encryption within a PDF file. The use of crypt filters involves the following structures:

- The encryption dictionary (see Table 20) contains entries that enumerate the crypt filters in the document (**CF**) and specify which ones are used by default to decrypt all the streams (**StmF**) and strings (**StrF**) in the document. In addition, the value of the **V** entry shall be 4 to use crypt filters.

- Each crypt filter specified in the **CF** entry of the encryption dictionary shall be represented by a *crypt filter dictionary*, whose entries are shown in Table 25.

- A stream filter type, the **Crypt** filter (see 7.4.10, "Crypt Filter") can be specified for any stream in the document to override the default filter for streams. A conforming reader shall provide a standard **Identity** filter which shall pass the data unchanged (see Table 26) to allow specific streams, such as document metadata, to be unencrypted in an otherwise encrypted document. The stream's **DecodeParms** entry shall contain a **Crypt** filter decode parameters dictionary (see Table 14) whose **Name** entry specifies the particular crypt filter to be used (if missing, **Identity** is used). Different streams may specify different crypt filters.

Authorization to decrypt a stream shall always be obtained before the stream can be accessed. This typically occurs when the document is opened, as specified by a value of **DocOpen** for the **AuthEvent** entry in the crypt filter dictionary. Conforming readers and security handlers shall treat any attempt to access a stream for which authorization has failed as an error. **AuthEvent** can also be **EFOpen**, which indicates the presence of an embedded file that is encrypted with a crypt filter that may be different from the crypt filters used by default to encrypt strings and streams in the document.

In the file specification dictionary (see 7.11.3, "File Specification Dictionaries"), related files (**RF**) shall use the same crypt filter as the embedded file (**EF**).

A value of **None** for the **CFM** entry in the crypt filter dictionary allows the security handler to do its own decryption. This allows the handler to tightly control key management and use any preferred symmetric-key cryptographic algorithm.

Table 25 – Entries common to all crypt filter dictionaries

Key	Type	Value
Type	name	*(Optional)* If present, shall be **CryptFilter** for a crypt filter dictionary.
CFM	name	*(Optional)* The method used, if any, by the conforming reader to decrypt data. The following values shall be supported: **None** The application shall not decrypt data but shall direct the input stream to the security handler for decryption. **V2** The application shall ask the security handler for the encryption key and shall implicitly decrypt data with "Algorithm 1: Encryption of data using the RC4 or AES algorithms", using the RC4 algorithm. **AESV2** *(PDF 1.6)* The application shall ask the security handler for the encryption key and shall implicitly decrypt data with "Algorithm 1: Encryption of data using the RC4 or AES algorithms", using the AES algorithm in Cipher Block Chaining (CBC) mode with a 16-byte block size and an initialization vector that shall be randomly generated and placed as the first 16 bytes in the stream or string. When the value is **V2** or **AESV2**, the application may ask once for this encryption key and cache the key for subsequent use for streams that use the same crypt filter. Therefore, there shall be a one-to-one relationship between a crypt filter name and the corresponding encryption key. Only the values listed here shall be supported. Applications that encounter other values shall report that the file is encrypted with an unsupported algorithm. Default value: **None**.
AuthEvent	name	*(Optional)* The event to be used to trigger the authorization that is required to access encryption keys used by this filter. If authorization fails, the event shall fail. Valid values shall be: **DocOpen**: Authorization shall be required when a document is opened. **EFOpen**: Authorization shall be required when accessing embedded files. Default value: **DocOpen**. If this filter is used as the value of **StrF** or **StmF** in the encryption dictionary (see Table 20), the conforming reader shall ignore this key and behave as if the value is **DocOpen**.
Length	integer	*(Optional)* The bit length of the encryption key. It shall be a multiple of 8 in the range of 40 to 128. Security handlers may define their own use of the ***Length*** entry and should use it to define the bit length of the encryption key. Standard security handler expresses the length in multiples of 8 (16 means 128) and public-key security handler expresses it as is (128 means 128).

Security handlers may add their own private data to crypt filter dictionaries. Names for private data entries shall conform to the PDF name registry (see Annex E).

Table 26 – Standard crypt filter names

Name	Description
Identity	Input data shall be passed through without any processing.

Table 27 lists the additional crypt filter dictionary entries used by public-key security handlers (see 7.6.4, "Public-Key Security Handlers"). When these entries are present, the value of **CFM** shall be **V2** or **AESV2**.

Table 27 – Additional crypt filter dictionary entries for public-key security handlers

Key	Type	Value
Recipients	array or string	*(Required)* If the crypt filter is referenced from **StmF** or **StrF** in the encryption dictionary, this entry shall be an array of byte strings, where each string shall be a binary-encoded PKCS#7 object that shall list recipients that have been granted equal access rights to the document. The enveloped data contained in the PKCS#7 object shall include both a 20-byte seed value that shall be used to compute the encryption key (see 7.6.4.3, "Public-Key Encryption Algorithms") followed by 4 bytes of permissions settings (see Table 22) that shall apply to the recipient list. There shall be only one object per unique set of access permissions. If a recipient appears in more than one list, the permissions used shall be those in the first matching list. If the crypt filter is referenced from a **Crypt** filter decode parameter dictionary (see Table 14), this entry shall be a string that shall be a binary-encoded PKCS#7 object shall contain a list of all recipients who are permitted to access the corresponding encrypted stream. The enveloped data contained in the PKCS#7 object shall be a 20-byte seed value that shall be used to create the encryption key that shall be used by the algorithm in "Algorithm 1: Encryption of data using the RC4 or AES algorithms".
EncryptMetadata	boolean	*(Optional; used only by crypt filters that are referenced from* ***StmF*** *in an encryption dictionary)* Indicates whether the document-level metadata stream (see 14.3.2, "Metadata Streams") shall be encrypted. Conforming readers shall respect this value when determining whether metadata shall be encrypted. The value of the EncryptMetadata entry is set by the security handler rather than the conforming reader. Default value: **true**.

EXAMPLE The following shows the use of crypt filters in an encrypted document containing a plaintext document-level metadata stream. The metadata stream is left as is by applying the **Identity** crypt filter. The remaining streams and strings are decrypted using the default filters.

```
%PDF–1.5
1 0 obj                          % Document catalog
   << /Type /Catalog
      /Pages 2 0 R
      /Metadata 6 0 R
   >>
endobj
2 0 obj                          % Page tree
   << /Type /Pages
      /Kids [3 0 R]
      /Count 1
   >>
endobj
3 0 obj                          % 1s t page
   << /Type /Page
```

```
          /Parent 2 0 R
          /MediaBox [0 0 612 792]
          /Contents 4 0 R
     >>
endobj
4 0 obj                                   % Page contents
     << /Length 35 >>
     stream
          *** Encrypted Page-marking operators ***
     endstream
endobj
5 0 obj
     << /Title ($#*#%*$#^&##) >>     % Info dictionary: encrypted text string
endobj
6 0 obj
     << /Type /Metadata
          /Subtype /XML
          /Length 15
          /Filter [/Crypt]               % Uses a crypt filter
          /DecodeParms                   % with these parameters
          << /Type /CryptFilterDecodeParms
               /Name /Identity                    % Indicates no encryption
          >>
     >>
     stream
          XML metadata                            % Unencrypted metadata
     endstream
endobj
8 0 obj                                           % Encryption dictionary
     << /Filter /MySecurityHandlerName
          /V 4                                    % Version 4: allow crypt filters
          /CF                                     % List of crypt filters
          << /MyFilter0
               << /Type /CryptFilter
                    /CFM V2  >>                   % Uses the standard algorithm
          >>
          /StrF /MyFilter0                        % Strings are decrypted using /MyFilter0
          /StmF /MyFilter0                        % Streams are decrypted using /MyFilter0
               ...                                % Private data for /MySecurityHandlerName
          /MyUnsecureKey (12345678)
          /EncryptMetadata false
     >>
endobj
xref
     ....
trailer
     << /Size 8
          /Root 1 0 R
          /Info 5 0 R
          /Encrypt 8 0 R
     >>
startxref
495
%%EOF
```

7.7 Document Structure

7.7.1 General

A PDF document can be regarded as a hierarchy of objects contained in the body section of a PDF file. At the root of the hierarchy is the document's *catalog* dictionary (see 7.7.2, "Document Catalog").

NOTE Most of the objects in the hierarchy are dictionaries. Figure 5 illustrates the structure of the object hierarchy.

EXAMPLE Each page of the document is represented by a *page object*—a dictionary that includes references to the page's contents and other attributes, such as its thumbnail image (12.3.4, "Thumbnail Images") and any annotations (12.5, "Annotations") associated with it. The individual page objects are tied together in a structure called the *page tree* (described in 7.7.3, "Page Tree"), which in turn is specified by an indirect reference in the document catalog. Parent, child, and sibling relationships within the hierarchy are defined by dictionary entries whose values are indirect references to other dictionaries.

The data structures described in this sub-clause, particularly the **Catalog** and **Page** dictionaries, combine entries describing document structure with ones dealing with the detailed semantics of documents and pages. All entries are listed here, but many of their descriptions are deferred to subsequent sub-clauses.

7.7.2 Document Catalog

The root of a document's object hierarchy is the *catalog* dictionary, located by means of the **Root** entry in the trailer of the PDF file (see 7.5.5, "File Trailer"). The catalog contains references to other objects defining the document's contents, outline, article threads, named destinations, and other attributes. In addition, it contains information about how the document shall be displayed on the screen, such as whether its outline and thumbnail page images shall be displayed automatically and whether some location other than the first page shall be shown when the document is opened. Table 28 shows the entries in the catalog dictionary.

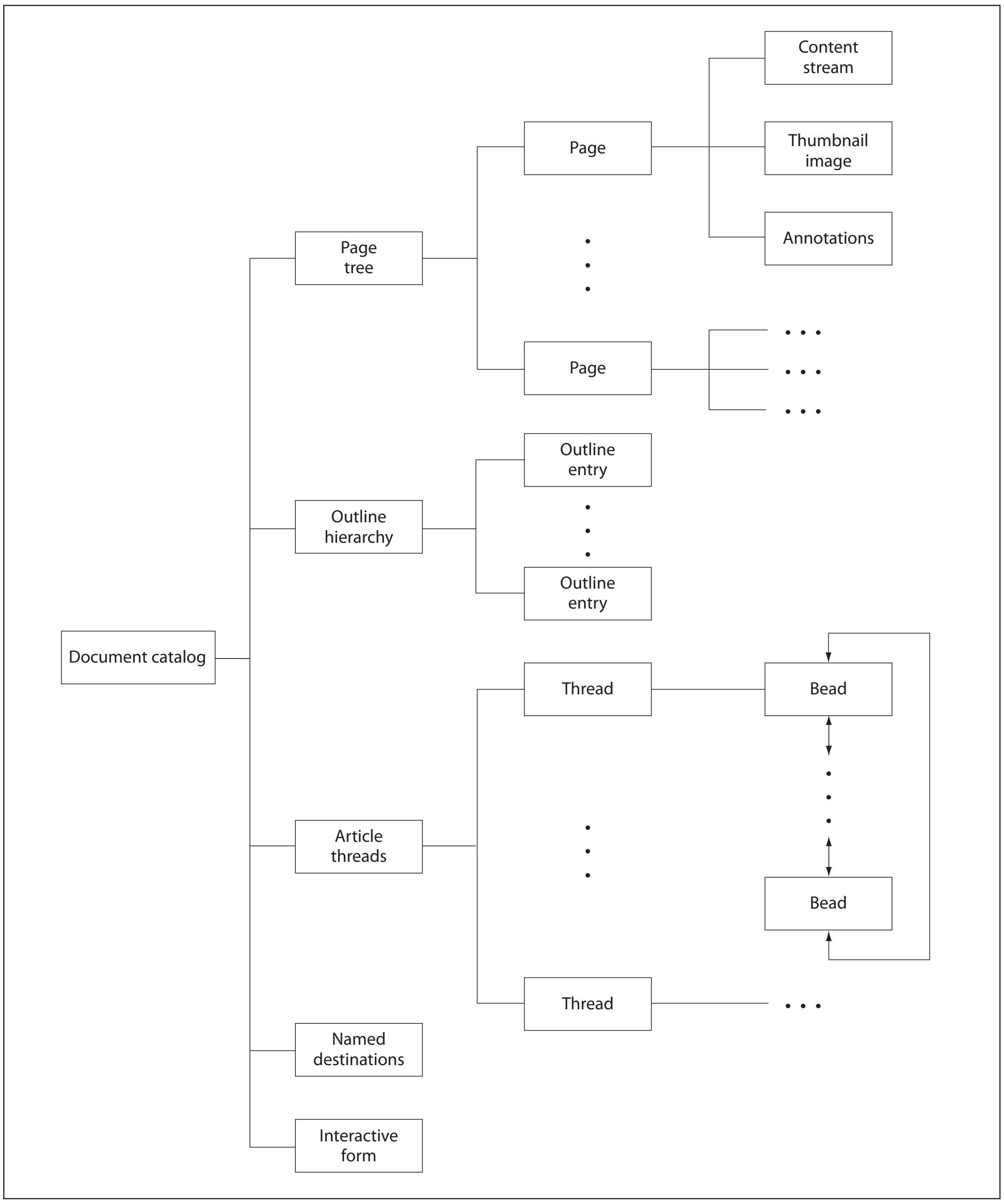


Figure 5 – Structure of a PDF document

Table 28 – Entries in the catalog dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Catalog** for the catalog dictionary.
Version	name	*(Optional; PDF 1.4)* The version of the PDF specification to which the document conforms (for example, 1.4) if later than the version specified in the file's header (see 7.5.2, "File Header"). If the header specifies a later version, or if this entry is absent, the document shall conform to the version specified in the header. This entry enables a conforming writer to update the version using an incremental update; see 7.5.6, "Incremental Updates." The value of this entry shall be a name object, not a number, and therefore shall be preceded by a SOLIDUS (2Fh) character (/) when written in the PDF file (for example, /1.4).
Extensions	dictionary	*(Optional; ISO 32000)* An extensions dictionary containing developer prefix identification and version numbers for developer extensions that occur in this document. 7.12, "Extensions Dictionary", describes this dictionary and how it shall be used.
Pages	dictionary	*(Required; shall be an indirect reference)* The *page tree node* that shall be the root of the document's *page tree* (see 7.7.3, "Page Tree").
PageLabels	number tree	*(Optional; PDF 1.3)* A number tree (see 7.9.7, "Number Trees") defining the page labelling for the document. The keys in this tree shall be page indices; the corresponding values shall be *page label dictionaries* (see 12.4.2, "Page Labels"). Each page index shall denote the first page in a *labelling range* to which the specified page label dictionary applies. The tree shall include a value for page index 0.
Names	dictionary	*(Optional; PDF 1.2)* The document's name dictionary (see 7.7.4, "Name Dictionary").
Dests	dictionary	*(Optional; PDF 1.1; shall be an indirect reference)* A dictionary of names and corresponding *destinations* (see 12.3.2.3, "Named Destinations").
ViewerPreferences	dictionary	*(Optional; PDF 1.2)* A viewer preferences dictionary (see 12.2, "Viewer Preferences") specifying the way the document shall be displayed on the screen. If this entry is absent, conforming readers shall use their own current user preference settings.
PageLayout	name	*(Optional)* A name object specifying the page layout shall be used when the document is opened: **SinglePage** Display one page at a time **OneColumn** Display the pages in one column **TwoColumnLeft** Display the pages in two columns, with odd-numbered pages on the left **TwoColumnRight** Display the pages in two columns, with odd-numbered pages on the right **TwoPageLeft** *(PDF 1.5)* Display the pages two at a time, with odd-numbered pages on the left **TwoPageRight** *(PDF 1.5)* Display the pages two at a time, with odd-numbered pages on the right Default value: SinglePage.

Table 28 – Entries in the catalog dictionary (continued)

Key	Type	Value
PageMode	name	*(Optional)* A name object specifying how the document shall be displayed when opened: **UseNone** Neither document outline nor thumbnail images visible **UseOutlines** Document outline visible **UseThumbs** Thumbnail images visible **FullScreen** Full-screen mode, with no menu bar, window controls, or any other window visible **UseOC** *(PDF 1.5)* Optional content group panel visible **UseAttachments** *(PDF 1.6)* Attachments panel visible Default value: UseNone.
Outlines	dictionary	*(Optional; shall be an indirect reference)* The outline dictionary that shall be the root of the document's *outline hierarchy* (see 12.3.3, "Document Outline").
Threads	array	*(Optional; PDF 1.1; shall be an indirect reference)* An array of thread dictionaries that shall represent the document's *article threads* (see 12.4.3, "Articles").
OpenAction	array or dictionary	*(Optional; PDF 1.1)* A value specifying a *destination* that shall be displayed or an *action* that shall be performed when the document is opened. The value shall be either an array defining a destination (see 12.3.2, "Destinations") or an action dictionary representing an action (12.6, "Actions"). If this entry is absent, the document shall be opened to the top of the first page at the default magnification factor.
AA	dictionary	*(Optional; PDF 1.4)* An additional-actions dictionary defining the actions that shall be taken in response to various *trigger events* affecting the document as a whole (see 12.6.3, "Trigger Events").
URI	dictionary	*(Optional; PDF 1.1)* A URI dictionary containing document-level information for *URI (uniform resource identifier) actions* (see 12.6.4.7, "URI Actions").
AcroForm	dictionary	*(Optional; PDF 1.2)* The document's *interactive form (AcroForm) dictionary* (see 12.7.2, "Interactive Form Dictionary").
Metadata	stream	*(Optional; PDF 1.4; shall be an indirect reference)* A *metadata stream* that shall contain metadata for the document (see 14.3.2, "Metadata Streams").
StructTreeRoot	dictionary	*(Optional; PDF 1.3)* The document's structure tree root dictionary (see 14.7.2, "Structure Hierarchy").
MarkInfo	dictionary	*(Optional; PDF 1.4)* A mark information dictionary that shall contain information about the document's usage of Tagged PDF conventions (see 14.7, "Logical Structure").
Lang	text string	*(Optional; PDF 1.4)* A *language identifier* that shall specify the natural language for all text in the document except where overridden by language specifications for structure elements or marked content (see 14.9.2, "Natural Language Specification"). If this entry is absent, the language shall be considered unknown.
SpiderInfo	dictionary	*(Optional; PDF 1.3)* A Web Capture information dictionary that shall contain state information used by any Web Capture extension (see 14.10.2, "Web Capture Information Dictionary").

Table 28 – Entries in the catalog dictionary (continued)

Key	Type	Value
OutputIntents	array	*(Optional; PDF 1.4)* An array of output intent dictionaries that shall specify the colour characteristics of output devices on which the document might be rendered (see 14.11.5, "Output Intents").
PieceInfo	dictionary	*(Optional; PDF 1.4)* A page-piece dictionary associated with the document (see 14.5, "Page-Piece Dictionaries").
OCProperties	dictionary	*(Optional; PDF 1.5; required if a document contains optional content)* The document's optional content properties dictionary (see 8.11.4, "Configuring Optional Content").
Perms	dictionary	*(Optional; PDF 1.5)* A permissions dictionary that shall specify user access permissions for the document. 12.8.4, "Permissions", describes this dictionary and how it shall be used.
Legal	dictionary	*(Optional; PDF 1.5)* A dictionary that shall contain attestations regarding the content of a PDF document, as it relates to the legality of digital signatures (see 12.8.5, "Legal Content Attestations").
Requirements	array	*(Optional; PDF 1.7)* An array of requirement dictionaries that shall represent requirements for the document. 12.10, "Document Requirements", describes this dictionary and how it shall be used.
Collection	dictionary	(*Optional; PDF 1.7*) A collection dictionary that a conforming reader shall use to enhance the presentation of file attachments stored in the PDF document. (see 12.3.5, "Collections").
NeedsRendering	boolean	(*Optional; PDF 1.7*) A flag used to expedite the display of PDF documents containing XFA forms. It specifies whether the document shall be regenerated when the document is first opened. See the *XML Forms Architecture (XFA) Specification* (Bibliography). Default value: **false**.

EXAMPLE The following shows a sample catalog object.

```
1 0 obj
   << /Type /Catalog
        /Pages 2 0 R
        /PageMode /UseOutlines
        /Outlines 3 0 R
   >>
endobj
```

7.7.3 Page Tree

7.7.3.1 General

The pages of a document are accessed through a structure known as the *page tree*, which defines the ordering of pages in the document. Using the tree structure, conforming readers using only limited memory, can quickly open a document containing thousands of pages. The tree contains nodes of two types—intermediate nodes, called *page tree nodes*, and leaf nodes, called *page objects*—whose form is described in the subsequent sub-clauses. Conforming products shall be prepared to handle any form of tree structure built of such nodes.

NOTE The simplest structure can consist of a single page tree node that references all of the document's page objects directly. However, to optimize application performance, a conforming writer can construct trees of a particular form, known as *balanced trees*. Further information on this form of tree can be found in *Data Structures and Algorithms*, by Aho, Hopcroft, and Ullman (see the Bibliography).

7.7.3.2 Page Tree Nodes

Table 29 shows the entries in a page tree node that shall always be present (Required).

Table 29 – Required entries in a page tree node

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Pages** for a page tree node.
Parent	dictionary	*(Required except in root node; prohibited in the root node; shall be an indirect reference)* The page tree node that is the immediate parent of this one.
Kids	array	*(Required)* An array of indirect references to the immediate children of this node. The children shall only be page objects or other page tree nodes.
Count	integer	*(Required)* The number of leaf nodes (page objects) that are descendants of this node within the page tree.

NOTE The structure of the page tree is not necessarily related to the logical structure of the document; that is, page tree nodes do not represent chapters, sections, and so forth. Other data structures are defined for this purpose; see 14.7, "Logical Structure".

Conforming products shall not be required to preserve the existing structure of the page tree.

EXAMPLE The following illustrates the page tree for a document with three pages. See 7.7.3.3, "Page Objects," for the contents of the individual page objects, and H.5, "Page Tree Example", for a more extended example showing the page tree for a longer document.

```
2  0  obj
   << /Type  /Pages
      /Kids   [ 4 0 R
                10 0 R
                24 0 R
              ]
      /Count  3
   >>
endobj

4  0  obj
   << /Type  /Page
       …Additional entries describing the attributes of this page…
   >>
endobj

10  0  obj
   << /Type  /Page
       …Additional entries describing the attributes of this page…
   >>
endobj

24  0  obj
   << /Type  /Page
       …Additional entries describing the attributes of this page…
   >>
endobj
```

In addition to the entries shown in Table 29, a page tree node may contain further entries defining *inherited attributes* for the page objects that are its descendants (see 7.7.3.4, "Inheritance of Page Attributes").

7.7.3.3 Page Objects

The leaves of the page tree are *page objects*, each of which is a dictionary specifying the attributes of a single page of the document. Table 30 shows the contents of this dictionary. The table also identifies which attributes a page may inherit from its ancestor nodes in the page tree, as described under 7.7.3.4, "Inheritance of Page Attributes." Attributes that are not explicitly identified in the table as inheritable shall not be inherited.

Table 30 – Entries in a page object

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Page** for a page object.
Parent	dictionary	*(Required; shall be an indirect reference)* The page tree node that is the immediate parent of this page object.
LastModified	date	*(Required if **PieceInfo** is present; optional otherwise; PDF 1.3)* The date and time (see 7.9.4, "Dates") when the page's contents were most recently modified. If a page-piece dictionary (**PieceInfo**) is present, the modification date shall be used to ascertain which of the application data dictionaries that it contains correspond to the current content of the page (see 14.5, "Page-Piece Dictionaries").
Resources	dictionary	*(Required; inheritable)* A dictionary containing any resources required by the page (see 7.8.3, "Resource Dictionaries"). If the page requires no resources, the value of this entry shall be an empty dictionary. Omitting the entry entirely indicates that the resources shall be inherited from an ancestor node in the page tree.
MediaBox	rectangle	*(Required; inheritable)* A rectangle (see 7.9.5, "Rectangles"), expressed in default user space units, that shall define the boundaries of the physical medium on which the page shall be displayed or printed (see 14.11.2, "Page Boundaries").
CropBox	rectangle	*(Optional; inheritable)* A rectangle, expressed in default user space units, that shall define the visible region of default user space. When the page is displayed or printed, its contents shall be clipped (cropped) to this rectangle and then shall be imposed on the output medium in some implementation-defined manner (see 14.11.2, "Page Boundaries"). Default value: the value of **MediaBox**.
BleedBox	rectangle	*(Optional; PDF 1.3)* A rectangle, expressed in default user space units, that shall define the region to which the contents of the page shall be clipped when output in a production environment (see 14.11.2, "Page Boundaries"). Default value: the value of **CropBox**.
TrimBox	rectangle	*(Optional; PDF 1.3)* A rectangle, expressed in default user space units, that shall define the intended dimensions of the finished page after trimming (see 14.11.2, "Page Boundaries"). Default value: the value of **CropBox**.
ArtBox	rectangle	*(Optional; PDF 1.3)* A rectangle, expressed in default user space units, that shall define the extent of the page's meaningful content (including potential white space) as intended by the page's creator (see 14.11.2, "Page Boundaries"). Default value: the value of **CropBox**.
BoxColorInfo	dictionary	*(Optional; PDF 1.4)* A *box colour information dictionary* that shall specify the colours and other visual characteristics that should be used in displaying guidelines on the screen for the various page boundaries (see 14.11.2.2, "Display of Page Boundaries"). If this entry is absent, the application shall use its own current default settings.

Table 30 – Entries in a page object (continued)

Key	Type	Value
Contents	stream or array	*(Optional)* A *content stream* (see 7.8.2, "Content Streams") that shall describe the contents of this page. If this entry is absent, the page shall be empty. The value shall be either a single stream or an array of streams. If the value is an array, the effect shall be as if all of the streams in the array were concatenated, in order, to form a single stream. Conforming writers can create image objects and other resources as they occur, even though they interrupt the content stream. The division between streams may occur only at the boundaries between lexical tokens (see 7.2, "Lexical Conventions") but shall be unrelated to the page's logical content or organization. Applications that consume or produce PDF files need not preserve the existing structure of the **Contents** array. Conforming writers shall not create a Contents array containing no elements.
Rotate	integer	*(Optional; inheritable)* The number of degrees by which the page shall be rotated clockwise when displayed or printed. The value shall be a multiple of 90. Default value: 0.
Group	dictionary	*(Optional; PDF 1.4)* A *group attributes dictionary* that shall specify the attributes of the page's page group for use in the transparent imaging model (see 11.4.7, "Page Group" and 11.6.6, "Transparency Group XObjects").
Thumb	stream	*(Optional)* A stream object that shall define the page's *thumbnail image* (see 12.3.4, "Thumbnail Images").
B	array	*(Optional; PDF 1.1; recommended if the page contains article beads)* An array that shall contain indirect references to all *article beads* appearing on the page (see 12.4.3, "Articles"). The beads shall be listed in the array in natural reading order. NOTE The information in this entry can be created or recreated from the information obtained from the Threads key in the Catalog.
Dur	number	*(Optional; PDF 1.1)* The page's *display duration* (also called its *advance timing*): the maximum length of time, in seconds, that the page shall be displayed during presentations before the viewer application shall automatically advance to the next page (see 12.4.4, "Presentations"). By default, the viewer shall not advance automatically.
Trans	dictionary	*(Optional; PDF 1.1)* A *transition dictionary* describing the transition effect that shall be used when displaying the page during presentations (see 12.4.4, "Presentations").
Annots	array	*(Optional)* An array of *annotation dictionaries* that shall contain indirect references to all annotations associated with the page (see 12.5, "Annotations").
AA	dictionary	*(Optional; PDF 1.2)* An *additional-actions dictionary* that shall define actions to be performed when the page is opened or closed (see 12.6.3, "Trigger Events"). *(PDF 1.3)* additional-actions dictionaries are not inheritable.
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* that shall contain metadata for the page (see 14.3.2, "Metadata Streams").
PieceInfo	dictionary	*(Optional; PDF 1.3)* A *page-piece dictionary* associated with the page (see 14.5, "Page-Piece Dictionaries").

Table 30 – Entries in a page object (continued)

Key	Type	Value
StructParents	integer	*(Required if the page contains structural content items; PDF 1.3)* The integer key of the page's entry in the *structural parent tree* (see 14.7.4.4, "Finding Structure Elements from Content Items").
ID	byte string	*(Optional; PDF 1.3; indirect reference preferred)* The digital identifier of the page's parent *Web Capture content set* (see 14.10.6, "Object Attributes Related to Web Capture").
PZ	number	*(Optional; PDF 1.3)* The page's preferred *zoom (magnification) factor*: the factor by which it shall be scaled to achieve the natural display magnification (see 14.10.6, "Object Attributes Related to Web Capture").
SeparationInfo	dictionary	*(Optional; PDF 1.3)* A *separation dictionary* that shall contain information needed to generate colour separations for the page (see 14.11.4, "Separation Dictionaries").
Tabs	name	*(Optional; PDF 1.5)* A name specifying the tab order that shall be used for annotations on the page. The possible values shall be R (row order), C (column order), and S (structure order). See 12.5, "Annotations" for details.
TemplateInstantiated	name	*(Required if this page was created from a named page object; PDF 1.5)* The name of the originating page object (see 12.7.6, "Named Pages").
PresSteps	dictionary	*(Optional; PDF 1.5)* A *navigation node dictionary* that shall represent the first node on the page (see 12.4.4.2, "Sub-page Navigation").
UserUnit	number	*(Optional; PDF 1.6)* A positive number that shall give the size of default user space units, in multiples of 1⁄72 inch. The range of supported values shall be implementation-dependent. Default value: 1.0 (user space unit is 1⁄72 inch).
VP	dictionary	*(Optional; PDF 1.6)* An array of *viewport dictionaries* (see Table 260) that shall specify rectangular regions of the page.

EXAMPLE The following shows the definition of a page object with a thumbnail image and two annotations. The media box specifies that the page is to be printed on letter-size paper. In addition, the resource dictionary is specified as a direct object and shows that the page makes use of three fonts named F3, F5, and F7.

```
3  0  obj
   <<  /Type  /Page
       /Parent  4 0 R
       /MediaBox  [0  0  612  792]
       /Resources  <<  /Font   <<  /F3  7 0 R
                                   /F5  9 0 R
                                   /F7  11 0 R
                               >>
                        /ProcSet  [/PDF]
                   >>
       /Contents  12 0 R
       /Thumb  14 0 R
       /Annots   [  23 0 R
                    24 0 R
                 ]
   >>
endobj
```

7.7.3.4 Inheritance of Page Attributes

Some of the page attributes shown in Table 30 are designated as *inheritable*. If such an attribute is omitted from a page object, its value shall be inherited from an ancestor node in the page tree. If the attribute is a required one, a value shall be supplied in an ancestor node. If the attribute is optional and no inherited value is specified, the default value shall be used.

An attribute can thus be defined once for a whole set of pages by specifying it in an intermediate page tree node and arranging the pages that share the attribute as descendants of that node.

EXAMPLE A document may specify the same media box for all of its pages by including a **MediaBox** entry in the root node of the page tree. If necessary, an individual page object may override this inherited value with a **MediaBox** entry of its own.

In a document conforming to the Linearized PDF organization (see Annex F), all page attributes shall be specified explicitly as entries in the page dictionaries to which they apply; they shall not be inherited from an ancestor node.

Figure 6 illustrates the inheritance of attributes. In the page tree shown, pages 1, 2, and 4 are rotated clockwise by 90 degrees, page 3 by 270 degrees, page 6 by 180 degrees, and pages 5 and 7 not at all (0 degrees).

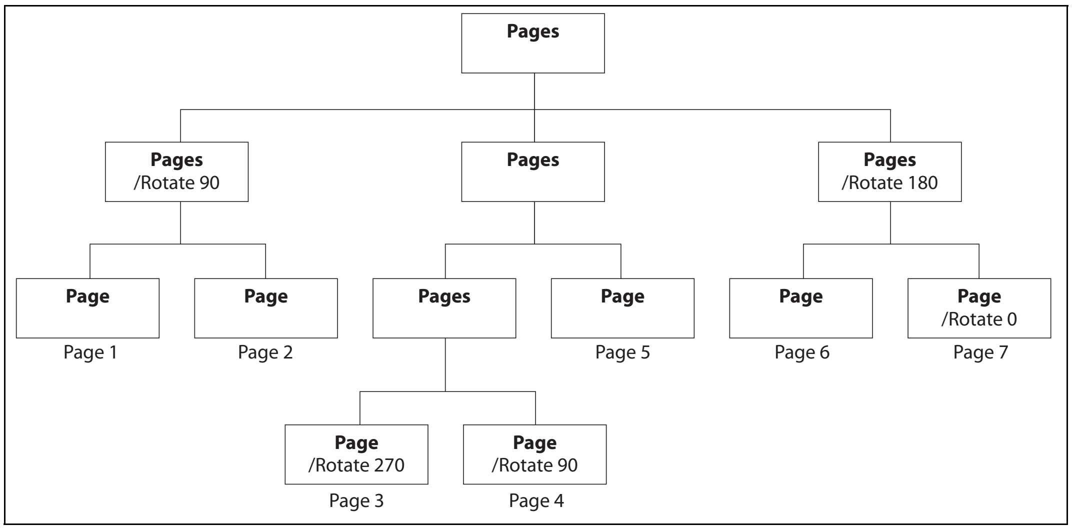


Figure 6 – Inheritance of attributes

7.7.4 Name Dictionary

Some categories of objects in a PDF file can be referred to by name rather than by object reference. The correspondence between names and objects is established by the document's *name dictionary (PDF 1.2)*, located by means of the **Names** entry in the document's catalog (see 7.7.2, "Document Catalog"). Each entry in this dictionary designates the root of a name tree (see 7.9.6, "Name Trees") defining names for a particular category of objects. Table 31 shows the contents of the name dictionary.

Table 31 – Entries in the name dictionary

Key	Type	Value
Dests	name tree	*(Optional; PDF 1.2)* A name tree mapping name strings to destinations (see 12.3.2.3, "Named Destinations").
AP	name tree	*(Optional; PDF 1.3)* A name tree mapping name strings to annotation appearance streams (see 12.5.5, "Appearance Streams").

Table 31 – Entries in the name dictionary (continued)

Key	Type	Value
JavaScript	name tree	*(Optional; PDF 1.3)* A name tree mapping name strings to document-level JavaScript actions (see 12.6.4.16, "JavaScript Actions").
Pages	name tree	*(Optional; PDF 1.3)* A name tree mapping name strings to visible pages for use in interactive forms (see 12.7.6, "Named Pages").
Templates	name tree	*(Optional; PDF 1.3)* A name tree mapping name strings to invisible (template) pages for use in interactive forms (see 12.7.6, "Named Pages").
IDS	name tree	*(Optional; PDF 1.3)* A name tree mapping digital identifiers to Web Capture content sets (see 14.10.4, "Content Sets").
URLS	name tree	*(Optional; PDF 1.3)* A name tree mapping uniform resource locators (URLs) to Web Capture content sets (see 14.10.4, "Content Sets").
EmbeddedFiles	name tree	*(Optional; PDF 1.4)* A name tree mapping name strings to file specifications for embedded file streams (see 7.11.4, "Embedded File Streams").
AlternatePresentations	name tree	*(Optional; PDF 1.4)* A name tree mapping name strings to alternate presentations (see 13.5, "Alternate Presentations").
Renditions	name tree	*(Optional; PDF 1.5)* A name tree mapping name strings (which shall have Unicode encoding) to rendition objects (see 13.2.3, "Renditions").

7.8 Content Streams and Resources

7.8.1 General

Content streams are the primary means for describing the appearance of pages and other graphical elements. A content stream depends on information contained in an associated resource dictionary; in combination, these two objects form a self-contained entity. This sub-clause describes these objects.

7.8.2 Content Streams

A *content stream* is a PDF stream object whose data consists of a sequence of instructions describing the graphical elements to be painted on a page. The instructions shall be represented in the form of PDF objects, using the same object syntax as in the rest of the PDF document. However, whereas the document as a whole is a static, random-access data structure, the objects in the content stream shall be interpreted and acted upon sequentially.

Each page of a document shall be represented by one or more content streams. Content streams shall also be used to package sequences of instructions as self-contained graphical elements, such as forms (see 8.10, "Form XObjects"), patterns (8.7, "Patterns"), certain fonts (9.6.5, "Type 3 Fonts"), and annotation appearances (12.5.5, "Appearance Streams").

A content stream, after decoding with any specified filters, shall be interpreted according to the PDF syntax rules described in 7.2, "Lexical Conventions." It consists of PDF objects denoting operands and operators. The operands needed by an operator shall precede it in the stream. See EXAMPLE 4 in 7.4, "Filters," for an example of a content stream.

An *operand* is a direct object belonging to any of the basic PDF data types except a stream. Dictionaries shall be permitted as operands only by certain specific operators. Indirect objects and object references shall not be permitted at all.

An *operator* is a PDF keyword specifying some action that shall be performed, such as painting a graphical shape on the page. An operator keyword shall be distinguished from a name object by the absence of an initial SOLIDUS character (2Fh) (/). Operators shall be meaningful only inside a content stream.

NOTE 1 This postfix notation, in which an operator is preceded by its operands, is superficially the same as in the PostScript language. However, PDF has no concept of an operand stack as PostScript has.

In PDF, all of the operands needed by an operator shall immediately precede that operator. Operators do not return results, and operands shall not be left over when an operator finishes execution.

NOTE 2 Most operators have to do with painting graphical elements on the page or with specifying parameters that affect subsequent painting operations. The individual operators are described in the clauses devoted to their functions:

Clause 8, "Graphics" describes operators that paint general graphics, such as filled areas, strokes, and sampled images, and that specify device-independent graphical parameters, such as colour.

Clause 9, "Text" describes operators that paint text using character glyphs defined in fonts.

Clause 10, "Rendering" describes operators that specify device-dependent rendering parameters.

Clause 14, "Document Interchange" describes the marked-content operators that associate higher-level logical information with objects in the content stream. These operators do not affect the rendered appearance of the content; they specify information useful to applications that use PDF for document interchange.

Ordinarily, when a conforming reader encounters an operator in a content stream that it does not recognize, an error shall occur. A pair of compatibility operators, **BX** and **EX** *(PDF 1.1)*, shall modify this behaviour (see Table 32). These operators shall occur in pairs and may be nested. They bracket a *compatibility section*, a portion of a content stream within which unrecognized operators shall be ignored without error. This mechanism enables a conforming writer to use operators defined in later versions of PDF without sacrificing compatibility with older applications. It should be used only in cases where ignoring such newer operators is the appropriate thing to do. The **BX** and **EX** operators are not themselves part of any graphics object (see 8.2, "Graphics Objects") or of the graphics state (8.4, "Graphics State").

Table 32 – Compatibility operators

Operands	Operator	Description
—	**BX**	*(PDF 1.1)* Begin a compatibility section. Unrecognized operators (along with their operands) shall be ignored without error until the balancing **EX** operator is encountered.
—	**EX**	*(PDF 1.1)* End a compatibility section begun by a balancing **BX** operator. Ignore any unrecognized operands and operators from previous matching **BX** onward.

7.8.3 Resource Dictionaries

As stated above, the operands supplied to operators in a content stream shall only be direct objects; indirect objects and object references shall not be permitted. In some cases, an operator shall refer to a PDF object that is defined outside the content stream, such as a font dictionary or a stream containing image data. This shall be accomplished by defining such objects as *named resources* and referring to them by name from within the content stream.

Named resources shall be meaningful only in the context of a content stream. The scope of a resource name shall be local to a particular content stream and shall be unrelated to externally known identifiers for objects such as fonts. References from one object outside of content streams to another outside of content streams shall be made by means of indirect object references rather than named resources.

A content stream’s named resources shall be defined by a *resource dictionary*, which shall enumerate the named resources needed by the operators in the content stream and the names by which they can be referred to.

EXAMPLE 1 If a text operator appearing within the content stream needs a certain font, the content stream’s resource dictionary can associate the name F42 with the corresponding font dictionary. The text operator can use this name to refer to the font.

A resource dictionary shall be associated with a content stream in one of the following ways:

- For a content stream that is the value of a page’s **Contents** entry (or is an element of an array that is the value of that entry), the resource dictionary shall be designated by the page dictionary’s **Resources** or is inherited, as described under 7.7.3.4, "Inheritance of Page Attributes," from some ancestor node of the page object.

- For other content streams, a conforming writer shall include a **Resources** entry in the stream's dictionary specifying the resource dictionary which contains all the resources used by that content stream. This shall apply to content streams that define form XObjects, patterns, Type 3 fonts, and annotation.

- PDF files written obeying earlier versions of PDF may have omitted the **Resources** entry in all form XObjects and Type 3 fonts used on a page. All resources that are referenced from those forms and fonts shall be inherited from the resource dictionary of the page on which they are used. This construct is obsolete and should not be used by conforming writers.

In the context of a given content stream, the term *current resource dictionary* refers to the resource dictionary associated with the stream in one of the ways described above.

Each key in a resource dictionary shall be the name of a resource type, as shown in Table 33. The corresponding values shall be as follows:

- For resource type **ProcSet**, the value shall be an array of procedure set names

- For all other resource types, the value shall be a subdictionary. Each key in the subdictionary shall be the name of a specific resource, and the corresponding value shall be a PDF object associated with the name.

Table 33 – Entries in a resource dictionary

Key	**Type**	**Value**
ExtGState	dictionary	*(Optional)* A dictionary that maps resource names to graphics state parameter dictionaries (see 8.4.5, "Graphics State Parameter Dictionaries").
ColorSpace	dictionary	*(Optional)* A dictionary that maps each resource name to either the name of a device-dependent colour space or an array describing a colour space (see 8.6, "Colour Spaces").
Pattern	dictionary	*(Optional)* A dictionary that maps resource names to pattern objects (see 8.7, "Patterns").
Shading	dictionary	*(Optional; PDF 1.3)* A dictionary that maps resource names to shading dictionaries (see 8.7.4.3, "Shading Dictionaries").
XObject	dictionary	*(Optional)* A dictionary that maps resource names to external objects (see 8.8, "External Objects").
Font	dictionary	*(Optional)* A dictionary that maps resource names to font dictionaries (see clause 9, "Text").
ProcSet	array	*(Optional)* An array of predefined procedure set names (see 14.2, "Procedure Sets").

Table 33 – Entries in a resource dictionary (continued)

Key	Type	Value
Properties	dictionary	*(Optional; PDF 1.2)* A dictionary that maps resource names to property list dictionaries for marked content (see 14.6.2, "Property Lists").

EXAMPLE 2 The following shows a resource dictionary containing procedure sets, fonts, and external objects. The procedure sets are specified by an array, as described in 14.2, "Procedure Sets". The fonts are specified with a subdictionary associating the names F5, F6, F7, and F8 with objects 6, 8, 10, and 12, respectively. Likewise, the **XObject** subdictionary associates the names Im1 and Im2 with objects 13 and 15, respectively.

```
<</ProcSet [/PDF /ImageB]
   /Font  << /F5  6 0 R
             /F6  8 0 R
             /F7  10 0 R
             /F8  12 0 R
          >>
   /XObject  << /Im1  13 0 R
                /Im2  15 0 R
             >>
>>
```

7.9 Common Data Structures

7.9.1 General

As mentioned at the beginning of this clause, there are some general-purpose data structures that are built from the basic object types described in 7.3, "Objects," and are used in many places throughout PDF. This sub-clause describes data structures for text strings, dates, rectangles, name trees, and number trees. More complex data structures are described in 7.10, "Functions," and 7.11, "File Specifications."

All of these data structures are meaningful only as part of the document hierarchy; they may not appear within content streams. In particular, the special conventions for interpreting the values of string objects apply only to strings outside content streams. An entirely different convention is used within content streams for using strings to select sequences of glyphs to be painted on the page (see clause 9, "Text"). Table 34 summarizes the basic and higher-level data types that are used throughout this standard to describe the values of dictionary entries and other PDF data values.

Table 34 – PDF data types

Type	Description	Sub-Clause
ASCII string	Bytes containing ASCII characters	7.9.2 7.9.2.4
array	Array object	7.3.6
boolean	Boolean value	7.3.2
byte string	A series of bytes that shall represent characters or other binary data. If such a type represents characters, the encoding shall be determined by the context.	7.9.2
date	Date (ASCII string)	7.9.4
dictionary	Dictionary object	7.3.7
file specification	File specification (string or dictionary)	7.11

Table 34 – PDF data types (continued)

Type	Description	Sub-Clause
function	Function (dictionary or stream)	7.10
integer	Integer number	
name	Name object	7.3.5
name tree	Name tree (dictionary)	7.9.6
null	Null object	7.3.9
number	Number (integer or real)	
number tree	Number tree (dictionary)	7.9.7
PDFDocEncoded string	Bytes containing a string that shall be encoded using PDFDocEncoding	7.9.2
rectangle	Rectangle (array)	7.9.5
stream	Stream object	7.3.8
string	Any string that is not a text string. Beginning with PDF 1.7, this type is further qualified as the types: PDFDocEncoded string, ASCII string, and byte string.	7.9.2
text string	Bytes that represent characters that shall be encoded using either PDFDocEncoding or UTF-16BE with a leading byte-order marker (as defined in “Text String Type” on page 86.)	7.9.2.2 7.9.2
text stream	Text stream	7.9.3

7.9.2 String Object Types

7.9.2.1 General

PDF supports one fundamental string object (see 7.3.4, "String Objects"). The string object shall be further qualified as a text string, a PDFDocEncoded string, ASCII string, or byte string. The further qualification reflects the encoding used to represent the characters or glyphs described by the string.

Table 35 summarizes the string object types that represent data encoded using specific conventions.

Table 35 – String Object Types

Type	Description
text string	Shall be used for human-readable text, such as text annotations, bookmark names, article names, and document information. These strings shall be encoded using either PDFDocEncoding or UTF-16BE with a leading byte-order marker. This type is described in 7.9.2.2, "Text String Type."
PDFDocEncoded string	Shall be used for characters and glyphs that are represented in a single byte, using PDFDocEncoding. This type is described in 7.9.2.3, "PDFDocEncoded String Type."

Table 35 – String Object Types (continued)

Type	Description
ASCII string	Shall be used for characters that are represented in a single byte using ASCII encoding.
byte string	Shall be used for binary data represented as a series of bytes, where each byte can be any value representable in 8 bits. The string may represent characters but the encoding is not known. The bytes of the string need not represent characters. This type shall be used for data such as MD5 hash values, signature certificates, and Web Capture identification values. This type is described in 7.9.2.4, "Byte String Type."

The string types described in Table 35 specify increasingly specific encoding schemes, as shown in Figure 7.

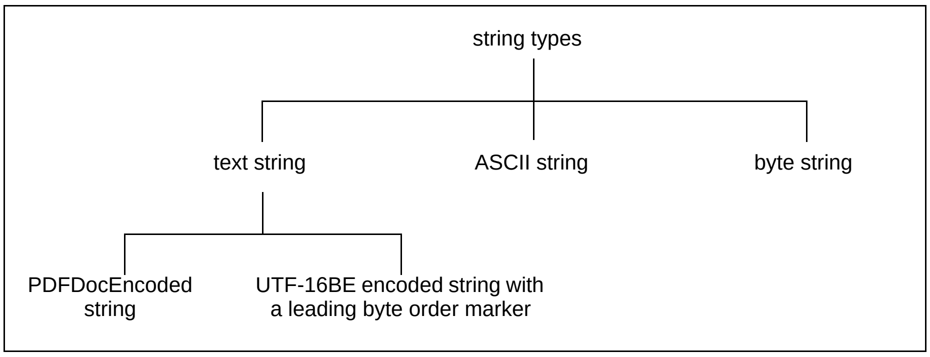


Figure 7 – Relationship between string types

7.9.2.2 Text String Type

The text string type shall be used for character strings that contain information intended to be human-readable, such as text annotations, bookmark names, article names, document information, and so forth.

NOTE 1 Text string type is a subtype of string type and represents data encoded using specific conventions.

The text string type shall be used for character strings that shall be encoded in either **PDFDocEncoding** or the UTF-16BE Unicode character encoding scheme. **PDFDocEncoding** can encode all of the ISO Latin 1 character set and is documented in Annex D. UTF-16BE can encode all Unicode characters. UTF-16BE and Unicode character encoding are described in the *Unicode Standard* by the Unicode Consortium (see the Bibliography).

NOTE 2 **PDFDocEncoding** does not support all Unicode characters whereas UTF-16BE does.

For text strings encoded in Unicode, the first two bytes shall be 254 followed by 255. These two bytes represent the Unicode byte order marker, U+FEFF, indicating that the string is encoded in the UTF-16BE (big-endian) encoding scheme specified in the Unicode standard.

NOTE 3 This mechanism precludes beginning a string using **PDFDocEncoding** with the two characters thorn ydieresis, which is unlikely to be a meaningful beginning of a word or phrase.

Conforming readers that process PDF files containing Unicode text strings shall be prepared to handle supplementary characters; that is, characters requiring more than two bytes to represent.

An escape sequence may appear anywhere in a Unicode text string to indicate the language in which subsequent text shall be written.

NOTE 4 This is useful when the language cannot be determined from the character codes used in the text.

The escape sequence shall consist of the following elements, in order:

a) The Unicode value U+001B (that is, the byte sequence 0 followed by 27).

b) A 2- byte ISO 639 language code.

EXAMPLE en for English or ja for Japanese encoded as ASCII characters.

c) *(Optional)* A 2-byte ISO 3166 country code.

EXAMPLE US for the United States or JP for Japan.

d) The Unicode value U+001B.

NOTE 5 The complete list of codes defined by ISO 639 and ISO 3166 can be obtained from the International Organization for Standardization (see the Bibliography).

7.9.2.3 PDFDocEncoded String Type

A PDFDocEncoded string is a character string in which the characters shall be represented in a single byte using PDFDocEncoding.

NOTE **PDFDocEncoding** does not support all Unicode characters whereas UTF-16BE does.

7.9.2.4 Byte String Type

The byte string type shall be used for binary data that shall be represented as a series of bytes, where each byte may be any value representable in 8 bits. Byte string type is a subtype of string type.

NOTE The string may represent characters but the encoding is not known. The bytes of the string may not represent characters.

7.9.3 Text Streams

A *text stream (PDF 1.5)* shall be a PDF stream object (7.3.8, "Stream Objects") whose unencoded bytes shall meet the same requirements as a text string (7.9.2.2, "Text String Type") with respect to encoding, byte order, and lead bytes.

7.9.4 Dates

Date values used in a PDF shall conform to a standard date format, which closely follows that of the international standard ASN.1 (Abstract Syntax Notation One), defined in ISO/IEC 8824. A date shall be a text string of the form

(D:*YYYYMMDDHHmmSSOHH*'*mm*)

where:

YYYY shall be the year

MM shall be the month (01–12)

DD shall be the day (01–31)

HH shall be the hour (00–23)

mm shall be the minute (00–59)

SS shall be the second (00–59)

O shall be the relationship of local time to Universal Time (UT), and shall be denoted by one of the characters PLUS SIGN (U+002B) (+), HYPHEN-MINUS (U+002D) (-), or LATIN CAPITAL LETTER Z (U+005A) (Z) (see below)

HH followed by APOSTROPHE (U+0027) (') shall be the absolute value of the offset from UT in hours (00–23)

mm shall be the absolute value of the offset from UT in minutes (00–59)

The prefix D: shall be present, the year field (YYYY) shall be present and all other fields may be present but only if all of their preceding fields are also present. The APOSTROPHE following the hour offset field (HH) shall only be present if the HH field is present. The minute offset field (mm) shall only be present if the APOSTROPHE following the hour offset field (HH) is present. The default values for *MM* and *DD* shall be both 01; all other numerical fields shall default to zero values. A PLUS SIGN as the value of the *O* field signifies that local time is later than UT, a HYPHEN-MINUS signifies that local time is earlier than UT, and the LATIN CAPITAL LETTER Z signifies that local time is equal to UT. If no UT information is specified, the relationship of the specified time to UT shall be considered to be GMT. Regardless of whether the time zone is specified, the rest of the date shall be specified in local time.

EXAMPLE For example, December 23, 1998, at 7:52 PM, U.S. Pacific Standard Time, is represented by the string `D:199812231952-08'00`

7.9.5 Rectangles

Rectangles are used to describe locations on a page and bounding boxes for a variety of objects. A rectangle shall be written as an array of four numbers giving the coordinates of a pair of diagonally opposite corners.

NOTE Although rectangles are conventionally specified by their lower-left and upper-right corners, it is acceptable to specify any two diagonally opposite corners. Applications that process PDF should be prepared to normalize such rectangles in situations where specific corners are required.

Typically, the array takes the form

$[ll_x \ ll_y \ ur_x \ ur_y]$

specifying the lower-left *x*, lower-left *y*, upper-right *x*, and upper-right *y* coordinates of the rectangle, in that order. The other two corners of the rectangle are then assumed to have coordinates (ll_x, ur_y) and (ur_x, ll_y).

7.9.6 Name Trees

A *name tree* serves a similar purpose to a dictionary—associating keys and values—but by different means. A name tree differs from a dictionary in the following important ways:

- Unlike the keys in a dictionary, which are name objects, those in a name tree are strings.
- The keys are ordered.
- The values associated with the keys may be objects of any type. Stream objects shall be specified by indirect object references (7.3.8, "Stream Objects"). The dictionary, array, and string objects should be specified by indirect object references, and other PDF objects (nulls, numbers, booleans, and names) should be specified as direct objects.
- The data structure can represent an arbitrarily large collection of key-value pairs, which can be looked up efficiently without requiring the entire data structure to be read from the PDF file. (In contrast, a dictionary can be subject to an implementation limit on the number of entries it can contain.)

A name tree shall be constructed of *nodes*, each of which shall be a dictionary object. Table 36 shows the entries in a node dictionary. The nodes shall be of three kinds, depending on the specific entries they contain. The tree shall always have exactly one *root node*, which shall contain a single entry: either **Kids** or **Names** but not both. If the root node has a **Names** entry, it shall be the only node in the tree. If it has a **Kids** entry, each of the remaining nodes shall be either an *intermediate node*, that shall contain a **Limits** entry and a **Kids** entry, or a *leaf node*, that shall contain a **Limits** entry and a **Names** entry.

Table 36 – Entries in a name tree node dictionary

Key	Type	Value
Kids	array	*(Root and intermediate nodes only; required in intermediate nodes; present in the root node if and only if* ***Names*** *is not present)* Shall be an array of indirect references to the immediate children of this node. The children may be intermediate or leaf nodes.
Names	array	*(Root and leaf nodes only; required in leaf nodes; present in the root node if and only if* ***Kids*** *is not present)* Shall be an array of the form [key_1 $value_1$ key_2 $value_2$ … key_n $value_n$] where each key_i shall be a string and the corresponding $value_i$ shall be the object associated with that key. The keys shall be sorted in lexical order, as described below.
Limits	array	*(Intermediate and leaf nodes only; required)* Shall be an array of two strings, that shall specify the (lexically) least and greatest keys included in the **Names** array of a leaf node or in the **Names** arrays of any leaf nodes that are descendants of an intermediate node.

The **Kids** entries in the root and intermediate nodes define the tree's structure by identifying the immediate children of each node. The **Names** entries in the leaf (or root) nodes shall contain the tree's keys and their associated values, arranged in key-value pairs and shall be sorted lexically in ascending order by key. Shorter keys shall appear before longer ones beginning with the same byte sequence. Any encoding of the keys may be used as long as it is self-consistent; keys shall be compared for equality on a simple byte-by-byte basis.

The keys contained within the various nodes' **Names** entries shall not overlap; each **Names** entry shall contain a single contiguous range of all the keys in the tree. In a leaf node, the **Limits** entry shall specify the least and greatest keys contained within the node's **Names** entry. In an intermediate node, it shall specify the least and greatest keys contained within the **Names** entries of any of that node's descendants. The value associated with a given key can thus be found by walking the tree in order, searching for the leaf node whose **Names** entry contains that key.

EXAMPLE 1 The following is an abbreviated outline, showing object numbers and nodes, of a name tree that maps the names of all the chemical elements, from actinium to zirconium, to their atomic numbers.

```
Example of a name tree
1:  Root node
        2:  Intermediate node: Actinium to Gold
                5:  Leaf node: Actinium = 25, …, Astatine = 31
                        25: Integer: 89
                        …
                        31: Integer: 85
                …
                11: Leaf node: Gadolinium = 56, …, Gold = 59
                        56: Integer: 64
                        …
                        59: Integer: 79
        3:  Intermediate node: Hafnium to Protactinium
                12: Leaf node: Hafnium = 60, …, Hydrogen = 65
                        60: Integer: 72
                        …
                        65: Integer: 1
                …
```

```
                    19: Leaf node: Palladium = 92, …, Protactinium = 100
                            92: Integer: 46
                            …
                            100:Integer: 91
            4:  Intermediate node: Radium to Zirconium
                    20: Leaf node: Radium = 101, …, Ruthenium = 107
                            101:Integer: 89
                            …
                            107:Integer: 85
                    …
                    24: Leaf node: Xenon = 129, …, Zirconium = 133
                            129:Integer: 54
                            …
                            133:Integer: 40
```

EXAMPLE 2 The following shows the representation of this tree in a PDF file

```
1  0  obj
    <<   /Kids   [   2 0 R                                    % Root node
                     3 0 R
                     4 0 R
                  ]
    >>
endobj

2  0  obj
    <<  /Limits  [(Actinium)  (Gold)]                         % Intermediate node
         /Kids   [  5 0 R
                    6 0 R
                    7 0 R
                    8 0 R
                    9 0 R
                    10 0 R
                    11 0 R
                 ]
    >>
endobj

3  0  obj
    <<  /Limits  [(Hafnium)  (Protactinium)]                  % Intermediate node
         /Kids   [  12 0 R
                    13 0 R
                    14 0 R
                    15 0 R
                    16 0 R
                    17 0 R
                    18 0 R
                    19 0 R
                 ]
    >>
endobj
4  0  obj
    <<  /Limits  [(Radium)  (Zirconium)]                      % Intermediate node
         /Kids   [  20 0 R
                    21 0 R
                    22 0 R
                    23 0 R
                    24 0 R
                 ]
    >>
endobj

5  0  obj
    <<  /Limits  [(Actinium)  (Astatine)]                     % Leaf node
         /Names  [   (Actinium)  25 0 R
```

```
                              (Aluminum)  26 0 R
                              (Americium)  27 0 R
                              (Antimony)  28 0 R
                              (Argon)  29 0 R
                              (Arsenic)  30 0 R
                              (Astatine)  31 0 R
                            ]
        >>
    endobj

    …

    24  0  obj
        << /Limits  [(Xenon)  (Zirconium)]                  % Leaf node
            /Names  [   (Xenon)  129 0 R
                              (Ytterbium)  130 0 R
                              (Yttrium)  131 0 R
                              (Zinc)  132 0 R
                              (Zirconium)  133 0 R
                            ]
        >>
    endobj

    25  0  obj
        89                                      % Atomic number (Actinium)
    endobj

    …

    133  0  obj
        40                                      % Atomic number (Zirconium)
    endobj
```

7.9.7 Number Trees

A *number tree* is similar to a name tree (see 7.9.6, "Name Trees"), except that its keys shall be integers instead of strings and shall be sorted in ascending numerical order. The entries in the leaf (or root) nodes containing the key-value pairs shall be named **Nums** instead of **Names** as in a name tree. Table 37 shows the entries in a number tree's node dictionaries.

Table 37 – Entries in a number tree node dictionary

Key	Type	Value
Kids	array	*(Root and intermediate nodes only; required in intermediate nodes; present in the root node if and only if* **Nums** *is not present)* Shall be an array of indirect references to the immediate children of this node. The children may be intermediate or leaf nodes.
Nums	array	*(Root and leaf nodes only; shall be required in leaf nodes; present in the root node if and only if* **Kids** *is not present)* Shall be an array of the form [key_1 $value_1$ key_2 $value_2$ … key_n $value_n$] where each key_i is an integer and the corresponding $value_i$ shall be the object associated with that key. The keys shall be sorted in numerical order, analogously to the arrangement of keys in a name tree as described in 7.9.6, "Name Trees."
Limits	array	*(Shall be present in Intermediate and leaf nodes only)* Shall be an array of two integers, that shall specify the (numerically) least and greatest keys included in the **Nums** array of a leaf node or in the **Nums** arrays of any leaf nodes that are descendants of an intermediate node.

7.10 Functions

7.10.1 General

PDF is not a programming language, and a PDF file is not a program. However, PDF provides several types of *function objects (PDF 1.2)* that represent parameterized classes of functions, including mathematical formulas and sampled representations with arbitrary resolution.

NOTE 1 Functions may be used in various ways in PDF, including device-dependent rasterization information for high-quality printing (halftone spot functions and transfer functions), colour transform functions for certain colour spaces, and specification of colours as a function of position for smooth shadings.

Functions in PDF represent static, self-contained numerical transformations.

NOTE 2 A function to add two numbers has two input values and one output value:

$$f(x_0, x_1) = x_0 + x_1$$

Similarly, a function that computes the arithmetic and geometric mean of two numbers can be viewed as a function of two input values and two output values:

$$f(x_0, x_1) = \frac{x_0 + x_1}{2}, \sqrt{x_0 \times x_1}$$

In general, a function can take any number *(m)* of input values and produce any number *(n)* of output values:

$$f(x_0, \ldots, x_{m-1}) = y_0, \ldots, y_{n-1}$$

In PDF functions, all the input values and all the output values shall be numbers, and functions shall have no side effects.

Each function definition includes a *domain*, the set of legal values for the input. Some types of functions also define a *range*, the set of legal values for the output. Input values passed to the function shall be clipped to the domain, and output values produced by the function shall be clipped to the range.

EXAMPLE Suppose the following function is defined with a domain of [-1 1]. If the function is called with the input value 6, that value is replaced with the nearest value in the defined domain, 1, before the function is evaluated; the resulting output value is therefore 3.

$$f(x) = x + 2$$

Similarly, if the following function is defined with a range of [0 100], and if the input values -6 and 4 are passed to the function (and are within its domain), then the output value produced by the function, -14, is replaced with 0, the nearest value in the defined range.

$$f(x_0, x_1) = 3 \times x_0 + x_1$$

A function object may be a dictionary or a stream, depending on the type of function. The term *function dictionary* is used generically in this sub-clause to refer to either a dictionary object or the dictionary portion of a stream object. A function dictionary specifies the function's representation, the set of attributes that parameterize that representation, and the additional data needed by that representation. Four types of functions are available, as indicated by the dictionary's **FunctionType** entry:

- *(PDF 1.2)* A *sampled function* (type 0) uses a table of *sample values* to define the function. Various techniques are used to interpolate values between the sample values; see 7.10.2, "Type 0 (Sampled) Functions."

- *(PDF 1.3)* An *exponential interpolation function* (type 2) defines a set of coefficients for an exponential function; see 7.10.3, "Type 2 (Exponential Interpolation) Functions."
- *(PDF 1.3)* A *stitching function* (type 3) is a combination of other functions, partitioned across a domain; see 7.10.4, "Type 3 (Stitching) Functions."
- *(PDF 1.3)* A *PostScript calculator function* (type 4) uses operators from the PostScript language to describe an arithmetic expression; see 7.10.5, "Type 4 (PostScript Calculator) Functions."

All function dictionaries shall share the entries listed in Table 38.

Table 38 – Entries common to all function dictionaries

Key	Type	Value
FunctionType	integer	*(Required)* The function type: 0 Sampled function 2 Exponential interpolation function 3 Stitching function 4 PostScript calculator function
Domain	array	*(Required)* An array of $2 \times m$ numbers, where m shall be the number of input values. For each i from 0 to $m - 1$, $\mathbf{Domain}_{2i}$ shall be less than or equal to $\mathbf{Domain}_{2i+1}$, and the ith input value, x_i, shall lie in the interval $\mathbf{Domain}_{2i} \le x_i \le \mathbf{Domain}_{2i+1}$. Input values outside the declared domain shall be clipped to the nearest boundary value.
Range	array	*(Required for type 0 and type 4 functions, optional otherwise; see below)* An array of $2 \times n$ numbers, where n shall be the number of output values. For each j from 0 to $n - 1$, $\mathbf{Range}_{2j}$ shall be less than or equal to $\mathbf{Range}_{2j+1}$, and the jth output value, y_j, shall lie in the interval $\mathbf{Range}_{2j} \le y_j \le \mathbf{Range}_{2j+1}$. Output values outside the declared range shall be clipped to the nearest boundary value. If this entry is absent, no clipping shall be done.

In addition, each type of function dictionary shall include entries appropriate to the particular function type. The number of output values can usually be inferred from other attributes of the function; if not (as is always the case for type 0 and type 4 functions), the **Range** entry is required. The dimensionality of the function implied by the **Domain** and **Range** entries shall be consistent with that implied by other attributes of the function.

7.10.2 Type 0 (Sampled) Functions

Type 0 functions use a sequence of *sample values* (contained in a stream) to provide an approximation for functions whose domains and ranges are bounded. The samples are organized as an *m*-dimensional table in which each entry has *n* components.

NOTE 1 Sampled functions are highly general and offer reasonably accurate representations of arbitrary analytic functions at low expense. For example, a 1-input sinusoidal function can be represented over the range [0 180] with an average error of only 1 percent, using just ten samples and linear interpolation. Two-input functions require significantly more samples but usually not a prohibitive number if the function does not have high frequency variations.

There shall be no dimensionality limit of a sampled function except for possible implementation limits.

NOTE 2 The number of samples required to represent functions with high dimensionality multiplies rapidly unless the sampling resolution is very low. Also, the process of multilinear interpolation becomes computationally intensive if the number of inputs *m* is greater than 2. The multidimensional spline interpolation is even more computationally intensive.

In addition to the entries in Table 38, a type 0 function dictionary includes those shown in Table 39.

The **Domain**, **Encode**, and **Size** entries determine how the function’s input variable values are mapped into the sample table. For example, if **Size** is [21 31], the default **Encode** array shall be [0 20 0 30], which maps the entire domain into the full set of sample table entries. Other values of **Encode** may be used.

To explain the relationship between **Domain**, **Encode**, **Size**, **Decode**, and **Range**, we use the following notation:

$$\begin{aligned} y &= \text{Interpolate}(x, x_{\min}, x_{\max}, y_{\min}, y_{\max}) \\ &= y_{\min} + \left((x - x_{\min}) \times \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \right) \end{aligned}$$

For a given value of *x*, Interpolate calculates the *y* value on the line defined by the two points ($x_{\min}$, $y_{\min}$) and ($x_{\max}$, $y_{\max}$).

Table 39 – Additional entries specific to a type 0 function dictionary

Key	Type	Value
Size	array	*(Required)* An array of *m* positive integers that shall specify the number of samples in each input dimension of the sample table.
BitsPerSample	integer	*(Required)* The number of bits that shall represent each sample. (If the function has multiple output values, each one shall occupy **BitsPerSample** bits.) Valid values shall be 1, 2, 4, 8, 12, 16, 24, and 32.
Order	integer	*(Optional)* The order of interpolation between samples. Valid values shall be 1 and 3, specifying linear and cubic spline interpolation, respectively. Default value: 1.
Encode	array	*(Optional)* An array of 2 × *m* numbers specifying the linear mapping of input values into the domain of the function’s sample table. Default value: [0 (**Size**0 – 1) 0 (**Size**1 – 1) …].
Decode	array	*(Optional)* An array of 2 × *n* numbers specifying the linear mapping of sample values into the range appropriate for the function’s output values. Default value: same as the value of **Range**.
other stream attributes	(various)	*(Optional)* Other attributes of the stream that shall provide the sample values, as appropriate (see Table 5).

When a sampled function is called, each input value x_i, for 0 £ *i* < *m*, shall be clipped to the domain:

$$x_i' = \min(\max(x_i, \text{Domain}_{2i}), \text{Domain}_{2i+1})$$

That value shall be encoded:

$$e_i = \text{Interpolate}(x_i', \text{Domain}_{2i}, \text{Domain}_{2i+1}, \text{Encode}_{2i}, \text{Encode}_{2i+1})$$

That value shall be clipped to the size of the sample table in that dimension:

$$e_i' = \min(\max(e_i, 0), \text{Size}_i - 1)$$

The encoded input values shall be real numbers, not restricted to integers. Interpolation shall be used to determine output values from the nearest surrounding values in the sample table. Each output value r_j, for 0 £ *j* < *n*, shall then be decoded:

$$r_j' = \text{Interpolate}(r_j, 0, 2^{\text{BitsPerSample}} - 1, \text{Decode}_{2j}, \text{Decode}_{2j+1})$$

Finally, each decoded value shall be clipped to the range:

$$y_j = \min(\max(r_j', \text{Range}_{2j}), \text{Range}_{2j+1})$$

Sample data shall be represented as a stream of bytes. The bytes shall constitute a continuous bit stream, with the high-order bit of each byte first. Each sample value shall be represented as a sequence of **BitsPerSample** bits. Successive values shall be adjacent in the bit stream; there shall be no padding at byte boundaries.

For a function with multidimensional input (more than one input variable), the sample values in the first dimension vary fastest, and the values in the last dimension vary slowest.

EXAMPLE 1 For a function *f*(*a*, *b*, *c*), where *a*, *b*, and *c* vary from 0 to 9 in steps of 1, the sample values would appear in this order: *f*(0, 0, 0), *f*(1, 0, 0), …, *f*(9, 0, 0), *f*(0, 1, 0), *f*(1, 1, 0), …, *f*(9, 1, 0), *f*(0, 2, 0), *f*(1, 2, 0), …, *f*(9, 9, 0), *f*(0, 0, 1), *f*(1, 0, 1), and so on.

For a function with multidimensional output (more than one output value), the values shall be stored in the same order as **Range**.

The stream data shall be long enough to contain the entire sample array, as indicated by **Size**, **Range**, and **BitsPerSample**; see 7.3.8.2, "Stream Extent."

Example 2 illustrates a sampled function with 4-bit samples in an array containing 21 columns and 31 rows (651 values). The function takes two arguments, *x* and *y*, in the domain [-1.0 1.0], and returns one value, *z*, in that same range. The *x* argument shall 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 (because **BitsPerSample** is 4) will be in the range [0 15], and the decoding transforms *z* to a number in the range [-1.0 1.0] for the result. The sample array shall be stored in a string of 326 bytes, calculated as follows (rounded up):

326 bytes = 31 rows ¥ 21 samples/row ¥ 4 bits/sample ³ 8 bits/byte

The first byte contains the sample for the point (-1.0, -1.0) in the high-order 4 bits and the sample for the point (-0.9, -1.0) in the low-order 4 bits.

EXAMPLE 2

```
14 0 obj
   << /FunctionType 0
        /Domain [-1.0 1.0  -1.0 1.0]
        /Size [21 31]
        /Encode [0 20  0 30]
        /BitsPerSample 4
        /Range [-1.0 1.0]
        /Decode [-1.0 1.0]
        /Length …
        /Filter …
   >>
stream
…651 sample values…
endstream
endobj
```

NOTE 3 The **Decode** entry can be used creatively to increase the accuracy of encoded samples corresponding to certain values in the range.

EXAMPLE 3 If the range of the function is [-1.0 1.0] and **BitsPerSample** is 4, the usual value of **Decode** would be [-1.0 1.0] and the sample values would be integers in the interval [0 15] (as shown in Figure 8). But if these values are used, the midpoint of the range, 0.0, is not represented exactly by any sample value, since it falls halfway between 7 and 8. However, if the **Decode** array is [-1.0 +1.1429] (1.1429 being approximately equal to 16 ³ 14) and the sample values supplied are in the interval [0 14], the effective range of [-1.0 1.0] is achieved, and the range value 0.0 is represented by the sample value 7.

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

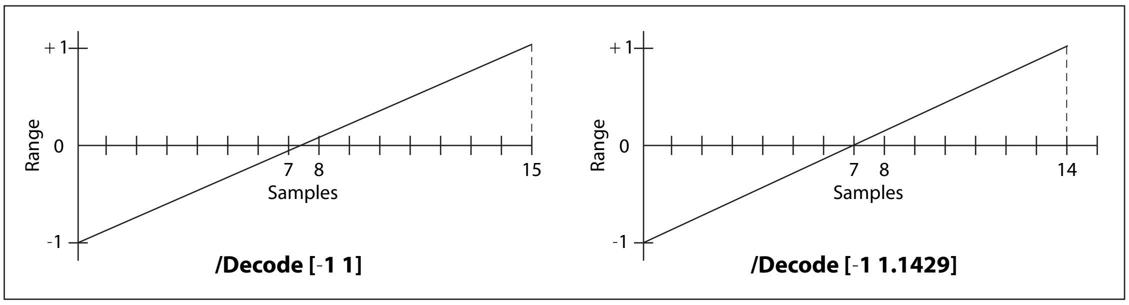


Figure 8 – Mapping with the Decode array

7.10.3 Type 2 (Exponential Interpolation) Functions

Type 2 functions *(PDF 1.3)* include a set of parameters that define an *exponential interpolation* of one input value and *n* output values:

$$f(x) = y_0, \ldots, y_{n-1}$$

In addition to the entries in Table 38, a type 2 function dictionary shall include those listed in Table 40.

Table 40 – Additional entries specific to a type 2 function dictionary

Key	Type	Value
C0	array	*(Optional)* An array of *n* numbers that shall define the function result when $x = 0.0$. Default value: [0.0].
C1	array	*(Optional)* An array of *n* numbers that shall define the function result when $x = 1.0$. Default value: [1.0].
N	number	*(Required)* The interpolation exponent. Each input value *x* shall return *n* values, given by $y_j = \mathbf{C0}_j + x^{\mathrm{N}} \times (\mathbf{C1}_j - \mathbf{C0}_j)$, for $0 \le j < n$.

Values of **Domain** shall constrain *x* in such a way that if **N** is not an integer, all values of *x* shall be non-negative, and if **N** is negative, no value of *x* shall be zero. Typically, **Domain** is declared as [0.0 1.0], and **N** is a positive number. To clip the output to a specified range The **Range** attribute shall be used.

NOTE When **N** is 1, the function performs a linear interpolation between **C0** and **C1**; therefore, the function can also be expressed as a sampled function (type 0).

7.10.4 Type 3 (Stitching) Functions

Type 3 functions *(PDF 1.3)* define a *stitching* of the subdomains of several 1-input functions to produce a single new 1-input function. Since the resulting stitching function is a 1-input function, the domain is given by a two-element array, [$\mathbf{Domain}_0$ $\mathbf{Domain}_1$].

In addition to the entries in Table 38, a type 3 function dictionary shall include those listed in Table 41.

Table 41 – Additional entries specific to a type 3 function dictionary

Key	Type	Value
Functions	array	*(Required)* An array of *k* 1-input functions that shall make up the stitching function. The output dimensionality of all functions shall be the same, and compatible with the value of **Range** if **Range** is present.
Bounds	array	*(Required)* An array of $k - 1$ numbers that, in combination with **Domain**, shall define the intervals to which each function from the **Functions** array shall apply. **Bounds** elements shall be in order of increasing value, and each value shall be within the domain defined by **Domain**.
Encode	array	*(Required)* An array of $2 \times k$ numbers that, taken in pairs, shall map each subset of the domain defined by **Domain** and the **Bounds** array to the domain of the corresponding function.

Domain shall be of size 2 (that is, $m = 1$), and $\mathbf{Domain}_0$ shall be strictly less than $\mathbf{Domain}_1$ unless $k = 1$. The domain shall be partitioned into *k* subdomains, as indicated by the dictionary's **Bounds** entry, which shall be an array of *k* - 1 numbers that obey the following relationships (with exceptions as noted below):

$$\text{Domain}_0 < \text{Bounds}_0 < \text{Bounds}_1 < \ldots < \text{Bounds}_{k-2} < \text{Domain}_1$$

The **Bounds** array shall describe a series of half-open intervals, closed on the left and open on the right (except the last, which is closed on the right as well). The value of the **Functions** entry shall be an array of *k* functions. The first function shall apply to *x* values in the first subdomain, $\mathbf{Domain}_0$ £ $x < \mathbf{Bounds}_0$; the second function shall apply to *x* values in the second subdomain, $\mathbf{Bounds}_0$ £ $x < \mathbf{Bounds}_1$; and so on. The last function shall apply to *x* values in the last subdomain, which includes the upper bound:
$\mathbf{Bounds}_{k\text{-}2}$ £ x £ $\mathbf{Domain}_1$. The value of *k* may be 1, in which case the **Bounds** array shall be empty and the single item in the **Functions** array shall apply to all *x* values, $\mathbf{Domain}_0$ £ x £ $\mathbf{Domain}_1$.

The **Encode** array contains 2 ¥ *k* numbers. A value *x* from the *i*th subdomain shall be encoded as follows:

$$x' = \text{Interpolate}\,(x, \text{Bounds}_{i-1}, \text{Bounds}_i, \text{Encode}_{2i}, \text{Encode}_{2i+1})$$

for 0 £ $i < k$. In this equation, $\mathbf{Bounds}_{-1}$ means $\mathbf{Domain}_0$, and $\mathbf{Bounds}_{k\text{-}1}$ means $\mathbf{Domain}_1$. If the last bound, $\mathbf{Bounds}_{k\text{-}2}$, is equal to $\mathbf{Domain}_1$, then *x*¢ shall be defined to be $\mathbf{Encode}_{2i}$.

NOTE The stitching function is designed to make it easy to combine several functions to be used within one shading pattern over different parts of the shading's domain. Shading patterns are discussed in 8.7.4, "Shading Patterns". The same effect could be achieved by creating a separate shading dictionary for each of the functions, with 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 multiple function definitions. Also, type 3 functions provide a general mechanism for inverting the domains of 1-input functions.

EXAMPLE Consider a function *f* with a **Domain** of [0.0 1.0] and a stitching function *g* with a **Domain** of [0.0 1.0], a **Functions** array containing *f*, and an **Encode** array of [1.0 0.0]. In effect, $g(x) = f(1 - x)$.

7.10.5 Type 4 (PostScript Calculator) Functions

7.10.5.1 General

A type 4 function *(PDF 1.3)*, also called a PostScript calculator function, shall be represented as a stream containing code written in a small subset of the PostScript language.

NOTE Although any function can be sampled (in a type 0 PDF function) and others can be described with exponential functions (type 2 in PDF), type 4 functions offer greater flexibility and potentially greater accuracy. For example, a tint transformation function for a hexachrome (six-component) **DeviceN** colour space with an alternate colour space of **DeviceCMYK** (see 8.6.6.5, "DeviceN Colour Spaces") requires a 6-in, 4-out function. If such a function were sampled with *m* values for each input variable, the number of samples, 4 ¥ m^6, could be prohibitively large. In practice, such functions can often be written as short, simple PostScript functions.

Type 4 functions also make it possible to include a wide variety of halftone spot functions without the loss of accuracy that comes from sampling, and without adding to the list of predefined spot functions (see 10.5.3, "Spot Functions"). All of the predefined spot functions can be written as type 4 functions.

The language that shall be used in a type 4 function contains expressions involving integers, real numbers, and boolean values only. There shall be no composite data structures such as strings or arrays, no procedures, and no variables or names. Table 42 lists the operators that can be used in this type of function. (For more information on these operators, see Appendix B of the *PostScript Language Reference*, Third Edition.) Although the semantics are those of the corresponding PostScript operators, a full PostScript interpreter is not required.

Table 42 – Operators in type 4 functions

Operator Type	Operators				
Arithmetic operators	**abs** **add** **atan** **ceiling** **cos**	**cvi** **cvr** **div** **exp**	**floor** **idiv** **ln** **log**	**mod** **mul** **neg** **round**	**sin** **sqrt** **sub** **truncate**
Relational, boolean, and bitwise operators	**and** **bitshift** **eq**	**false** **ge** **gt**	**le** **lt** **ne**	**not** **or**	**true** **xor**
Conditional operators	**if**	**ifelse**			
Stack operators	**copy** **dup**	**exch** **index**	**pop** **roll**		

The operand syntax for type 4 functions shall follow PDF conventions rather than PostScript conventions. The entire code stream defining the function shall be enclosed in braces { } (using LEFT CURLY BRACE (7Bh) and RIGHT CURLY BRACE (07hD)). Braces also shall delimit expressions that are executed conditionally by the **if** and **ifelse** operators:

- *boolean* {*expression*} if
- *boolean* {*expression*$_1$} {*expression*$_2$} ifelse

This construct is purely syntactic; unlike in PostScript, no “procedure objects” shall be involved.

A type 4 function dictionary shall include the entries in Table 38, as well as other appropriate stream attributes (see Table 5). The following example shows a type 4 function equivalent to the predefined spot function **DoubleDot** (see 10.5.3, "Spot Functions").

EXAMPLE
```
10 0 obj
   << /FunctionType 4
       /Domain [-1.0 1.0  -1.0 1.0]
       /Range [-1.0  1.0]
       /Length 71
   >>
stream
```

```
        { 360  mul  sin
            2  div
            exch  360  mul  sin
            2  div
            add
        }
    endstream
    endobj
```

The **Domain** and **Range** entries shall both be required. The input variables shall constitute the initial operand stack; the items remaining on the operand stack after execution of the function shall be the output variables. It shall be an error for the number of remaining operands to differ from the number of output variables specified by **Range** or for any of them to be objects other than numbers.

Implementations of type 4 functions shall provide a stack with room for at least 100 entries. No implementation shall be required to provide a larger stack, and it shall be an error to overflow the stack.

Although any integers or real numbers that may appear in the stream fall under the same implementation limits (defined in Annex C) as in other contexts, the *intermediate* results in type 4 function computations shall not. An implementation may use a representation that exceeds those limits. Operations on real numbers, for example, might use single-precision or double-precision floating-point numbers.

7.10.5.2 Errors in Type 4 Functions

The part of a conforming reader that reads a type 4 function (analogous to the PostScript *scanner*) shall detect and report syntax errors. Any errors detected by the conforming reader shall be errors in the PDF file and shall be handled like other errors in the file.

The part of a conforming reader that executes a type 4 function (analogous to the PostScript *interpreter*) shall detect and report errors. This specification does not define a representation for the errors; those details shall be provided by the conforming reader that processes the PDF file. The following types of errors can occur (among others):

- Stack overflow
- Stack underflow
- A type error (for example, applying **not** to a real number)
- A range error (for example, applying **sqrt** to a negative number)
- An undefined result (for example, dividing by 0)

7.11 File Specifications

7.11.1 General

A PDF file can refer to the contents of another file by using a *file specification (PDF 1.1)*, which shall take either of two forms:

- A *simple* file specification shall give just the name of the target file in a standard format, independent of the naming conventions of any particular file system. It shall take the form of either a string or a dictionary
- A *full* file specification shall include information related to one or more specific file systems. It shall only be represented as a dictionary.

A file specification shall refer to a file external to the PDF file or to a file embedded within the referring PDF file, allowing its contents to be stored or transmitted along with the PDF file. The file shall be considered to be external to the PDF file in either case.

7.11.2 File Specification Strings

7.11.2.1 General

The standard format for representing a simple file specification in string form divides the string into component substrings separated by the SOLIDUS character (2Fh) (/). The SOLIDUS is a generic component separator that shall be mapped to the appropriate platform-specific separator when generating a platform-dependent file name. Any of the components may be empty. If a component contains one or more literal SOLIDI, each shall be preceded by a REVERSE SOLIDUS (5Ch) (\), which in turn shall be preceded by another REVERSE SOLIDUS to indicate that it is part of the string and not an escape character.

EXAMPLE (in\\/out)
represents the file name
in/out

The REVERSE SOLIDI shall be removed in processing the string; they are needed only to distinguish the component values from the component separators. The component substrings shall be stored as bytes and shall be passed to the operating system without interpretation or conversion of any sort.

7.11.2.2 Absolute and Relative File Specifications

A simple file specification that begins with a SOLIDUS shall be an *absolute* file specification. The last component shall be the file name; the preceding components shall specify its context. In some file specifications, the file name may be empty; for example, URL (uniform resource locator) specifications can specify directories instead of files. A file specification that does not begin with a SOLIDUS shall be a *relative* file specification giving the location of the file relative to that of the PDF file containing it.

In the case of a URL-based file system, the rules of Internet RFC 1808, *Relative Uniform Resource Locators* (see the Bibliography), shall be used to compute an absolute URL from a relative file specification and the specification of the PDF file. Prior to this process, the relative file specification shall be converted to a relative URL by using the escape mechanism of RFC 1738, *Uniform Resource Locators*, to represent any bytes that would be either unsafe according to RFC 1738 or not representable in 7-bit U.S. ASCII. In addition, such URL-based relative file specifications shall be limited to paths as defined in RFC 1808. The scheme, network location/login, fragment identifier, query information, and parameter sections shall not be allowed.

In the case of other file systems, a relative file specification shall be converted to an absolute file specification by removing the file name component from the specification of the containing PDF file and appending the relative file specification in its place.

EXAMPLE 1 The relative file specification

ArtFiles/Figure1.pdf

appearing in a PDF file whose specification is

/HardDisk/PDFDocuments/AnnualReport/Summary.pdf

yields the absolute specification

/HardDisk/PDFDocuments/AnnualReport/ArtFiles/Figure1.pdf

The special component .. (two PERIODs) (2Eh) can be used in a relative file specification to move up a level in the file system hierarchy. After an absolute specification has been derived, when the component immediately preceding .. is not another .., the two cancel each other; both are eliminated from the file specification and the process is repeated.

EXAMPLE 2 The relative file specification from EXAMPLE 1 in this sub-clause using the .. (two PERIODs) special component

../../ArtFiles/Figure1.pdf

would yield the absolute specification

/HardDisk/ArtFiles/Figure1.pdf

7.11.2.3 Conversion to Platform-Dependent File Names

The conversion of a file specification to a platform-dependent file name depends on the specific file naming conventions of each platform:

- For DOS, the initial component shall be either a physical or logical drive identifier or a network resource name as returned by the Microsoft Windows function WNetGetConnection, and shall be followed by a COLON (3Ah) (:). A network resource name shall be constructed from the first two components; the first component shall be the server name and the second shall be the share name (volume name). All components shall be separated by REVERSE SOLIDI (backslashes) (5Ch). It shall be possible to specify an absolute DOS path without a drive by making the first component empty. (Empty components are ignored by other platforms.)
- For Mac OS, all components shall be separated by COLONs.
- For UNIX, all components shall be separated by SOLIDI (2Fh) (slashes). An initial SOLIDUS, if present, shall be preserved.

Strings used to specify a file name shall be interpreted in the standard encoding for the platform on which the document is being viewed. Table 43 shows examples of file specifications on the most common platforms.

Table 43 – Examples of file specifications

System	System-dependent paths	Written form
DOS	\pdfdocs\spec.pdf (no drive) r:\pdfdocs\spec.pdf pclib/eng:\pdfdocs\spec.pdf	(//pdfdocs/spec.pdf) (/r/pdfdocs/spec.pdf) (/pclib/eng/pdfdocs/spec.pdf)
Mac OS	Mac HD:PDFDocs:spec.pdf	(/Mac HD/PDFDocs/spec.pdf)
UNIX	/user/fred/pdfdocs/spec.pdf pdfdocs/spec.pdf (relative)	(/user/fred/pdfdocs/spec.pdf) (pdfdocs/spec.pdf)

NOTE 1 When creating documents that are to be viewed on multiple platforms, care should be taken to ensure file name compatibility. Only a subset of the U.S. ASCII character set should be used in file specifications: the uppercase alphabetic characters (A–Z), the numeric characters (0–9), the PERIOD (2Eh) and the LOW LINE (underscore) (5Fh). The PERIOD has special meaning in DOS and Windows file names, and as the first character in a Mac OS pathname. In file specifications, the PERIOD should be used only to separate a base file name from a file extension.

NOTE 2 Some file systems are case-insensitive, and names within a directory are unique so names should remain distinguishable if lowercase letters are changed to uppercase or vice versa. On DOS and Windows 3.1 systems and on some CD-ROM file systems, file names are limited to 8 characters plus a 3-character extension. File system software typically converts long names to short names by retaining the first 6 or 7 characters of the file name and the first 3 characters after the last PERIOD, if any. Since characters beyond the sixth or seventh are often converted to other values unrelated to the original value, file names should be distinguishable from the first 6 characters.

7.11.2.4 Multiple-Byte Strings in File Specifications

In PDF 1.2 or higher, a file specification may contain multiple-byte character codes, represented in hexadecimal form between angle brackets (< and >) (using LESS-THAN SIGN (3Ch) and GREATER-THAN SIGN (3Eh)). Since the SOLIDUS (2Fh) (slash character), denoted as <2F>, is used as a component delimiter and the REVERSE SOLIDUS (5Ch) (backslash character), denoted as <5C>, is used as an escape character, any occurrence of either of these bytes in a multiple-byte character shall be preceded by the ASCII code for the SOLIDUS (2Fh).

EXAMPLE A file name containing the 2-byte character code <89 5C> is written as <89 5C 5C>. When the application encounters this sequence of bytes in a file name, it replaces the sequence with the original 2-byte code.

7.11.3 File Specification Dictionaries

The dictionary form of file specification provides more flexibility than the string form, allowing different files to be specified for different file systems or platforms, or for file systems other than the standard ones (DOS/Windows, Mac OS, and UNIX). Table 44 shows the entries in a file specification dictionary. Regardless of the platform, conforming readers should use the **F** and **UF** (beginning with PDF 1.7) entries to specify files. The **UF** entry is optional, but should be included because it enables cross-platform and cross-language compatibility.

Table 44 – Entries in a file specification dictionary

Key	Type	Value
Type	name	*(Required if an* **EF** *or* **RF** *entry is present; recommended always)* The type of PDF object that this dictionary describes; shall be **Filespec** for a file specification dictionary.
FS	name	*(Optional)* The name of the file system that shall be used to interpret this file specification. If this entry is present, all other entries in the dictionary shall be interpreted by the designated file system. PDF shall define only one standard file system name, **URL** (see 7.11.5, "URL Specifications"); an application can register other names (see Annex E). This entry shall be independent of the **F**, **UF**, **DOS**, **Mac**, and **Unix** entries.
F	string	*(Required if the* ***DOS****,* ***Mac****, and* ***Unix*** *entries are all absent; amended with the* ***UF*** *entry for PDF 1.7)* A *file specification* string of the form described in 7.11.2, "File Specification Strings," or (if the file system is URL) a uniform resource locator, as described in 7.11.5, "URL Specifications." The **UF** entry should be used in addition to the **F** entry. The **UF** entry provides cross-platform and cross-language compatibility and the **F** entry provides backwards compatibility.
UF	text string	*(Optional, but recommended if the* ***F*** *entry exists in the dictionary; PDF 1.7)* A Unicode text string that provides file specification of the form described in 7.11.2, "File Specification Strings." This is a text string encoded using **PDFDocEncoding** or UTF-16BE with a leading byte-order marker (as defined in 7.9.2.2, "Text String Type"). The **F** entry should be included along with this entry for backwards compatibility reasons.
DOS	byte string	*(Optional)* A file specification string (see 7.11.2, "File Specification Strings") representing a DOS file name. This entry is obsolescent and should not be used by conforming writers.
Mac	byte string	*(Optional)* A file specification string (see 7.11.2, "File Specification Strings") representing a Mac OS file name. This entry is obsolescent and should not be used by conforming writers.
Unix	byte string	*(Optional)* A file specification string (see 7.11.2, "File Specification Strings") representing a UNIX file name. This entry is obsolescent and should not be used by conforming writers.

Table 44 – Entries in a file specification dictionary (continued)

Key	Type	Value
ID	array	*(Optional)* An array of two byte strings constituting a file identifier (see 14.4, "File Identifiers") that should be included in the referenced file. NOTE The use of this entry improves an application's chances of finding the intended file and allows it to warn the user if the file has changed since the link was made.
V	boolean	*(Optional; PDF 1.2)* A flag indicating whether the file referenced by the file specification is *volatile* (changes frequently with time). If the value is **true**, applications shall not cache a copy of the file. For example, a movie annotation referencing a URL to a live video camera could set this flag to **true** to notify the conforming reader that it should re-acquire the movie each time it is played. Default value: **false**.
EF	dictionary	*(Required if* ***RF*** *is present; PDF 1.3; amended to include the* ***UF*** *key in PDF 1.7)* A dictionary containing a subset of the keys **F**, **UF**, **DOS**, **Mac**, and **Unix**, corresponding to the entries by those names in the file specification dictionary. The value of each such key shall be an embedded file stream (see 7.11.4, "Embedded File Streams") containing the corresponding file. If this entry is present, the **Type** entry is required and the file specification dictionary shall be indirectly referenced. The **F** and **UF** entries should be used in place of the **DOS**, **Mac**, or **Unix** entries.
RF	dictionary	*(Optional; PDF 1.3)* A dictionary with the same structure as the **EF** dictionary, which shall be present. Each key in the **RF** dictionary shall also be present in the **EF** dictionary. Each value shall be a related files array (see 7.11.4.2, "Related Files Arrays") identifying files that are related to the corresponding file in the **EF** dictionary. If this entry is present, the **Type** entry is required and the file specification dictionary shall be indirectly referenced.
Desc	text string	*(Optional; PDF 1.6)* Descriptive text associated with the file specification. It shall be used for files in the **EmbeddedFiles** name tree (see 7.7.4, "Name Dictionary").
CI	dictionary	(*Optional; shall be indirect reference; PDF 1.7*) A collection item dictionary, which shall be used to create the user interface for portable collections (see 7.11.6, "Collection Items").

7.11.4 Embedded File Streams

7.11.4.1 General

If a PDF file contains file specifications that refer to an external file and the PDF file is archived or transmitted, some provision should be made to ensure that the external references will remain valid. One way to do this is to arrange for copies of the external files to accompany the PDF file. *Embedded file streams (PDF 1.3)* address this problem by allowing the contents of referenced files to be embedded directly within the body of the PDF file. This makes the PDF file a self-contained unit that can be stored or transmitted as a single entity. (The embedded files are included purely for convenience and need not be directly processed by any conforming reader.)

NOTE If the file contains OPI (Open Prepress Interface) dictionaries that refer to externally stored high-resolution images (see 14.11.7, "Open Prepress Interface (OPI)"), the image data can be incorporated into the PDF file with embedded file streams.

An embedded file stream shall be included in a PDF document in one of the following ways:

- Any file specification dictionary in the document may have an **EF** entry that specifies an embedded file stream. The stream data shall still be associated with a location in the file system. In particular, this method shall be used for file attachment annotations (see 12.5.6.15, "File Attachment Annotations"), which associate the embedded file with a location on a page in the document.

- Embedded file streams may be associated with the document as a whole through the **EmbeddedFiles** entry (*PDF 1.4*) in the PDF document's name dictionary (see 7.7.4, "Name Dictionary"). The associated name tree shall map name strings to file specifications that refer to embedded file streams through their **EF** entries.

Beginning with PDF 1.6, the ***Desc*** entry of the file specification dictionary (see Table 44) should be used to provide a textual description of the embedded file, which can be displayed in the user interface of a conforming reader. Previously, it was necessary to identify document-level embedded files by the name string provided in the name dictionary associated with an embedded file stream in much the same way that the JavaScript name tree associates name strings with document-level JavaScript actions (see 12.6.4.16, "JavaScript Actions").

The stream dictionary describing an embedded file shall contain the standard entries for any stream, such as **Length** and **Filter** (see Table 5), as well as the additional entries shown in Table 45.

Table 45 – Additional entries in an embedded file stream dictionary

Key	**Type**	**Value**
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **EmbeddedFile** for an embedded file stream.
Subtype	name	*(Optional)* The subtype of the embedded file. The value of this entry shall be a first-class name, as defined in Annex E. Names without a registered prefix shall conform to the MIME media type names defined in Internet RFC 2046, *Multipurpose Internet Mail Extensions (MIME), Part Two: Media Types* (see the Bibliography), with the provision that characters not allowed in names shall use the 2-character hexadecimal code format described in 7.3.5, "Name Objects."
Params	dictionary	*(Optional)* An *embedded file parameter dictionary* that shall contain additional file-specific information (see Table 46).

Table 46 – Entries in an embedded file parameter dictionary

Key	**Type**	**Value**
Size	integer	*(Optional)* The size of the uncompressed embedded file, in bytes.
CreationDate	date	*(Optional)* The date and time when the embedded file was created.
ModDate	date	*(Optional)* The date and time when the embedded file was last modified.
Mac	dictionary	*(Optional)* A subdictionary containing additional information specific to Mac OS files (see Table 47).
CheckSum	string	*(Optional)* A 16-byte string that is the checksum of the bytes of the uncompressed embedded file. The checksum shall be calculated by applying the standard MD5 message-digest algorithm (described in Internet RFC 1321, *The MD5 Message-Digest Algorithm*; see the Bibliography) to the bytes of the embedded file stream.

For Mac OS files, the **Mac** entry in the embedded file parameter dictionary should hold a further subdictionary containing Mac OS–specific file information. Table 47 shows the contents of this subdictionary.

Table 47 – Entries in a Mac OS file information dictionary

Key	Type	Value
Subtype	integer	*(Optional)* The embedded file's file type. It shall be encoded as an integer according to Mac OS conventions: a 4-character ASCII text literal, that shall be a 32-bit integer, with the high-order byte first. EXAMPLE The file type "CARO" is represented as the hexadecimal integer 4341524F, which is expressed in decimal as 1128354383.
Creator	integer	*(Optional)* The embedded file's creator signature shall be encoded in the same way as **Subtype**.
ResFork	stream	*(Optional)* The binary contents of the embedded file's resource fork.

7.11.4.2 Related Files Arrays

In some circumstances, a PDF file can refer to a group of related files, such as the set of five files that make up a DCS 1.0 colour-separated image. The file specification explicitly names only one of the files; the rest shall be identified by some systematic variation of that file name (such as by altering the extension). When such a file is to be embedded in a PDF file, the related files shall be embedded as well. This is accomplished by including a *related files array (PDF 1.3)* as the value of the **RF** entry in the file specification dictionary. The array shall have 2 ¥ *n* elements, which shall be paired in the form

$$
\begin{array}{l}
[\ \mathit{string}_1\ \ \mathit{stream}_1 \\
\ \ \mathit{string}_2\ \ \mathit{stream}_2 \\
\ \ \ldots \\
\ \ \mathit{string}_n\ \ \mathit{stream}_n \\
]
\end{array}
$$

The first element of each pair shall be a string giving the name of one of the related files; the second element shall be an embedded file stream holding the file's contents.

EXAMPLE In the following example, objects 21, 31, and 41 are embedded file streams containing the DOS file SUNSET.EPS, the Mac OS file Sunset.eps, and the UNIX file Sunset.eps, respectively. The file specification dictionary's RF entry specifies an array, object 30, identifying a set of embedded files related to the Mac OS file, forming a DCS 1.0 set. The example shows only the first two embedded file streams in the set; an actual PDF file would, of course, include all of them.

```
10 0 obj                                  % File specification dictionary
   << /Type /Filespec
      /DOS (SUNSET.EPS)
      /Mac (Sunset.eps)                      % Name of Mac OS file
      /Unix (Sunset.eps)

      /EF   << /DOS  21 0 R
                /Mac  31 0 R                 % Embedded Mac OS file
                /Unix 41 0 R
            >>
      /RF << /Mac 30 0 R >>                  % Related files array for Mac OS file
   >>
endobj

30 0 obj                                     % Related files array for Mac OS file
   [ (Sunset.eps) 31 0 R                     % Includes file Sunset.eps itself
     (Sunset.C) 32 0 R
     (Sunset.M) 33 0 R
     (Sunset.Y) 34 0 R
```

```
        (Sunset.K)  35 0 R
    ]
endobj

31  0  obj                                      % Embedded file stream for Mac OS file
    <<  /Type  /EmbeddedFile                        % Sunset.eps
         /Length  …
         /Filter  …
    >>
stream
…Data for Sunset.eps…
endstream
endobj

32  0  obj                                      % Embedded file stream for related file
    <<  /Type  /EmbeddedFile                        % Sunset.C
         /Length  …
         /Filter  …
    >>
stream
…Data for Sunset.C…
endstream
endobj
```

7.11.5 URL Specifications

When the **FS** entry in a file specification dictionary has the value **URL**, the value of the **F** entry in that dictionary is not a file specification string, but a uniform resource locator (URL) of the form defined in Internet RFC 1738, *Uniform Resource Locators* (see the Bibliography).

EXAMPLE The following example shows a URL specification.

```
<<  /FS  /URL
     /F  (ftp://www.beatles.com/Movies/AbbeyRoad.mov)
>>
```

The URL shall adhere to the character-encoding requirements specified in RFC 1738. Because 7-bit U.S. ASCII is a strict subset of **PDFDocEncoding**, this value shall also be considered to be in that encoding.

7.11.6 Collection Items

Beginning with PDF 1.7, a *collection item dictionary* shall contain the data described by the collection schema dictionary for a particular file in a collection (see 12.3.5, "Collections"). Table 48 describes the entries in a collection item dictionary.

Table 48 – Entries in a collection item dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be CollectionItem for a collection item dictionary.

Table 48 – Entries in a collection item dictionary (continued)

Key	Type	Value
Other keys	text string, date, number or dictionary	(*Optional*) Provides the data corresponding to the related fields in the collection dictionary. If the entry is a dictionary, then it shall be a collection subitem dictionary (see Table 49). The type of each entry shall match the type of data identified by the collection field dictionary (see Table 157) referenced by the same key in the collection schema dictionary (see Table 156). EXAMPLE If the corresponding collection field has a **Subtype** entry of *S*, then the entry is a text string. A single collection item dictionary may contain multiple entries, with one entry representing each key (see EXAMPLE 1 in 12.3.5, "Collections").

A *collection subitem dictionary* provides the data corresponding to the related fields in the collection dictionary, and it provides a means of associating a prefix string with that data value. The prefix shall be ignored by the sorting algorithm. Table 49 describes the entries in a collection subitem dictionary.

Table 49 – Entries in a collection subitem dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be CollectionSubitem for a collection item dictionary.
D	text string, date, or number	(*Optional*) The data corresponding to the related entry in the collection field dictionary (see Table 157). The type of data shall match the data type identified by the corresponding collection field dictionary. Default: none.
P	text string	(*Optional*) A prefix string that shall be concatenated with the text string presented to the user. This entry is ignored when a conforming reader sorts the items in the collection. Default: none.

7.11.7 Maintenance of File Specifications

The techniques described in this sub-clause can be used to maintain the integrity of the file specifications within a PDF file during the following types of operations:

- Updating the relevant file specification when a referenced file is renamed
- Determining the complete collection of files that are copied to a mirror site
- When creating new links to external files, discovering existing file specifications that refer to the same files and sharing them
- Finding the file specifications associated with embedded files to be packed or unpacked

NOTE 1 It is not possible, in general, to find all file specification strings in a PDF file because there is no way to determine whether a given string is a file specification string. It is possible, however, to find all file specification *dictionaries*, provided that they meet the following conditions:

They are indirect objects.

They contain a **Type** entry whose value is the name **Filespec**.

NOTE 2 A conforming reader can locate all of the file specification dictionaries by traversing the PDF file's cross-reference table (see 7.5.4, "Cross-Reference Table") and finding all dictionaries with **Type** keys whose value is **Filespec**. For this reason, all file specifications should be expressed in dictionary form and meet the conditions stated above. Any file specification dictionary specifying embedded files (that is, one that contains an **EF** entry) should satisfy these conditions (see Table 44).

NOTE 3 It may not be possible to locate file specification dictionaries that are direct objects, since they are neither self-typed nor necessarily reachable by any standard path of object references.

NOTE 4 Files may be embedded in a PDF file either directly, using the **EF** entry in a file specification dictionary, or indirectly, using related files arrays specified in the **RF** entry. If a file is embedded indirectly, its name is given by the string that precedes the embedded file stream in the related files array. If it is embedded directly, its name is obtained from the value of the corresponding entry in the file specification dictionary.

EXAMPLE The EXAMPLE in 7.11.4.2, "Related Files Arrays," for instance, shows the **EF** dictionary having a DOS entry identifying object number 21 as an embedded file stream. The name of the embedded DOS file, SUNSET.EPS, is given by the DOS entry in the file specification dictionary.

NOTE 5 A given external file may be referenced from more than one file specification. Therefore, when embedding a file with a given name, it is recommended to check for other occurrences of the same name as the value associated with the corresponding key in other file specification dictionaries. This requires finding all embeddable file specifications and, for each matching key, checking for both of the following conditions:

The string value associated with the key matches the name of the file being embedded.

A file has not already been embedded for the file specification.

NOTE 6 If there is already a corresponding key in the **EF** dictionary, a file has already been embedded for that use of the file name.

NOTE 7 Files associated with a given file name need not be unique. The same file name, such as readme.txt, may be associated with different embedded files in distinct file specifications.

7.12 Extensions Dictionary

7.12.1 General

The extensions dictionary, an entry in the document's catalog dictionary, if present, shall contain one or more entries identifying developer-defined extensions to the ISO 32000-1 Standard. An extensions dictionary, not shown, may optionally contain a **Type** entry whose value is the name **Extensions**. The keys in the extensions dictionary shall be names consisting only of the registered prefixes, described in Annex E, of the developers whose extensions are being used. The values shall be developer extensions dictionaries specifying developer-defined version information as shown in Table 50. The extensions dictionary, all developer extensions dictionary entries in the extensions dictionary, as well as their entries, all shall be direct objects (i.e., this information shall be nested directly within the catalog dictionary with no indirect objects used).

7.12.2 Developer Extensions Dictionary

Table 50 describes the entries in a developer extensions dictionary.

Table 50 – Entries in a developer extensions dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be **DeveloperExtensions**.
BaseVersion	name	(*Required*) The name of the PDF version to which this extension applies. The name shall be consistent with the syntax used for the **Version** entry of the catalog dictionary (see 7.7.2, "Document Catalog").
ExtensionLevel	integer	(*Required*) An integer defined by the developer to denote the extension being used. If the developer introduces more than one extension to a given **BaseVersion** the extension level numbers assigned by that developer shall increase over time.

7.12.3 BaseVersion

The value of the **BaseVersion** entry shall be a name and shall be consistent with the syntax used for the **Version** entry value of the catalog dictionary (see 7.7.2, “Document Catalog”). The value of **BaseVersion**, when treated as a version number, shall be less than or equal to the PDF version, both in the document header (see 7.5.2, “File Header”) and the catalog **Version** key value, if present. The value of **BaseVersion** may be different from the version number in the document header or that supplied by the **Version** key in the **Catalog**. This is because it reflects the version of the standard that has been extended and not the version of this particular file.

NOTE 1 The value of **BaseVersion** is not to be interpreted as a real number but as two integers with a PERIOD (2Eh) between them.

7.12.4 ExtensionLevel

The value of the **ExtensionLeve**l entry shall be an integer, which shall be interpreted with respect to the **BaseVersion** value. If a developer has released multiple extensions against the same **BaseVersion** value, they shall be ordered over time and the **ExtensionsLeve**l numbers shall be a monotonically increasing sequence over time.

EXAMPLE 1

```
%PDF–1.7
<</Type /Catalog
  /Extensions
      <</ADBE
          <</BaseVersion  /1.7
             /ExtensionLevel  3
          >>
      >>
>>
```

EXAMPLE 2

```
%PDF–1.7
<</Type /Catalog
  /Extensions
      <</GLGR
          <</BaseVersion  /1.7
             /ExtensionLevel  1002
          >>
      >>
>>
```

EXAMPLE 3

```
%PDF–1.7
<</Type /Catalog
  /Extensions
      <</ADBE
          <</BaseVersion  /1.7
             /ExtensionLevel  3
          >>
         /GLGR
          <</BaseVersion  /1.7
             /ExtensionLevel  1002
          >>
      >>
>>
```

8 Graphics

8.1 General

The graphics operators used in PDF content streams describe the appearance of pages that are to be reproduced on a raster output device. The facilities described in this clause are intended for both printer and display applications.

The graphics operators form six main groups:

- *Graphics state operators* manipulate the data structure called the *graphics state*, the global framework within which the other graphics operators execute. The graphics state includes the *current transformation matrix* (CTM), which maps user space coordinates used within a PDF content stream into output device coordinates. It also includes the *current colour*, the *current clipping path*, and many other parameters that are implicit operands of the painting operators.

- *Path construction operators* specify *paths*, which define shapes, line trajectories, and regions of various sorts. They include operators for beginning a new path, adding line segments and curves to it, and closing it.

- *Path-painting operators* fill a path with a colour, paint a stroke along it, or use it as a clipping boundary.

- *Other painting operators* paint certain self-describing graphics objects. These include sampled images, geometrically defined shadings, and entire content streams that in turn contain sequences of graphics operators.

- *Text operators* select and show *character glyphs* from *fonts* (descriptions of typefaces for representing text characters). Because PDF treats glyphs as general graphical shapes, many of the text operators could be grouped with the graphics state or painting operators. However, the data structures and mechanisms for dealing with glyph and font descriptions are sufficiently specialized that clause 9, "Text" focuses on them.

- *Marked-content operators* associate higher-level logical information with objects in the content stream. This information does not affect the rendered appearance of the content (although it may determine if the content should be presented at all; see 8.11, "Optional Content"); it is useful to applications that use PDF for document interchange. Marked content is described in 14.6, "Marked Content".

This clause presents general information about device-independent graphics in PDF: how a PDF content stream describes the abstract appearance of a page. *Rendering*—the device-dependent part of graphics—is covered in clause 10, "Rendering". The Bibliography lists a number of books that give details of these computer graphics concepts and their implementation.

8.2 Graphics Objects

As discussed in 7.8.2, "Content Streams", the data in a content stream shall be interpreted as a sequence of *operators* and their *operands*, expressed as basic data objects according to standard PDF syntax. A content stream can describe the appearance of a page, or it can be treated as a graphical element in certain other contexts.

The operands and operators shall be written sequentially using postfix notation. Although this notation resembles the sequential execution model of the PostScript language, a PDF content stream is not a program to be interpreted; rather, it is a static description of a sequence of *graphics objects*. There are specific rules, described below, for writing the operands and operators that describe a graphics object.

PDF provides five types of graphics objects:

- A *path object* is an arbitrary shape made up of straight lines, rectangles, and cubic Bézier curves. A path may intersect itself and may have disconnected sections and holes. A path object ends with one or more painting operators that specify whether the path shall be stroked, filled, used as a clipping boundary, or some combination of these operations.

- A *text object* consists of one or more character strings that identify sequences of glyphs to be painted. Like a path, text can be stroked, filled, or used as a clipping boundary.

- An *external object (XObject)* is an object defined outside the content stream and referenced as a named resource (see 7.8.3, "Resource Dictionaries"). The interpretation of an XObject depends on its type. An *image XObject* defines a rectangular array of colour samples to be painted; a *form XObject* is an entire content stream to be treated as a single graphics object. Specialized types of form XObjects shall be used to import content from one PDF file into another *(reference XObjects)* and to group graphical elements together as a unit for various purposes *(group XObjects)*. In particular, the latter are used to define *transparency groups* for use in the transparent imaging model (*transparency group XObjects*, discussed in detail in clause 11, "Transparency"). There is also a PostScript XObject that may appear in some existing PDF files, but it should not be used by a PDF 1.7 conforming writer.

- An *inline image object* uses a special syntax to express the data for a small image directly within the content stream.

- A *shading object* describes a geometric shape whose colour is an arbitrary function of position within the shape. (A shading can also be treated as a colour when painting other graphics objects; it is not considered to be a separate graphics object in that case.)

PDF 1.3 and earlier versions use an *opaque imaging model* in which each graphics object is painted in sequence, completely obscuring any previous marks it may overlay on the page. PDF 1.4 introduced a *transparent imaging model* in which objects can be less than fully opaque, allowing previously painted marks to show through. Each object is painted on the page with a specified *opacity*, which may be constant at every point within the object's shape or may vary from point to point. The previously existing contents of the page form a *backdrop* with which the new object is *composited*, producing results that combine the colours of the object and backdrop according to their respective opacity characteristics. The objects at any given point on the page *forms* a *transparency stack*, where the stacking order is defined to be the order in which the objects shall be specified, bottommost object first. All objects in the stack can potentially contribute to the result, depending on their colours, shapes, and opacities.

PDF's graphics parameters are so arranged that objects shall be painted by default with full opacity, reducing the behaviour of the transparent imaging model to that of the opaque model. Accordingly, the material in this clause applies to both the opaque and transparent models except where explicitly stated otherwise; the transparent model is described in its full generality in clause 11, "Transparency".

Although the painting behaviour described above is often attributed to individual operators making up an object, it is always the object as a whole that is painted. Figure 9 in Annex L shows the ordering rules for the operations that define graphics objects. Some operations shall be permitted only in certain types of graphics objects or in the intervals between graphics objects (called the *page description level* in the figure). Every content stream begins at the page description level, where changes may be made to the graphics state, such as colours and text attributes, as discussed in the following sub-clauses.

In the Figure 9 in Annex L, arrows indicate the operators that mark the beginning or end of each type of graphics object. Some operators are identified individually, others by general category. Table 51 summarizes these categories for all PDF operators.

Table 51 – Operator Categories

Category	Operators	Table
General graphics state	**w**, **J**, **j**, **M**, **d**, **ri**, **i**, **gs**	57
Special graphics state	**q**, **Q**, **cm**	57
Path construction	**m**, **l**, **c**, **v**, **y**, **h**, **re**	59

Table 51 – Operator Categories (continued)

Category	**Operators**	**Table**
Path painting	**S**, **s**, **f**, **F**, **f***, **B**, **B***, **b**, **b***, **n**	60
Clipping paths	**W**, **W***	61
Text objects	**BT**, **ET**	107
Text state	**Tc**, **Tw**, **Tz**, **TL**, **Tf, Tr, Ts**	
Text positioning	**Td**, **TD**, **Tm**, **T***	108
Text showing	**Tj**, **TJ**, **'**, **"**	109
Type 3 fonts	**d0**, **d1**	113
Color	**CS**, **cs**, **SC**, **SCN**, **sc**, **scn**, **G**, **g**, **RG**, **rg**, **K**, **k**	74
Shading patterns	**sh**	77
Inline images	**BI**, **ID**, **EI**	92
XObjects	**Do**	87
Marked content	**MP**, **DP**, **BMC**, **BDC**, **EMC**	320
Compatibility	**BX**, **EX**	32

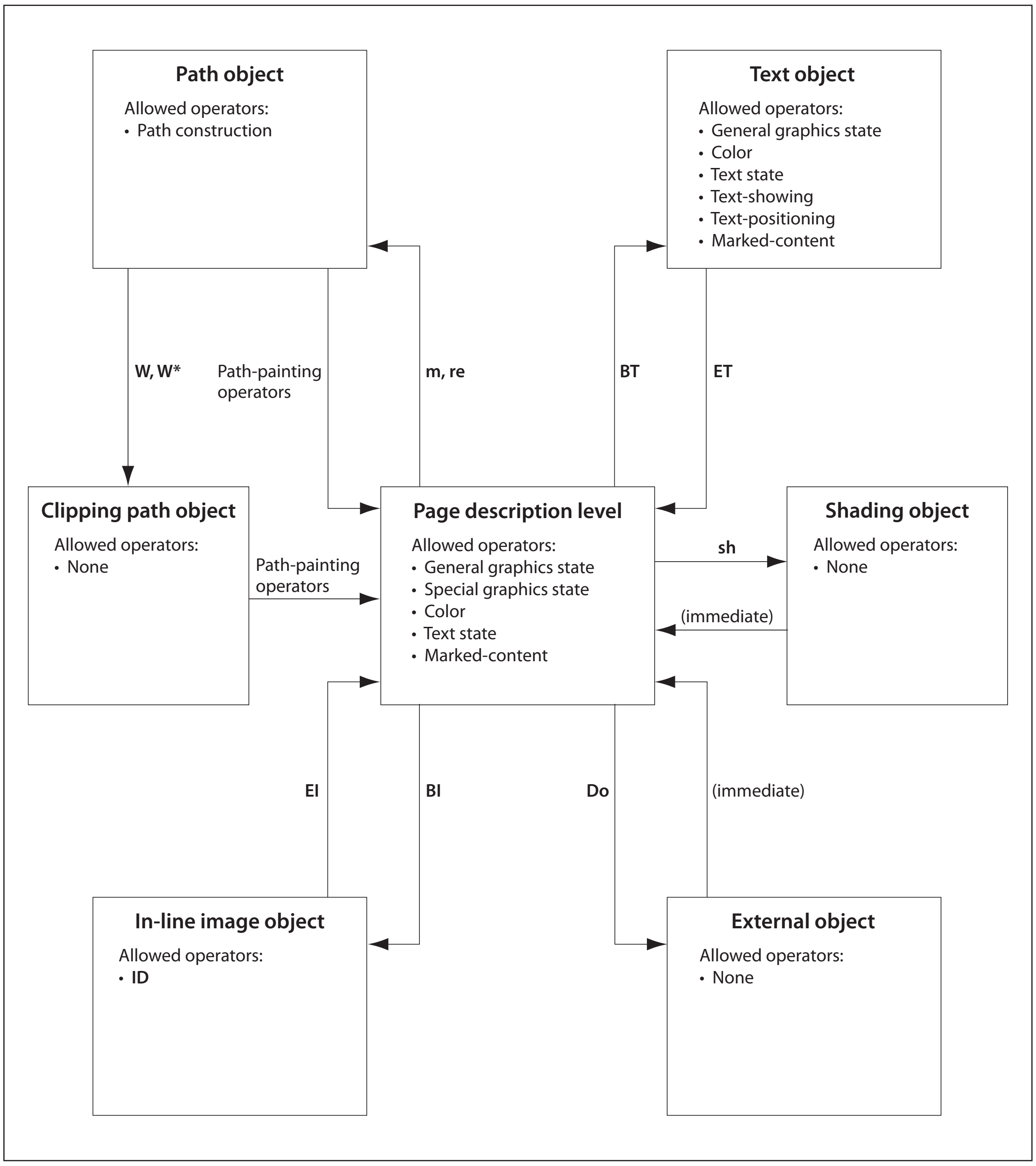


Figure 9 – Graphics Objects

EXAMPLE The path construction operators **m** and **re** signal the beginning of a path object. Inside the path object, additional path construction operators are permitted, as are the clipping path operators **W** and **W***, but not general graphics state operators such as **w** or **J**. A path-painting operator, such as **S** or **f**, ends the path object and returns to the page description level.

NOTE A conforming reader may process a content stream whose operations violate these rules for describing graphics objects and can produce unpredictable behaviour, even though it may display and print *the stream* correctly. Applications that attempt to extract graphics objects for editing or other purposes depend on the objects' being well formed. The rules for graphics objects are also important for the proper interpretation of marked content (see 14.6, "Marked Content").

A graphics object also implicitly includes all graphics state parameters that affect its behaviour. For instance, a path object depends on the value of the current colour parameter at the moment the path object is defined. The effect shall be as if this parameter were specified as part of the definition of the path object. However, the operators that are invoked at the page description level to set graphics state parameters shall not be considered to belong to any particular graphics object. Graphics state parameters should be specified only when they change. A graphics object can depend on parameters that were defined much earlier.

Similarly, the individual character strings within a text object implicitly include the graphics state parameters on which they depend. Most of these parameters may be set inside or outside the text object. The effect is as if they were separately specified for each text string.

The important point is that there is no semantic significance to the exact arrangement of graphics state operators. A conforming reader or writer of a PDF content stream may change an arrangement of graphics state operators to any other arrangement that achieves the same values of the relevant graphics state parameters for each graphics object. A conforming reader or writer shall not infer any higher-level logical semantics from the arrangement of tokens constituting a graphics object. A separate mechanism, *marked content* (see 14.6, "Marked Content"), allows such higher-level information to be explicitly associated with the graphics objects.

8.3 Coordinate Systems

8.3.1 General

Coordinate systems define the canvas on which all painting occurs. They determine the position, orientation, and size of the text, graphics, and images that appear on a page. This sub-clause describes each of the coordinate systems used in PDF, how they are related, and how transformations among them are specified.

NOTE The coordinate systems discussed in this sub-clause apply to two-dimensional graphics. PDF 1.6 introduced the ability to display 3D artwork, in which objects are described in a three-dimensional coordinate system, as described in 13.6.5, "Coordinate Systems for 3D".

8.3.2 Coordinate Spaces

8.3.2.1 General

Paths and positions shall be defined in terms of pairs of *coordinates* on the Cartesian plane. A coordinate pair is a pair of real numbers *x* and *y* that locate a point horizontally and vertically within a two-dimensional *coordinate space*. A coordinate space is determined by the following properties with respect to the current page:

- The location of the origin
- The orientation of the *x* and *y* axes
- The lengths of the units along each axis

PDF defines several coordinate spaces in which the coordinates specifying graphics objects shall be interpreted. The following sub-clauses describe these spaces and the relationships among them.

Transformations among coordinate spaces shall be defined by *transformation matrices*, which can specify any linear mapping of two-dimensional coordinates, including translation, scaling, rotation, reflection, and skewing. Transformation matrices are discussed in 8.3.3, "Common Transformations" and 8.3.4, "Transformation Matrices".

8.3.2.2 Device Space

The contents of a page ultimately appear on a raster output device such as a display or a printer. Such devices vary greatly in the built-in coordinate systems they use to address pixels within their imageable areas. A particular device's coordinate system is called its device space. The origin of the device space on different devices can fall in different places on the output page; on displays, the origin can vary depending on the window system. Because the paper or other output medium moves through different printers and imagesetters in different directions, the axes of their device spaces may be oriented differently. For instance, vertical (y) coordinates may increase from the top of the page to the bottom on some devices and from bottom to top on others. Finally, different devices have different resolutions; some even have resolutions that differ in the horizontal and vertical directions.

NOTE If coordinates in a PDF file were specified in device space, the file would be device-dependent and would appear differently on different devices.

EXAMPLE Images specified in the typical device spaces of a 72-pixel-per-inch display and a 600-dot-per-inch printer would differ in size by more than a factor of 8; an 8-inch line segment on the display would appear less than 1 inch long on the printer. Figure 10 in Annex L shows how the same graphics object, specified in device space, can appear drastically different when rendered on different output devices.

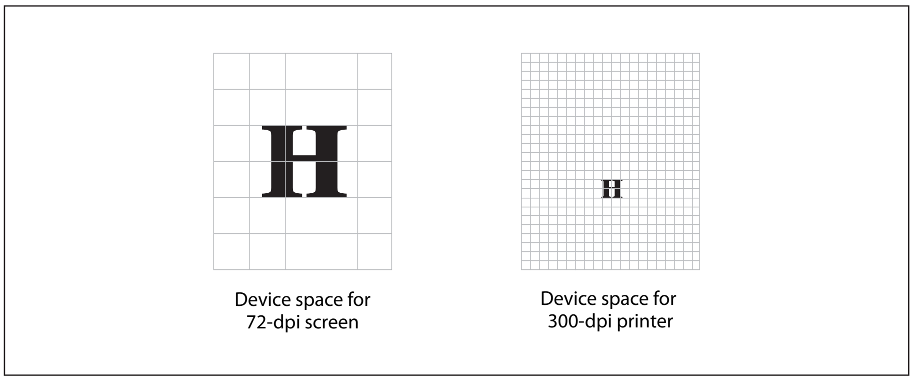

H
H
Device space for 72-dpi screen
Device space for 300-dpi printer

Figure 10 – Device Space

8.3.2.3 User Space

To avoid the device-dependent effects of specifying objects in device space, PDF defines a device-independent coordinate system that always bears the same relationship to the current page, regardless of the output device on which printing or displaying occurs. This device-independent coordinate system is called *user space*.

The user space coordinate system shall be initialized to a default state for each page of a document. The **CropBox** entry in the page dictionary shall specify the rectangle of user space corresponding to the visible area of the intended output medium (display window or printed page). The positive *x* axis extends horizontally to the right and the positive *y* axis vertically upward, as in standard mathematical practice (subject to alteration by the **Rotate** entry in the page dictionary). The length of a unit along both the *x* and *y* axes is set by the **UserUnit** entry *(PDF 1.6)* in the page dictionary (see Table 30). If that entry is not present or supported, the default value of 1⁄72 inch is used. This coordinate system is called *default user space*.

NOTE 1 In PostScript, the origin of default user space always corresponds to the lower-left corner of the output medium. While this convention is common in PDF documents as well, it is not required; the page dictionary's ***CropBox*** entry can specify any rectangle of default user space to be made visible on the medium.

NOTE 2 The default for the size of the unit in default user space (1⁄72 inch) is approximately the same as a *point*, a unit widely used in the printing industry. It is not exactly the same, however; there is no universal definition of a point.

Conceptually, user space is an infinite plane. Only a small portion of this plane corresponds to the imageable area of the output device: a rectangular region defined by the **CropBox** entry in the page dictionary. The region of default user space that is viewed or printed can be different for each page and is described in 14.11.2, "Page Boundaries".

Coordinates in user space (as in any other coordinate space) may be specified as either integers or real numbers, and the unit size in default user space does not constrain positions to any arbitrary grid. The resolution of coordinates in user space is not related in any way to the resolution of pixels in device space.

The transformation from user space to device space is defined by the *current transformation matrix* (CTM), an element of the PDF graphics state (see 8.4, "Graphics State"). A conforming reader can adjust the CTM for the native resolution of a particular output device, maintaining the device-independence of the PDF page description. Figure 11 in Annex L shows how this allows an object specified in user space to appear the same regardless of the device on which it is rendered.

NOTE 3 The default user space provides a consistent, dependable starting place for PDF page descriptions regardless of the output device used. If necessary, a PDF content stream may modify user space to be more suitable to its needs by applying the *coordinate transformation operator*, **cm** (see 8.4.4, "Graphics State Operators"). Thus, what may appear to be absolute coordinates in a content stream are not absolute with respect to the current page because they are expressed in a coordinate system that may slide around and shrink or expand. Coordinate system transformation not only enhances device-independence but is a useful tool in its own right.

EXAMPLE A content stream originally composed to occupy an entire page can be incorporated without change as an element of another page by shrinking the coordinate system in which it is drawn.

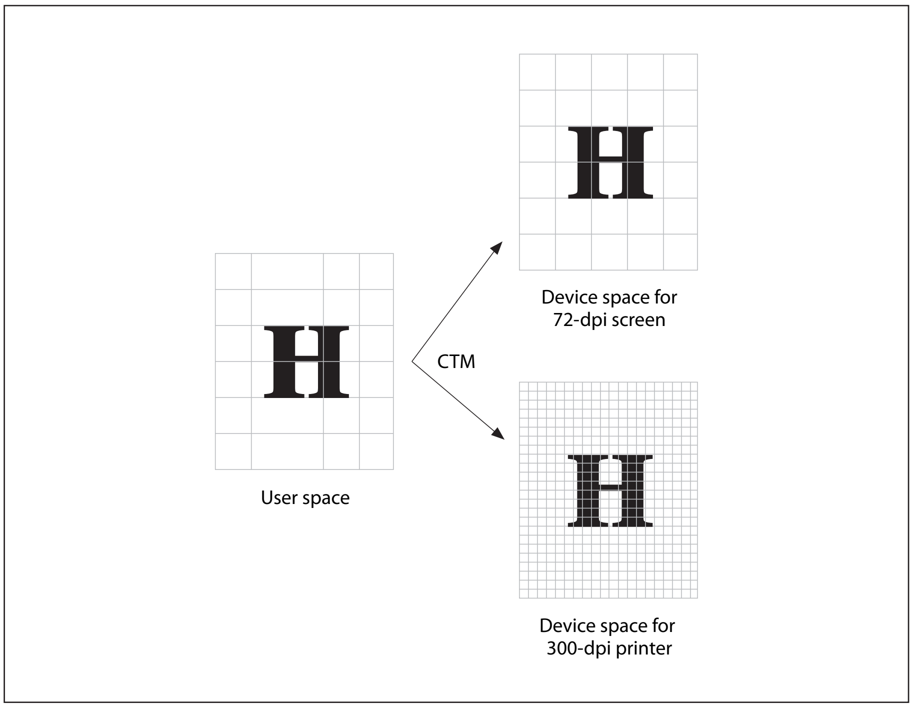


Figure 11 – User Space

8.3.2.4 Other Coordinate Spaces

In addition to device space and user space, PDF uses a variety of other coordinate spaces for specialized purposes:

- The coordinates of text shall be specified in *text space*. The transformation from text space to user space shall be defined by a *text matrix* in combination with several text-related parameters in the graphics state (see 9.4.2, "Text-Positioning Operators").
- Character glyphs in a font shall be defined in *glyph space* (see 9.2.4, "Glyph Positioning and Metrics"). The transformation from glyph space to text space shall be defined by the *font matrix*. For most types of fonts, this matrix shall be predefined to map 1000 units of glyph space to 1 unit of text space; for Type 3 fonts, the font matrix shall be given explicitly in the font dictionary (see 9.6.5, "Type 3 Fonts").
- All sampled images shall be defined in *image space*. The transformation from image space to user space shall be predefined and cannot be changed. All images shall be 1 unit wide by 1 unit high in user space,

regardless of the number of samples in the image. To be painted, an image shall be mapped to a region of the page by temporarily altering the CTM.

- A form XObject (discussed in 8.10, "Form XObjects") is a self-contained content stream that can be treated as a graphical element within another content stream. The space in which it is defined is called *form space*. The transformation from form space to user space shall be specified by a *form matrix* contained in the form XObject.
- PDF 1.2 defined a type of colour known as a *pattern*, discussed in 8.7, "Patterns". A pattern shall be defined either by a content stream that shall be invoked repeatedly to tile an area or by a shading whose colour is a function of position. The space in which a pattern is defined is called *pattern space*. The transformation from pattern space to user space shall be specified by a *pattern matrix* contained in the pattern.
- PDF 1.6 embedded 3D artwork, which is described in three-dimensional coordinates (see 13.6.5, "Coordinate Systems for 3D") that are projected into an annotation's target coordinate system (see 13.6.2, "3D Annotations").

8.3.2.5 Relationships among Coordinate Spaces

Figure 12 in Annex L shows the relationships among the coordinate spaces described above. Each arrow in the figure represents a transformation from one coordinate space to another. PDF allows modifications to many of these transformations.

Because PDF coordinate spaces are defined relative to one another, changes made to one transformation can affect the appearance of objects defined in several coordinate spaces.

EXAMPLE A change in the CTM, which defines the transformation from user space to device space, affects forms, text, images, and patterns, since they are all upstream from user space.

8.3.3 Common Transformations

A *transformation matrix* specifies the relationship between two coordinate spaces. By modifying a transformation matrix, objects can be scaled, rotated, translated, or transformed in other ways.

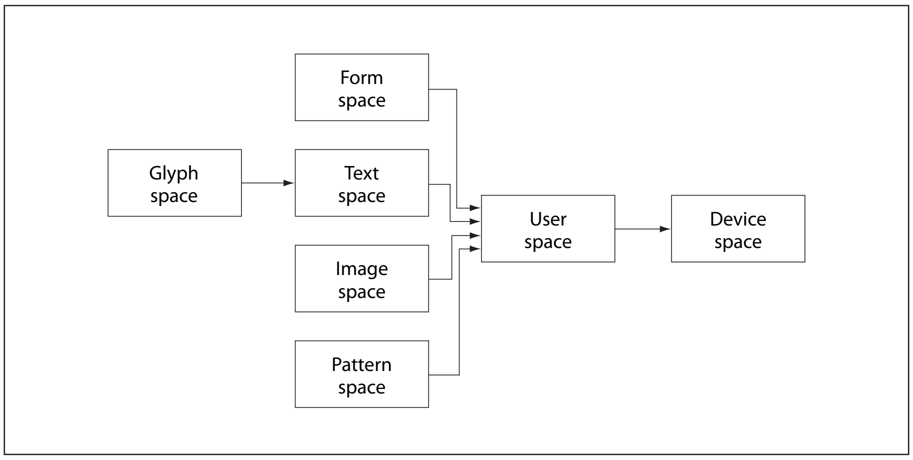


Figure 12 – Relationships Among Coordinate Systems

A transformation matrix in PDF shall be specified by six numbers, usually in the form of an array containing six elements. In its most general form, this array is denoted [*a b c d e f*]; it can represent any linear transformation from one coordinate system to another. This sub-clause lists the arrays that specify the most common transformations; 8.3.4, "Transformation Matrices", discusses more mathematical details of transformations, including information on specifying transformations that are combinations of those listed here:

- Translations shall be specified as [1 0 0 1 t_x t_y], where t_x and t_y shall be the distances to translate the origin of the coordinate system in the horizontal and vertical dimensions, respectively.
- Scaling shall be obtained by [s_x 0 0 s_y 0 0]. This scales the coordinates so that 1 unit in the horizontal and vertical dimensions of the new coordinate system is the same size as s_x and s_y units, respectively, in the previous coordinate system.
- Rotations shall be produced by [cos *q* sin *q* -sin *q* cos *q* 0 0], which has the effect of rotating the coordinate system axes by an angle *q* counter clockwise.
- Skew shall be specified by [1 tan *a* tan *b* 1 0 0], which skews the *x* axis by an angle *a* and the *y* axis by an angle *b*.

Figure 13 in Annex L shows examples of each transformation. The directions of translation, rotation, and skew shown in the figure correspond to positive values of the array elements.

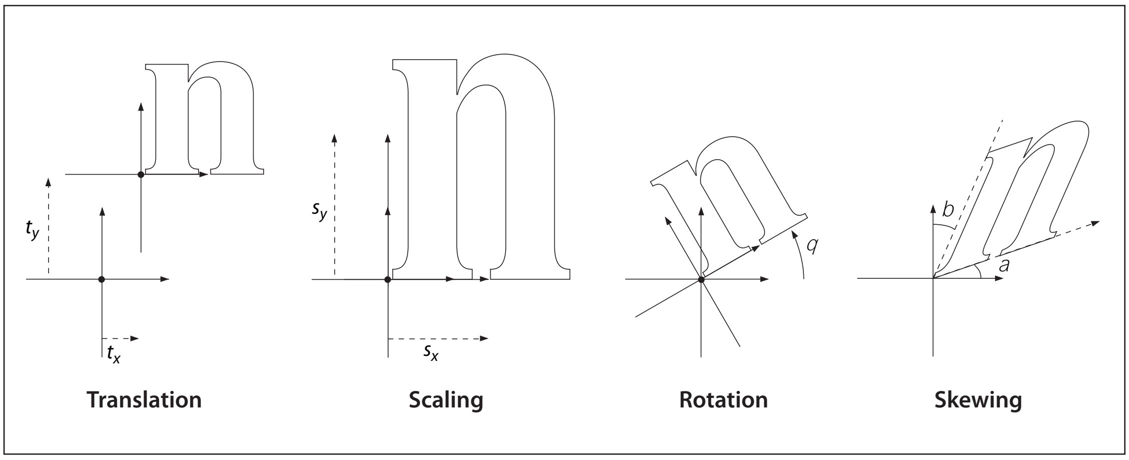


Figure 13 – Effects of Coordinate Transformations

NOTE If several transformations are combined, the order in which they are applied is significant. For example, first scaling and then translating the *x* axis is not the same as first translating and then scaling it. In general, to obtain the expected results, transformations should be done in the following order: Translate, Rotate, Scale or skew.

Figure 14 in Annex L shows the effect of the order in which transformations are applied. The figure shows two sequences of transformations applied to a coordinate system. After each successive transformation, an outline of the letter n is drawn.

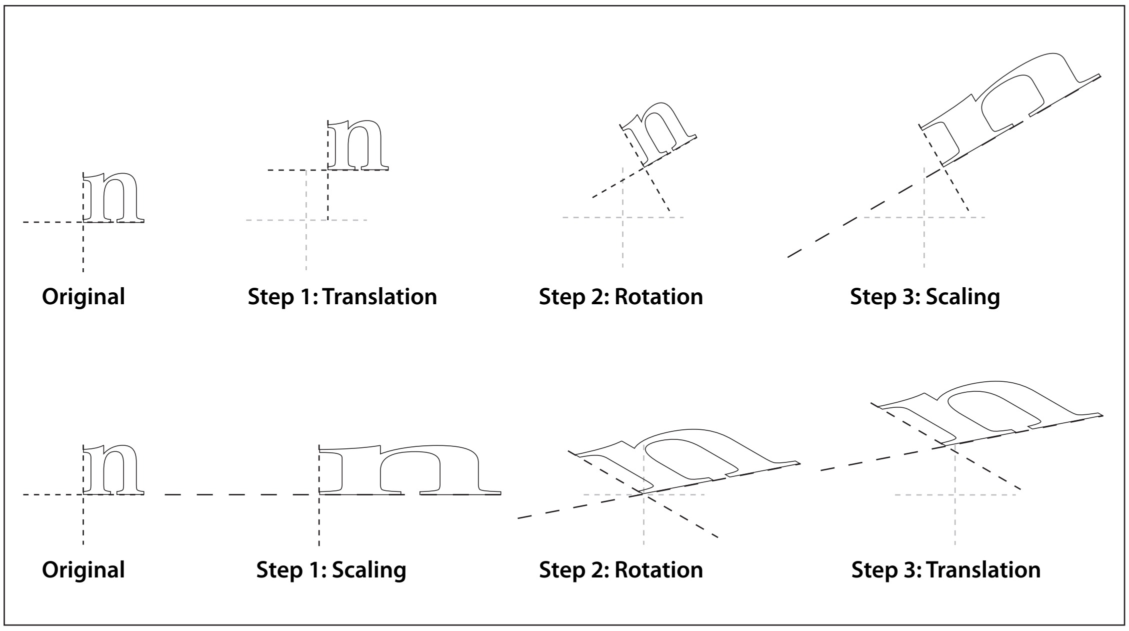


Figure 14 – Effect of Transformation Order

NOTE The following transformations are shown in the figure: a translation of 10 units in the *x* direction and 20 units in the *y* direction; a rotation of 30 degrees; a scaling by a factor of 3 in the *x* direction

In the figure, the axes are shown with a dash pattern having a 2-unit dash and a 2-unit gap. In addition, the original (untransformed) axes are shown in a lighter colour for reference. Notice that the scale-rotate-translate ordering results in a distortion of the coordinate system, leaving the *x* and *y* axes no longer perpendicular; the recommended translate-rotate-scale ordering results in no distortion.

8.3.4 Transformation Matrices

This sub-clause discusses the mathematics of transformation matrices.

To understand the mathematics of coordinate transformations in PDF, it is vital to remember two points:

- *Transformations alter coordinate systems, not graphics objects*. All objects painted before a transformation is applied shall be unaffected by the transformation. Objects painted after the transformation is applied shall be interpreted in the transformed coordinate system.

- *Transformation matrices specify the transformation from the new (transformed) coordinate system to the original (untransformed) coordinate system*. All coordinates used after the transformation shall be expressed in the transformed coordinate system. PDF applies the transformation matrix to find the equivalent coordinates in the untransformed coordinate system.

NOTE 1 Many computer graphics textbooks consider transformations of graphics objects rather than of coordinate systems. Although either approach is correct and self-consistent, some details of the calculations differ depending on which point of view is taken.

PDF represents coordinates in a two-dimensional space. The point (*x*, *y*) in such a space can be expressed in vector form as [*x* *y* 1]. The constant third element of this vector (1) is needed so that the vector can be used with 3-by-3 matrices in the calculations described below.

The transformation between two coordinate systems can be represented by a 3-by-3 transformation matrix written as follows:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Because a transformation matrix has only six elements that can be changed, in most cases in PDF it shall be specified as the six-element array [*a b c d e f*].

Coordinate transformations shall be expressed as matrix multiplications:

$$[\,x'\ y'\ 1\,] = [\,x\ y\ 1\,] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Because PDF transformation matrices specify the conversion from the transformed coordinate system to the original (untransformed) coordinate system, *x¢* and *y¢* in this equation shall be the coordinates in the untransformed coordinate system, and *x* and *y* shall be the coordinates in the transformed system. The multiplication is carried out as follows:

$$x' = a \times x + c \times y + e$$
$$y' = b \times x + d \times y + f$$

If a series of transformations is carried out, the matrices representing each of the individual transformations can be multiplied together to produce a single equivalent matrix representing the composite transformation.

NOTE 2 Matrix multiplication is not commutative—the order in which matrices are multiplied is significant. Consider a sequence of two transformations: a scaling transformation applied to the user space coordinate system, followed by a conversion from the resulting scaled user space to device space. Let M_S be the matrix specifying the scaling and M_C the current transformation matrix, which transforms user space to device space. Recalling that coordinates are always specified in the transformed space, the correct order of transformations first converts the scaled coordinates to default user space and then converts the default user space coordinates to device space. This can be expressed as

$$X_D = X_U \times M_C = (X_S \times M_S) \times M_C = X_S \times (M_S \times M_C$$

where

X_D denotes the coordinates in device space

X_U denotes the coordinates in default user space

X_S denotes the coordinates in scaled user space

This shows that when a new transformation is concatenated with an existing one, the matrix representing it shall be multiplied *before* (*premultiplied* with) the existing transformation matrix.

This result is true in general for PDF: when a sequence of transformations is carried out, the matrix representing the combined transformation *(M¢)* is calculated by premultiplying the matrix representing the additional transformation *(M_T)* with the one representing all previously existing transformations *(M)*:

$$M' = M_T \times M$$

NOTE 3 When rendering graphics objects, it is sometimes necessary for a conforming reader to perform the inverse of a transformation—that is, to find the user space coordinates that correspond to a given pair of device space coordinates. Not all transformations are invertible, however. For example, if a matrix contains *a*, *b*, *c*, and *d* elements that are all zero, all user coordinates map to the same device coordinates and there is no unique inverse transformation. Such noninvertible transformations are not very useful and generally arise from unintended operations, such as scaling by 0. Use of a noninvertible matrix when painting graphics objects can result in unpredictable behaviour.

8.4 Graphics State

8.4.1 General

A conforming reader shall maintain an internal data structure called the *graphics state* that holds current graphics control parameters. These parameters define the global framework within which the graphics operators execute.

EXAMPLE 1 The **f** (fill) operator implicitly uses the current colour parameter, and the **S** (stroke) operator additionally uses the current line width parameter from the graphics state.

A conforming reader shall initialize the graphic state at the beginning of each page with the values specified in Table 52 and Table 53. Table 52 lists those graphics state parameters that are device-independent and are appropriate to specify in page descriptions. The parameters listed in Table 53 control details of the rendering (scan conversion) process and are device-dependent; a page description that is intended to be device-independent should not be written to modify these parameters.

Table 52 – Device-Independent Graphics State Parameters

Parameter	Type	Value
CTM	array	The *current transformation matrix*, which maps positions from user coordinates to device coordinates (see 8.3, "Coordinate Systems"). This matrix is modified by each application of the coordinate transformation operator, **cm**. Initial value: a matrix that transforms default user coordinates to device coordinates.
clipping path	(internal)	The *current clipping path*, which defines the boundary against which all output shall be cropped (see 8.5.4, "Clipping Path Operators"). Initial value: the boundary of the entire imageable portion of the output page.
color space	name or array	The *current colour space* in which colour values shall be interpreted (see 8.6, "Colour Spaces"). There are two separate colour space parameters: one for stroking and one for all other painting operations. Initial value: **DeviceGray**.
color	(various)	The *current colour* to be used during painting operations (see 8.6, "Colour Spaces"). The type and interpretation of this parameter depend on the current colour space; for most colour spaces, a colour value consists of one to four numbers. There are two separate colour parameters: one for stroking and one for all other painting operations. Initial value: black.
text state	(various)	A set of nine graphics state parameters that pertain only to the painting of text. These include parameters that select the font, scale the glyphs to an appropriate size, and accomplish other effects. The text state parameters are described in 9.3, "Text State Parameters and Operators".
line width	number	The thickness, in user space units, of paths to be stroked (see 8.4.3.2, "Line Width"). Initial value: 1.0.

Table 52 – Device-Independent Graphics State Parameters (continued)

Parameter	Type	Value
line cap	integer	A code specifying the shape of the endpoints for any open path that is stroked (see 8.4.3.3, "Line Cap Style"). Initial value: 0, for square butt caps.
line join	integer	A code specifying the shape of joints between connected segments of a stroked path (see 8.4.3.4, "Line Join Style"). Initial value: 0, for mitered joins.
miter limit	number	The maximum length of mitered line joins for stroked paths (see 8.4.3.5, "Miter Limit"). This parameter limits the length of “spikes” produced when line segments join at sharp angles. Initial value: 10.0, for a miter cutoff below approximately 11.5 degrees.
dash pattern	array and number	A description of the dash pattern to be used when paths are stroked (see 8.4.3.6, "Line Dash Pattern"). Initial value: a solid line.
rendering intent	name	The *rendering intent* to be used when converting CIE-based colours to device colours (see 8.6.5.8, "Rendering Intents"). Initial value: **RelativeColorimetric**.
stroke adjustment	boolean	*(PDF 1.2)* A flag specifying whether to compensate for possible rasterization effects when stroking a path with a line width that is small relative to the pixel resolution of the output device (see 10.6.5, "Automatic Stroke Adjustment"). NOTE This is considered a device-independent parameter, even though the details of its effects are device-dependent. Initial value: **false**.
blend mode	name or array	*(PDF 1.4)* The *current blend mode* to be used in the transparent imaging model (see 11.3.5, "Blend Mode" and 11.6.3, "Specifying Blending Colour Space and Blend Mode"). A conforming reader shall implicitly reset this parameter to its initial value at the beginning of execution of a transparency group XObject (see 11.6.6, "Transparency Group XObjects"). Initial value: **Normal**.
soft mask	dictionary or name	*(PDF 1.4)* A *soft-mask dictionary* (see 11.6.5.2, "Soft-Mask Dictionaries") specifying the mask shape or mask opacity values to be used in the transparent imaging model (see 11.3.7.2, "Source Shape and Opacity" and 11.6.4.3, "Mask Shape and Opacity"), or the name **None** if no such mask is specified. A conforming reader shall implicitly reset this parameter implicitly reset to its initial value at the beginning of execution of a transparency group XObject (see 11.6.6, "Transparency Group XObjects"). Initial value: **None**.
alpha constant	number	*(PDF 1.4)* The constant shape or constant opacity value to be used in the transparent imaging model (see 11.3.7.2, "Source Shape and Opacity" and 11.6.4.4, "Constant Shape and Opacity"). There are two separate alpha constant parameters: one for stroking and one for all other painting operations. A conforming reader shall implicitly reset this parameter to its initial value at the beginning of execution of a transparency group XObject (see 11.6.6, "Transparency Group XObjects"). Initial value: 1.0.

Table 52 – Device-Independent Graphics State Parameters (continued)

Parameter	Type	Value
alpha source	boolean	*(PDF 1.4)* A flag specifying whether the current soft mask and alpha constant parameters shall be interpreted as shape values (**true**) or opacity values (**false**). This flag also governs the interpretation of the **SMask** entry, if any, in an image dictionary (see 8.9.5, "Image Dictionaries"). Initial value: **false**.

Table 53 – Device-Dependent Graphics State Parameters

Parameter	Type	Value
overprint	boolean	*(PDF 1.2)* A flag specifying (on output devices that support the overprint control feature) whether painting in one set of colorants should cause the corresponding areas of other colorants to be erased (**false**) or left unchanged (**true**); see 8.6.7, "Overprint Control". In PDF 1.3, there are two separate overprint parameters: one for stroking and one for all other painting operations. Initial value: **false**.
overprint mode	number	*(PDF 1.3)* A code specifying whether a colour component value of 0 in a **DeviceCMYK** colour space should erase that component (0) or leave it unchanged (1) when overprinting (see 8.6.7, "Overprint Control"). Initial value: 0.
black generation	function or name	*(PDF 1.2)* A function that calculates the level of the black colour component to use when converting *RGB* colours to *CMYK* (see 10.3.4, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: a conforming reader shall initialize this to a suitable device dependent value.
undercolor removal	function or name	*(PDF 1.2)* A function that calculates the reduction in the levels of the cyan, magenta, and yellow colour components to compensate for the amount of black added by black generation (see 10.3.4, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: a conforming reader shall initialize this to a suitable device dependent value.
transfer	function, array, or name	*(PDF 1.2)* A function that adjusts device gray or colour component levels to compensate for nonlinear response in a particular output device (see 10.4, “Transfer Functions“). Initial value: a conforming reader shall initialize this to a suitable device dependent value.
halftone	dictionary, stream, or name	*(PDF 1.2)* A halftone screen for gray and colour rendering, specified as a halftone dictionary or stream (see 10.5, "Halftones"). Initial value: a conforming reader shall initialize this to a suitable device dependent value.
flatness	number	The precision with which curves shall be rendered on the output device (see 10.6.2, "Flatness Tolerance"). The value of this parameter (positive number) gives the maximum error tolerance, measured in output device pixels; smaller numbers give smoother curves at the expense of more computation and memory use. Initial value: 1.0.

Table 53 – Device-Dependent Graphics State Parameters (continued)

Parameter	Type	Value
smoothness	number	*(PDF 1.3)* The precision with which colour gradients are to be rendered on the output device (see 10.6.3, "Smoothness Tolerance"). The value of this parameter (0 to 1.0) gives the maximum error tolerance, expressed as a fraction of the range of each colour component; smaller numbers give smoother colour transitions at the expense of more computation and memory use. Initial value: a conforming reader shall initialize this to a suitable device dependent value.

NOTE 1 Some graphics state parameters are set with specific PDF operators, some are set by including a particular entry in a graphics state parameter dictionary, and some can be specified either way.

EXAMPLE 2 The current line width can be set either with the **w** operator or (in PDF 1.3) with the **LW** entry in a graphics state parameter dictionary, whereas the current colour is set only with specific operators, and the current halftone is set only with a graphics state parameter dictionary.

In general, a conforming reader, when interpreting the operators that set graphics state parameters, shall simply store them unchanged for later use when interpreting the painting operators.However, some parameters have special properties or call for behaviour that a conforming reader shall handle:

- Most parameters shall be of the correct type or have values that fall within a certain range.
- Parameters that are numeric values, such as the current colour, line width, and miter limit, shall be forced into valid range, if necessary. However, they shall not be adjusted to reflect capabilities of the raster output device, such as resolution or number of distinguishable colours. Painting operators perform such adjustments, but the adjusted values shall not be stored back into the graphics state.
- Paths shall be internal objects that shall not be directly represented in PDF.

NOTE 2 As indicated in Table 52 and Table 53, some of the parameters—color space, color, and overprint—have two values, one used for stroking (of paths and text objects) and one for all other painting operations. The two parameter values can be set independently, allowing for operations such as combined filling and stroking of the same path with different colours. Except where noted, a term such as *current colour* should be interpreted to refer to whichever colour parameter applies to the operation being performed. When necessary, the individual colour parameters are distinguished explicitly as the *stroking colour* and the *nonstroking colour*.

8.4.2 Graphics State Stack

A PDF document typically contains many graphical elements that are independent of each other and nested to multiple levels. The *graphics state stack* allows these elements to make local changes to the graphics state without disturbing the graphics state of the surrounding environment. The stack is a LIFO (last in, first out) data structure in which the contents of the graphics state may be saved and later restored using the following operators:

- The **q** operator shall push a copy of the entire graphics state onto the stack.
- The **Q** operator shall restore the entire graphics state to its former value by popping it from the stack.

NOTE These operators can be used to encapsulate a graphical element so that it can modify parameters of the graphics state and later restore them to their previous values.

Occurrences of the **q** and **Q** operators shall be balanced within a given content stream (or within the sequence of streams specified in a page dictionary's **Contents** array).

8.4.3 Details of Graphics State Parameters

8.4.3.1 General

This sub-clause gives details of several of the device-independent graphics state parameters listed in Table 52.

8.4.3.2 Line Width

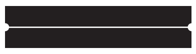
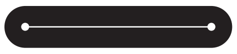
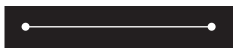
The *line width* parameter specifies the thickness of the line used to stroke a path. It shall be a non-negative number expressed in user space units; stroking a path shall entail painting all points whose perpendicular distance from the path in user space is less than or equal to half the line width. The effect produced in device space depends on the current transformation matrix (CTM) in effect at the time the path is stroked. If the CTM specifies scaling by different factors in the horizontal and vertical dimensions, the thickness of stroked lines in device space shall vary according to their orientation. The actual line width achieved can differ from the requested width by as much as 2 device pixels, depending on the positions of lines with respect to the pixel grid. Automatic stroke adjustment may be used to ensure uniform line width; see 10.6.5, "Automatic Stroke Adjustment".

A line width of 0 shall denote the thinnest line that can be rendered at device resolution: 1 device pixel wide. However, some devices cannot reproduce 1-pixel lines, and on high-resolution devices, they are nearly invisible. Since the results of rendering such zero-width lines are device-dependent, they should not be used.

8.4.3.3 Line Cap Style

The *line cap style* shall specify the shape that shall be used at the ends of open subpaths (and dashes, if any) when they are stroked. Table 54 shows the possible values.

Table 54 – Line Cap Styles

Style	Appearance	Description
0		*Butt cap*. The stroke shall be squared off at the endpoint of the path. There shall be no projection beyond the end of the path.
1		*Round cap*. A semicircular arc with a diameter equal to the line width shall be drawn around the endpoint and shall be filled in.
2		*Projecting square cap*. The stroke shall continue beyond the endpoint of the path for a distance equal to half the line width and shall be squared off.

8.4.3.4 Line Join Style

The *line join style* shall specify the shape to be used at the corners of paths that are stroked. Table 55 shows the possible values. Join styles shall be significant only at points where consecutive segments of a path connect at an angle; segments that meet or intersect fortuitously shall receive no special treatment.

Table 55 – Line Join Styles

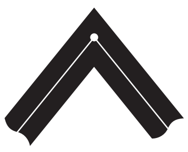
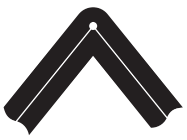
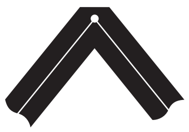
Style	Appearance	Description
0		*Miter join*. The outer edges of the strokes for the two segments shall be extended until they meet at an angle, as in a picture frame. If the segments meet at too sharp an angle (as defined by the miter limit parameter—see 8.4.3.5, "Miter Limit"), a bevel join shall be used instead.

Table 55 – Line Join Styles (continued)

Style	Appearance	Description
1		*Round join*. An arc of a circle with a diameter equal to the line width shall be drawn around the point where the two segments meet, connecting the outer edges of the strokes for the two segments. This pieslice-shaped figure shall be filled in, producing a rounded corner.
2		*Bevel join*. The two segments shall be finished with butt caps (see 8.4.3.3, "Line Cap Style") and the resulting notch beyond the ends of the segments shall be filled with a triangle.

NOTE The definition of round join was changed in PDF 1.5. In rare cases, the implementation of the previous specification could produce unexpected results.

8.4.3.5 Miter Limit

When two line segments meet at a sharp angle and mitered joins have been specified as the line join style, it is possible for the miter to extend far beyond the thickness of the line stroking the path. The *miter limit* shall impose a maximum on the ratio of the miter length to the line width (see Figure 15 in Annex L). When the limit is exceeded, the join is converted from a miter to a bevel.

The ratio of miter length to line width is directly related to the angle *j* between the segments in user space by the following formula:

$$\frac{miterLength}{lineWidth} = \frac{1}{\sin\left(\frac{\varphi}{2}\right)}$$

EXAMPLE A miter limit of 1.414 converts miters to bevels for j less than 90 degrees, a limit of 2.0 converts them for j less than 60 degrees, and a limit of 10.0 converts them for j less than approximately 11.5 degrees.

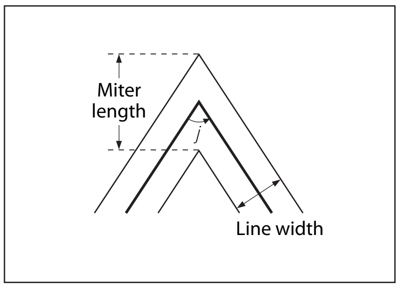


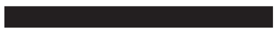
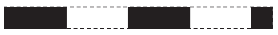
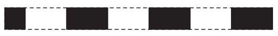
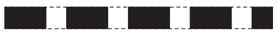
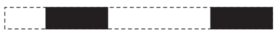
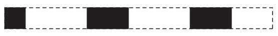
Figure 15 – Miter Length

8.4.3.6 Line Dash Pattern

The *line dash pattern* shall control the pattern of dashes and gaps used to stroke paths. It shall be specified by a *dash array* and a *dash phase*. The dash array's elements shall be numbers that specify the lengths of alternating dashes and gaps; the numbers shall be nonnegative and not all zero. The dash phase shall specify the distance into the dash pattern at which to start the dash. The elements of both the dash array and the dash phase shall be expressed in user space units.

Before beginning to stroke a path, the dash array shall be cycled through, adding up the lengths of dashes and gaps. When the accumulated length equals the value specified by the dash phase, stroking of the path shall begin, and the dash array shall be used cyclically from that point onward. Table 56 shows examples of line dash patterns. As can be seen from the table, an empty dash array and zero phase can be used to restore the dash pattern to a solid line.

Table 56 – Examples of Line Dash Patterns

Dash Array and Phase	Appearance	Description
[ ] 0		No dash; solid, unbroken lines
[3] 0		3 units on, 3 units off, …
[2] 1		1 on, 2 off, 2 on, 2 off, …
[2 1] 0		2 on, 1 off, 2 on, 1 off, …
[3 5] 6		2 off, 3 on, 5 off, 3 on, 5 off, …
[2 3] 11		1 on, 3 off, 2 on, 3 off, 2 on, …

Dashed lines shall wrap around curves and corners just as solid stroked lines do. The ends of each dash shall be treated with the current line cap style, and corners within dashes shall be treated with the current line join style. A stroking operation shall take no measures to coordinate the dash pattern with features of the path; it simply shall dispense dashes and gaps along the path in the pattern defined by the dash array.

When a path consisting of several subpaths is stroked, each subpath shall be treated independently—that is, the dash pattern shall be restarted and the dash phase shall be reapplied to it at the beginning of each subpath.

8.4.4 Graphics State Operators

Table 57 shows the operators that set the values of parameters in the graphics state. (See also the colour operators listed in Table 74 and the text state operators in Table 105.)

Table 57 – Graphics State Operators

Operands	Operator	Description
—	**q**	Save the current graphics state on the graphics state stack (see 8.4.2, "Graphics State Stack").
—	**Q**	Restore the graphics state by removing the most recently saved state from the stack and making it the current state (see 8.4.2, "Graphics State Stack").
a b c d e f	**cm**	Modify the current transformation matrix (CTM) by concatenating the specified matrix (see 8.3.2, "Coordinate Spaces"). Although the operands specify a matrix, they shall be written as six separate numbers, not as an array.
lineWidth	**w**	Set the line width in the graphics state (see 8.4.3.2, "Line Width").
lineCap	**J**	Set the line cap style in the graphics state (see 8.4.3.3, "Line Cap Style").
lineJoin	**j**	Set the line join style in the graphics state (see 8.4.3.4, "Line Join Style").
miterLimit	**M**	Set the miter limit in the graphics state (see 8.4.3.5, "Miter Limit").
dashArray dashPhase	**d**	Set the line dash pattern in the graphics state (see 8.4.3.6, "Line Dash Pattern").
intent	**ri**	*(PDF 1.1)* Set the colour rendering intent in the graphics state (see 8.6.5.8, "Rendering Intents").

Table 57 – Graphics State Operators (continued)

Operands	Operator	Description
flatness	**i**	Set the flatness tolerance in the graphics state (see 10.6.2, "Flatness Tolerance"). *flatness* is a number in the range 0 to 100; a value of 0 shall specify the output device's default flatness tolerance.
dictName	**gs**	*(PDF 1.2)* Set the specified parameters in the graphics state. *dictName* shall be the name of a graphics state parameter dictionary in the **ExtGState** subdictionary of the current resource dictionary (see the next sub-clause).

8.4.5 Graphics State Parameter Dictionaries

While some parameters in the graphics state may be set with individual operators, as shown in Table 57, others may not. The latter may only be set with the generic graphics state operator **gs** *(PDF 1.2).* The operand supplied to this operator shall be the name of a *graphics state parameter dictionary* whose contents specify the values of one or more graphics state parameters. This name shall be looked up in the **ExtGState** subdictionary of the current resource dictionary.

The graphics state parameter dictionary is also used by type 2 patterns, which do not have a content stream in which the graphics state operators could be invoked (see 8.7.4, "Shading Patterns").

Each entry in the parameter dictionary shall specify the value of an individual graphics state parameter, as shown in Table 58. All entries need not be present for every invocation of the **gs** operator; the supplied parameter dictionary may include any combination of parameter entries. The results of **gs** shall be cumulative; parameter values established in previous invocations persist until explicitly overridden.

NOTE Note that some parameters appear in both Table 57 and Table 58; these parameters can be set either with individual graphics state operators or with **gs**. It is expected that any future extensions to the graphics state will be implemented by adding new entries to the graphics state parameter dictionary rather than by introducing new graphics state operators.

Table 58 – Entries in a Graphics State Parameter Dictionary

Key	Type	Description
Type	name	*(Optional)* The type of PDF object that this dictionary describes; shall be **ExtGState** for a graphics state parameter dictionary.
LW	number	*(Optional; PDF 1.3)* The line width (see 8.4.3.2, "Line Width").
LC	integer	*(Optional; PDF 1.3)* The line cap style (see 8.4.3.3, "Line Cap Style").
LJ	integer	*(Optional; PDF 1.3)* The line join style (see 8.4.3.4, "Line Join Style").
ML	number	*(Optional; PDF 1.3)* The miter limit (see 8.4.3.5, "Miter Limit").
D	array	*(Optional; PDF 1.3)* The line dash pattern, expressed as an array of the form [*dashArray dashPhase*], where *dashArray* shall be itself an array and *dashPhase* shall be an integer (see 8.4.3.6, "Line Dash Pattern").
RI	name	*(Optional; PDF 1.3)* The name of the rendering intent (see 8.6.5.8, "Rendering Intents").

Table 58 – Entries in a Graphics State Parameter Dictionary (continued)

Key	Type	Description
OP	boolean	*(Optional)* A flag specifying whether to apply overprint (see 8.6.7, "Overprint Control"). In PDF 1.2 and earlier, there is a single overprint parameter that applies to all painting operations. Beginning with PDF 1.3, there shall be two separate overprint parameters: one for stroking and one for all other painting operations. Specifying an **OP** entry shall set both parameters unless there is also an **op** entry in the same graphics state parameter dictionary, in which case the **OP** entry shall set only the overprint parameter for stroking.
op	boolean	*(Optional; PDF 1.3)* A flag specifying whether to apply overprint (see 8.6.7, "Overprint Control") for painting operations other than stroking. If this entry is absent, the **OP** entry, if any, shall also set this parameter.
OPM	integer	*(Optional; PDF 1.3)* The overprint mode (see 8.6.7, "Overprint Control").
Font	array	*(Optional; PDF 1.3)* An array of the form [*font size*], where *font* shall be an indirect reference to a font dictionary and *size* shall be a number expressed in text space units. These two objects correspond to the operands of the **Tf** operator (see 9.3, "Text State Parameters and Operators"); however, the first operand shall be an indirect object reference instead of a resource name.
BG	function	*(Optional)* The black-generation function, which maps the interval [0.0 1.0] to the interval [0.0 1.0] (see 10.3.4, "Conversion from DeviceRGB to DeviceCMYK").
BG2	function or name	*(Optional; PDF 1.3)* Same as **BG** except that the value may also be the name Default, denoting the black-generation function that was in effect at the start of the page. If both **BG** and **BG2** are present in the same graphics state parameter dictionary, **BG2** shall take precedence.
UCR	function	*(Optional)* The undercolor-removal function, which maps the interval [0.0 1.0] to the interval [−1.0 1.0] (see 10.3.4, "Conversion from DeviceRGB to DeviceCMYK").
UCR2	function or name	*(Optional; PDF 1.3)* Same as **UCR** except that the value may also be the name Default, denoting the undercolor-removal function that was in effect at the start of the page. If both **UCR** and **UCR2** are present in the same graphics state parameter dictionary, **UCR2** shall take precedence.
TR	function, array, or name	*(Optional)* The transfer function, which maps the interval [0.0 1.0] to the interval [0.0 1.0] (see 10.4, "Transfer Functions"). The value shall be either a single function (which applies to all process colorants) or an array of four functions (which apply to the process colorants individually). The name **Identity** may be used to represent the identity function.
TR2	function, array, or name	*(Optional; PDF 1.3)* Same as **TR** except that the value may also be the name Default, denoting the transfer function that was in effect at the start of the page. If both **TR** and **TR2** are present in the same graphics state parameter dictionary, **TR2** shall take precedence.
HT	dictionary, stream, or name	*(Optional)* The halftone dictionary or stream (see 10.5, "Halftones") or the name Default, denoting the halftone that was in effect at the start of the page.
FL	number	*(Optional; PDF 1.3)* The flatness tolerance (see 10.6.2, "Flatness Tolerance").
SM	number	*(Optional; PDF 1.3)* The smoothness tolerance (see 10.6.3, "Smoothness Tolerance").

Table 58 – Entries in a Graphics State Parameter Dictionary (continued)

Key	Type	Description
SA	boolean	*(Optional)* A flag specifying whether to apply automatic stroke adjustment (see 10.6.5, "Automatic Stroke Adjustment").
BM	name or array	*(Optional; PDF 1.4)* The current blend mode to be used in the transparent imaging model (see 11.3.5, "Blend Mode" and 11.6.3, "Specifying Blending Colour Space and Blend Mode").
SMask	dictionary or name	*(Optional; PDF 1.4)* The current soft mask, specifying the mask shape or mask opacity values that shall be used in the transparent imaging model (see 11.3.7.2, "Source Shape and Opacity" and 11.6.4.3, "Mask Shape and Opacity"). Although the current soft mask is sometimes referred to as a "soft clip," altering it with the ***gs*** operator completely replaces the old value with the new one, rather than intersecting the two as is done with the current clipping path parameter (see 8.5.4, "Clipping Path Operators").
CA	number	*(Optional; PDF 1.4)* The current stroking alpha constant, specifying the constant shape or constant opacity value that shall be used for stroking operations in the transparent imaging model (see 11.3.7.2, "Source Shape and Opacity" and 11.6.4.4, "Constant Shape and Opacity").
ca	number	*(Optional; PDF 1.4)* Same as **CA**, but for nonstroking operations.
AIS	boolean	*(Optional; PDF 1.4)* The alpha source flag ("alpha is shape"), specifying whether the current soft mask and alpha constant shall be interpreted as shape values (**true**) or opacity values (**false**).
TK	boolean	*(Optional; PDF 1.4)* The text knockout flag, shall determine the behaviour of overlapping glyphs within a text object in the transparent imaging model (see 9.3.8, "Text Knockout").

EXAMPLE The following shows two graphics state parameter dictionaries. In the first, automatic stroke adjustment is turned on, and the dictionary includes a transfer function that inverts its value, $f(x) = 1 - x$. In the second, overprint is turned off, and the dictionary includes a parabolic transfer function, $f(x) = (2x - 1)^2$, with a sample of 21 values. The domain of the transfer function, [0.0 1.0], is mapped to [0 20], and the range of the sample values, [0 255], is mapped to the range of the transfer function, [0.0 1.0].

```
10  0  obj                                 % Page object
    << /Type  /Page
        /Parent  5 0 R
        /Resources  20 0 R
        /Contents  40 0 R
    >>
endobj

20  0  obj                                 % Resource dictionary for page
    <<  /ProcSet  [/PDF  /Text]
        /Font  <<  /F1  25 0 R  >>
        /ExtGState  <<  /GS1  30 0 R
                        /GS2  35 0 R
                    >>
    >>
endobj

30  0  obj                                 % First graphics state parameter dictionary
    <<  /Type  /ExtGState
        /SA  true
        /TR  31 0 R
    >>
endobj
```

```
31  0  obj                            % First transfer function
    <<  /FunctionType  0
        /Domain  [0.0  1.0]
        /Range  [0.0  1.0]
        /Size  2
        /BitsPerSample  8
        /Length  7
        /Filter  /ASCIIHexDecode
    >>

stream
01  00  >
endstream
endobj

35  0  obj                            % Second graphics state parameter dictionary
    <<  /Type  /ExtGState
        /OP  false
        /TR  36 0 R
    >>
endobj


36  0  obj                            % Second transfer function
    <<  /FunctionType  0
        /Domain  [0.0  1.0]
        /Range  [0.0  1.0]
        /Size  21
        /BitsPerSample  8
        /Length  63
        /Filter  /ASCIIHexDecode
    >>

stream
FF  CE  A3  7C  5B  3F  28  16  0A  02  00  02  0A  16  28  3F  5B  7C  A3  CE  FF  >
endstream
endobj
```

8.5 Path Construction and Painting

8.5.1 General

Paths define shapes, trajectories, and regions of all sorts. They shall be used to draw lines, define the shapes of filled areas, and specify boundaries for clipping other graphics. The graphics state shall include a *current clipping path* that shall define the clipping boundary for the current page. At the beginning of each page, the clipping path shall be initialized to include the entire page.

A path shall be composed of straight and curved line segments, which may connect to one another or may be disconnected. A pair of segments shall be said to *connect* only if they are defined consecutively, with the second segment starting where the first one ends. Thus, the order in which the segments of a path are defined shall be significant. Nonconsecutive segments that meet or intersect fortuitously shall not be considered to connect.

NOTE A path is made up of one or more disconnected *subpaths*, each comprising a sequence of connected segments. The topology of the path is unrestricted: it may be concave or convex, may contain multiple subpaths representing disjoint areas, and may intersect itself in arbitrary ways.

The **h** operator explicitly shall connect the end of a subpath back to its starting point; such a subpath is said to be *closed*. A subpath that has not been explicitly closed is said to be *open*.

As discussed in 8.2, "Graphics Objects", a path object is defined by a sequence of operators to construct the path, followed by one or more operators to paint the path or to use it as a clipping boundary. PDF path operators fall into three categories:

- *Path construction operators* (8.5.2, "Path Construction Operators") define the geometry of a path. A path is constructed by sequentially applying one or more of these operators.
- *Path-painting operators* (8.5.3, "Path-Painting Operators") end a path object, usually causing the object to be painted on the current page in any of a variety of ways.
- *Clipping path operators* (8.5.4, "Clipping Path Operators"), invoked immediately before a path-painting operator, cause the path object also to be used for clipping of subsequent graphics objects.

8.5.2 Path Construction Operators

8.5.2.1 General

A page description shall begin with an empty path and shall build up its definition by invoking one or more path construction operators to add segments to it. The path construction operators may be invoked in any sequence, but the first one invoked shall be **m** or **re** to begin a new subpath. The path definition may conclude with the application of a path-painting operator such as **S**, **f**, or **b** (see 8.5.3, "Path-Painting Operators"); this operator may optionally be preceded by one of the clipping path operators **W** or **W*** (8.5.4, "Clipping Path Operators").

NOTE Note that the path construction operators do not place any marks on the page; only the painting operators do that. A path definition is not complete until a path-painting operator has been applied to it.

The path currently under construction is called the *current path*. In PDF (unlike PostScript), the current path is *not* part of the graphics state and is *not* saved and restored along with the other graphics state parameters. PDF paths shall be strictly internal objects with no explicit representation. After the current path has been painted, it shall become no longer defined; there is then no current path until a new one is begun with the **m** or **re** operator.

The trailing endpoint of the segment most recently added to the current path is referred to as the *current point.* If the current path is empty, the current point shall be undefined. Most operators that add a segment to the current path start at the current point; if the current point is undefined, an error shall be generated.

Table 59 shows the path construction operators. All operands shall be numbers denoting coordinates in user space.

Table 59 – Path Construction Operators

Operands	Operator	Description
x y	**m**	Begin a new subpath by moving the current point to coordinates (x, y), omitting any connecting line segment. If the previous path construction operator in the current path was also **m**, the new **m** overrides it; no vestige of the previous **m** operation remains in the path.
x y	**l** (lowercase **L**)	Append a straight line segment from the current point to the point (x, y). The new current point shall be (x, y).
x_1 y_1 x_2 y_2 x_3 y_3	**c**	Append a cubic Bézier curve to the current path. The curve shall extend from the current point to the point (x_3, y_3), using (x_1, y_1) and (x_2, y_2) as the Bézier control points (see 8.5.2.2, "Cubic Bézier Curves"). The new current point shall be (x_3, y_3).

Table 59 – Path Construction Operators (continued)

Operands	Operator	Description
x_2 y_2 x_3 y_3	**v**	Append a cubic Bézier curve to the current path. The curve shall extend from the current point to the point (x_3, y_3), using the current point and (x_2, y_2) as the Bézier control points (see 8.5.2.2, "Cubic Bézier Curves"). The new current point shall be (x_3, y_3).
x_1 y_1 x_3 y_3	**y**	Append a cubic Bézier curve to the current path. The curve shall extend from the current point to the point (x_3, y_3), using (x_1, y_1) and (x_3, y_3) as the Bézier control points (see 8.5.2.2, "Cubic Bézier Curves"). The new current point shall be (x_3, y_3).
—	**h**	Close the current subpath by appending a straight line segment from the current point to the starting point of the subpath. If the current subpath is already closed, **h** shall do nothing. This operator terminates the current subpath. Appending another segment to the current path shall begin a new subpath, even if the new segment begins at the endpoint reached by the **h** operation.
x y width height	**re**	Append a rectangle to the current path as a complete subpath, with lower-left corner (*x*, *y*) and dimensions *width* and *height* in user space. The operation *x y width height* re is equivalent to *x y* m (*x* + *width*) *y* l (*x* + *width*) (*y* + *height*) l *x* (*y* + *height*) l h

8.5.2.2 Cubic Bézier Curves

Curved path segments shall be specified as *cubic Bézier curves*. Such curves shall be defined by four points: the two endpoints (the current point P_0 and the final point P_3) and two *control points* P_1 and P_2. Given the coordinates of the four points, the curve shall be generated by varying the parameter *t* from 0.0 to 1.0 in the following equation:

$$R(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

When $t = 0.0$, the value of the function $R(t)$ coincides with the current point P_0; when $t = 1.0$, $R(t)$ coincides with the final point P_3. Intermediate values of *t* generate intermediate points along the curve. The curve does not, in general, pass through the two control points P_1 and P_2.

NOTE 1 Cubic Bézier curves have two useful properties:

The curve can be very quickly split into smaller pieces for rapid rendering.

The curve is contained within the convex hull of the four points defining the curve, most easily visualized as the polygon obtained by stretching a rubber band around the outside of the four points. This property allows rapid testing of whether the curve lies completely outside the visible region, and hence does not have to be rendered.

NOTE 2 The Bibliography lists several books that describe cubic Bézier curves in more depth.

The most general PDF operator for constructing curved path segments is the **c** operator, which specifies the coordinates of points P_1, P_2, and P_3 explicitly, as shown in Figure 16 in Annex L. (The starting point, P_0, is defined implicitly by the current point.)

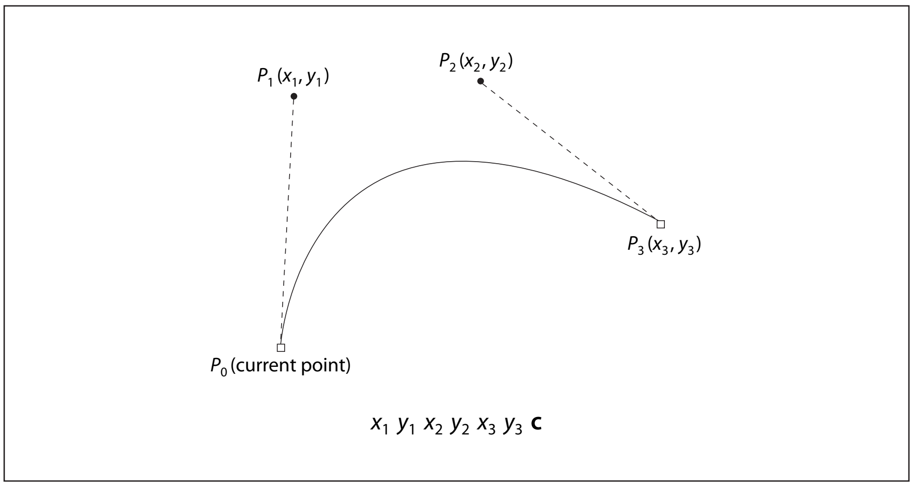


Figure 16 – Cubic Bézier Curve Generated by the c Operator

Two more operators, **v** and **y**, each specify one of the two control points implicitly (see Figure 17 in Annex L). In both of these cases, one control point and the final point of the curve shall be supplied as operands; the other control point shall be implied:

- For the **v** operator, the first control point shall coincide with initial point of the curve.
- For the **y** operator, the second control point shall coincide with final point of the curve.

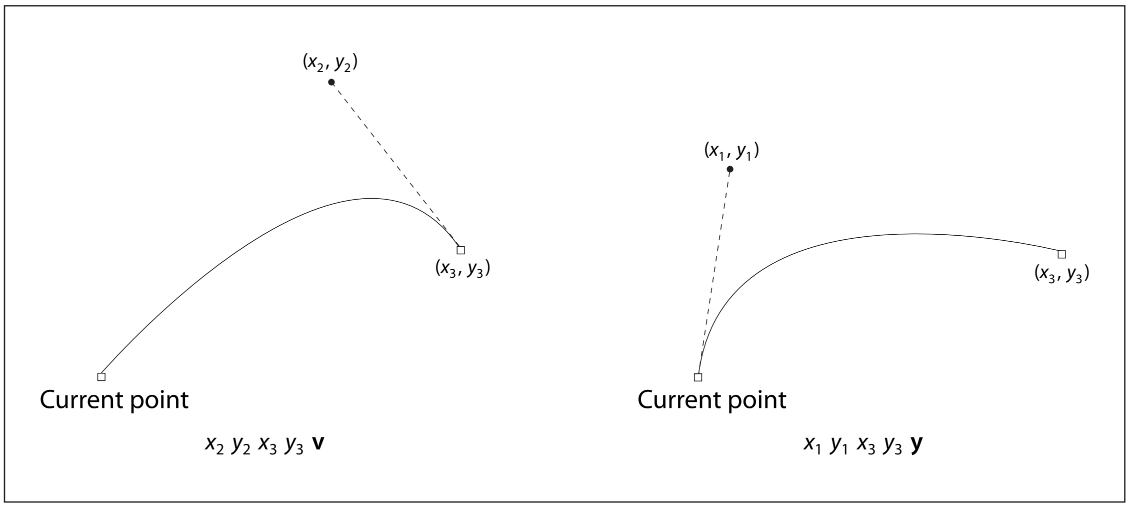


Figure 17 – Cubic Bézier Curves Generated by the v and y Operators

8.5.3 Path-Painting Operators

8.5.3.1 General

The path-painting operators end a path object, causing it to be painted on the current page in the manner that the operator specifies. The principal path-painting operators shall be **S** (for *stroking*) and **f** (for *filling*). Variants of these operators combine stroking and filling in a single operation or apply different rules for determining the area to be filled. Table 60 lists all the path-painting operators.

Table 60 – Path-Painting Operators

Operands	Operator	Description
—	**S**	Stroke the path.
—	**s**	Close and stroke the path. This operator shall have the same effect as the sequence h S.
—	**f**	Fill the path, using the nonzero winding number rule to determine the region to fill (see 8.5.3.3.2, "Nonzero Winding Number Rule"). Any subpaths that are open shall be implicitly closed before being filled.
—	**F**	Equivalent to **f**; included only for compatibility. Although PDF reader applications shall be able to accept this operator, PDF writer applications should use **f** instead.
—	**f***	Fill the path, using the even-odd rule to determine the region to fill (see 8.5.3.3.3, "Even-Odd Rule").
—	**B**	Fill and then stroke the path, using the nonzero winding number rule to determine the region to fill. This operator shall produce the same result as constructing two identical path objects, painting the first with **f** and the second with **S**. NOTE The filling and stroking portions of the operation consult different values of several graphics state parameters, such as the current colour. See also 11.7.4.4, "Special Path-Painting Considerations".
—	**B***	Fill and then stroke the path, using the even-odd rule to determine the region to fill. This operator shall produce the same result as **B**, except that the path is filled as if with **f*** instead of **f**. See also 11.7.4.4, "Special Path-Painting Considerations".
—	**b**	Close, fill, and then stroke the path, using the nonzero winding number rule to determine the region to fill. This operator shall have the same effect as the sequence h B. See also 11.7.4.4, "Special Path-Painting Considerations".
—	**b***	Close, fill, and then stroke the path, using the even-odd rule to determine the region to fill. This operator shall have the same effect as the sequence h B*. See also 11.7.4.4, "Special Path-Painting Considerations".
—	**n**	End the path object without filling or stroking it. This operator shall be a path-painting no-op, used primarily for the side effect of changing the current clipping path (see 8.5.4, "Clipping Path Operators").

8.5.3.2 Stroking

The **S** operator shall paint a line along the current path. The stroked line shall follow each straight or curved segment in the path, centred on the segment with sides parallel to it. Each of the path's subpaths shall be treated separately.

The results of the **S** operator shall depend on the current settings of various parameters in the graphics state (see 8.4, "Graphics State", for further information on these parameters):

- The width of the stroked line shall be determined by the current line width parameter (8.4.3.2, "Line Width").
- The colour or pattern of the line shall be determined by the current colour and colour space for stroking operations.

- The line may be painted either solid or with a dash pattern, as specified by the current line dash pattern (see 8.4.3.6, "Line Dash Pattern").

- If a subpath is open, the unconnected ends shall be treated according to the current line cap style, which may be butt, rounded, or square (see 8.4.3.3, "Line Cap Style").

- Wherever two consecutive segments are connected, the joint between them shall be treated according to the current *line join* style, which may be mitered, rounded, or beveled (see 8.4.3.4, "Line Join Style"). Mitered joins shall be subject to the current miter limit (see 8.4.3.5, "Miter Limit").

Points at which unconnected segments happen to meet or intersect receive no special treatment. In particular, using an explicit **l** operator to give the appearance of closing a subpath, rather than using **h**, may result in a messy corner, because line caps are applied instead of a line join.

- The *stroke adjustment* parameter *(PDF 1.2)* specifies that coordinates and line widths be adjusted automatically to produce strokes of uniform thickness despite rasterization effects (see 10.6.5, "Automatic Stroke Adjustment").

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), the **S** operator shall paint it only if round line caps have been specified, producing a filled circle centered at the single point. If butt or projecting square line caps have been specified, **S** shall produce no output, because the orientation of the caps would be indeterminate. This rule shall apply only to zero-length subpaths of the path being stroked, and not to zero-length dashes in a dash pattern. In the latter case, the line caps shall always be painted, since their orientation is determined by the direction of the underlying path. A single-point open subpath (specified by a trailing **m** operator) shall produce no output.

8.5.3.3 Filling

8.5.3.3.1 General

The **f** operator shall use the current nonstroking colour to paint the entire region enclosed by the current path. If the path consists of several disconnected subpaths, **f** shall paint the insides of all subpaths, considered together. Any subpaths that are open shall be implicitly closed before being filled.

If a subpath is degenerate (consists of a single-point closed path or of two or more points at the same coordinates), **f** shall paint the single device pixel lying under that point; the result is device-dependent and not generally useful. A single-point open subpath (specified by a trailing **m** operator) shall produce no output.

For a simple path, it is intuitively clear what region lies inside. However, for a more complex path, it is not always obvious which points lie inside the path. For more detailed information, see 10.6.4, “Scan Conversion Rules“.

EXAMPLE A path that intersects itself or has one subpath that encloses another.

The path machinery shall use one of two rules for determining which points lie inside a path: the nonzero winding number rule and the even-odd rule, both discussed in detail below. The nonzero winding number rule is more versatile than the even-odd rule and shall be the standard rule the **f** operator uses. Similarly, the **W** operator shall use this rule to determine the inside of the current clipping path. The even-odd rule is occasionally useful for special effects or for compatibility with other graphics systems; the **f*** and **W*** operators invoke this rule.

8.5.3.3.2 Nonzero Winding Number Rule

The *nonzero winding number rule* determines whether a given point is inside a path by conceptually drawing a ray from that point to infinity in any direction and then examining the places where a segment of the path crosses the ray. Starting with a count of 0, the rule adds 1 each time a path segment crosses the ray from left to right and subtracts 1 each time a segment crosses from right to left. After counting all the crossings, if the result is 0, the point is outside the path; otherwise, it is inside.

The method just described does not specify what to do if a path segment coincides with or is tangent to the chosen ray. Since the direction of the ray is arbitrary, the rule simply chooses a ray that does not encounter such problem intersections.

For simple convex paths, the nonzero winding number rule defines the inside and outside as one would intuitively expect. The more interesting cases are those involving complex or self-intersecting paths like the ones shown in Figure 18 in Annex L. For a path consisting of a five-pointed star, drawn with five connected straight line segments intersecting each other, the rule considers the inside to be the entire area enclosed by the star, including the pentagon in the centre. For a path composed of two concentric circles, the areas enclosed by both circles are considered to be inside, provided that both are drawn in the same direction. If the circles are drawn in opposite directions, only the doughnut shape between them is inside, according to the rule; the doughnut hole is outside.

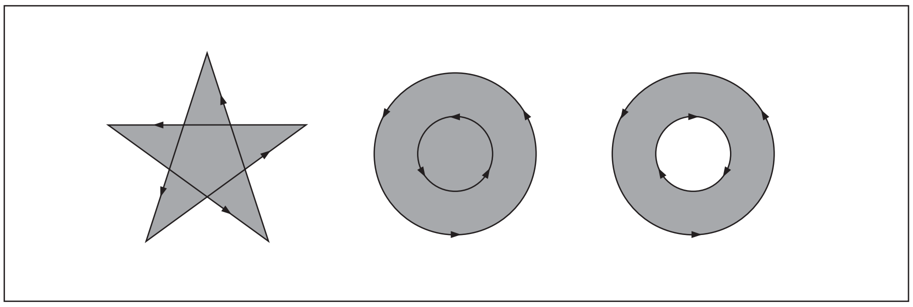


Figure 18 – Nonzero Winding Number Rule

8.5.3.3.3 Even-Odd Rule

An alternative to the nonzero winding number rule is the *even-odd rule*. This rule determines whether a point is inside a path by drawing a ray from that point in any direction and simply counting the number of path segments that cross the ray, regardless of direction. If this number is odd, the point is inside; if even, the point is outside. This yields the same results as the nonzero winding number rule for paths with simple shapes, but produces different results for more complex shapes.

Figure 19 shows the effects of applying the even-odd rule to complex paths. For the five-pointed star, the rule considers the triangular points to be inside the path, but not the pentagon in the centre. For the two concentric circles, only the doughnut shape between the two circles is considered inside, regardless of the directions in which the circles are drawn.

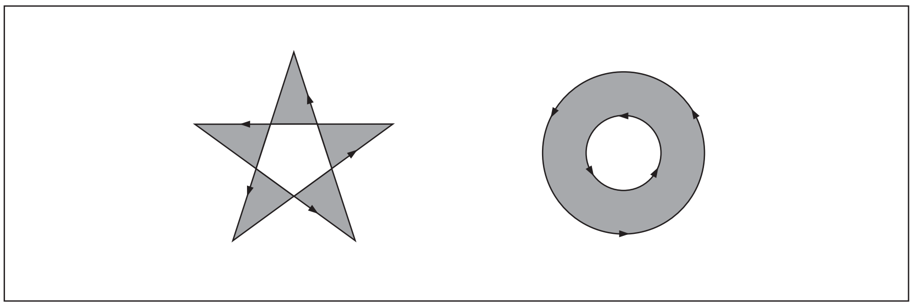


Figure 19 – Even-Odd Rule

8.5.4 Clipping Path Operators

The graphics state shall contain a *current clipping path* that limits the regions of the page affected by painting operators. The closed subpaths of this path shall define the area that can be painted. Marks falling inside this area shall be applied to the page; those falling outside it shall not be. Sub-clause 8.5.3.3, "Filling" discusses precisely what shall be considered to be inside a path.

In the context of the transparent imaging model *(PDF 1.4)*, the current clipping path constrains an object's shape (see 11.2, "Overview of Transparency"). The effective shape is the intersection of the object's intrinsic shape with the clipping path; the source shape value shall be 0.0 outside this intersection. Similarly, the shape of a transparency group (defined as the union of the shapes of its constituent objects) shall be influenced both by the clipping path in effect when each of the objects is painted and by the one in effect at the time the group's results are painted onto its backdrop.

The initial clipping path shall include the entire page. A clipping path operator (**W** or **W***, shown in Table 61) may appear after the last path construction operator and before the path-painting operator that terminates a path object. Although the clipping path operator appears before the painting operator, it shall not alter the clipping path at the point where it appears. Rather, it shall modify the effect of the succeeding painting operator. *After* the path has been painted, the clipping path in the graphics state shall be set to the intersection of the current clipping path and the newly constructed path.

Table 61 – Clipping Path Operators

Operands	Operator	Description
—	**W**	Modify the current clipping path by intersecting it with the current path, using the nonzero winding number rule to determine which regions lie inside the clipping path.
—	**W***	Modify the current clipping path by intersecting it with the current path, using the even-odd rule to determine which regions lie inside the clipping path.

NOTE 1 In addition to path objects, text objects may also be used for clipping; see 9.3.6, "Text Rendering Mode".

The **n** operator (see Table 60) is a no-op path-painting operator; it shall cause no marks to be placed on the page, but can be used with a clipping path operator to establish a new clipping path. That is, after a path has been constructed, the sequence W n shall intersect that path with the current clipping path and shall establish a new clipping path.

NOTE 2 There is no way to enlarge the current clipping path or to set a new clipping path without reference to the current one. However, since the clipping path is part of the graphics state, its effect can be localized to specific graphics objects by enclosing the modification of the clipping path and the painting of those objects between a pair of **q** and **Q** operators (see 8.4.2, "Graphics State Stack"). Execution of the **Q** operator causes the clipping path to revert to the value that was saved by the **q** operator before the clipping path was modified.

8.6 Colour Spaces

8.6.1 General

PDF includes facilities for specifying the colours of graphics objects to be painted on the current page. The colour facilities are divided into two parts:

- *Colour specification*. A conforming writer may specify abstract colours in a device-independent way. Colours may be described in any of a variety of colour systems, or *colour spaces*. Some colour spaces are related to device colour representation (grayscale, *RGB*, *CMYK*), others to human visual perception (CIE-based). Certain special features are also modelled as colour spaces: patterns, colour mapping, separations, and high-fidelity and multitone colour.

- *Colour rendering*. A conforming reader shall reproduce colours on the raster output device by a multiple-step process that includes some combination of colour conversion, gamma correction, halftoning, and scan conversion. Some aspects of this process use information that is specified in PDF. However, unlike the facilities for colour specification, the colour-rendering facilities are device-dependent and should not be included in a page description.

Figure 20 and Figure 21 illustrate the division between PDF's (device-independent) colour specification and (device-dependent) colour-rendering facilities. This sub-clause describes the colour specification features, covering everything that PDF documents need to specify colours. The facilities for controlling colour rendering are described in clause 10, "Rendering"; a conforming writer should use these facilities only to configure or calibrate an output device or to achieve special device-dependent effects.

8.6.2 Colour Values

As described in 8.5.3, "Path-Painting Operators", marks placed on the page by operators such as **f** and **S** shall have a colour that is determined by the *current colour* parameter of the graphics state. A colour value consists of one or more *colour components*, which are usually numbers. A gray level shall be specified by a single number ranging from 0.0 (black) to 1.0 (white). Full colour values may be specified in any of several ways; a common method uses three numeric values to specify red, green, and blue components.

Colour values shall be interpreted according to the *current colour space*, another parameter of the graphics state. A PDF content stream first selects a colour space by invoking the **CS** operator (for the stroking colour) or the **cs** operator (for the nonstroking colour). It then selects colour values within that colour space with the **SC** operator (stroking) or the **sc** operator (nonstroking). There are also convenience operators—**G**, **g**, **RG**, **rg**, **K**, and **k**—that select both a colour space and a colour value within it in a single step. Table 74 lists all the colour-setting operators.

Sampled images (see 8.9, "Images") specify the colour values of individual samples with respect to a colour space designated by the image object itself. While these values are independent of the current colour space and colour parameters in the graphics state, all later stages of colour processing shall treat them in exactly the same way as colour values specified with the **SC** or **sc** operator.

8.6.3 Colour Space Families

Colour spaces are classified into *colour space families*. Spaces within a family share the same general characteristics; they shall be distinguished by parameter values supplied at the time the space is specified. The families fall into three broad categories:

- *Device colour spaces* directly specify colours or shades of gray that the output device shall produce. They provide a variety of colour specification methods, including grayscale, *RGB* (red-green-blue), and *CMYK* (cyan-magenta-yellow-black), corresponding to the colour space families **DeviceGray**, **DeviceRGB**, and **DeviceCMYK**. Since each of these families consists of just a single colour space with no parameters, they may be referred to as the **DeviceGray**, **DeviceRGB**, and **DeviceCMYK** colour spaces.

- *CIE-based colour spaces* shall be based on an international standard for colour specification created by the Commission Internationale de l'Éclairage (International Commission on Illumination). These spaces specify colours in a way that is independent of the characteristics of any particular output device. Colour space families in this category include **CalGray**, **CalRGB**, **Lab**, and **ICCBased**. Individual colour spaces within these families shall be specified by means of dictionaries containing the parameter values needed to define the space.

- *Special colour spaces* add features or properties to an underlying colour space. They include facilities for patterns, colour mapping, separations, and high-fidelity and multitone colour. The corresponding colour space families are **Pattern**, **Indexed**, **Separation**, and **DeviceN**. Individual colour spaces within these families shall be specified by means of additional parameters.

Table 62 summarizes the colour space families in PDF.

Table 62 – Colour Space Families

Device	CIE-based	Special
DeviceGray *(PDF 1.1)*	**CalGray** *(PDF 1.1)*	**Indexed** *(PDF 1.1)*
DeviceRGB *(PDF 1.1)*	**CalRGB** *(PDF 1.1)*	**Pattern** *(PDF 1.2)*
DeviceCMYK *(PDF 1.1)*	**Lab** *(PDF 1.1)*	**Separation** *(PDF 1.2)*
	ICCBased *(PDF 1.3)*	**DeviceN** *(PDF 1.3)*

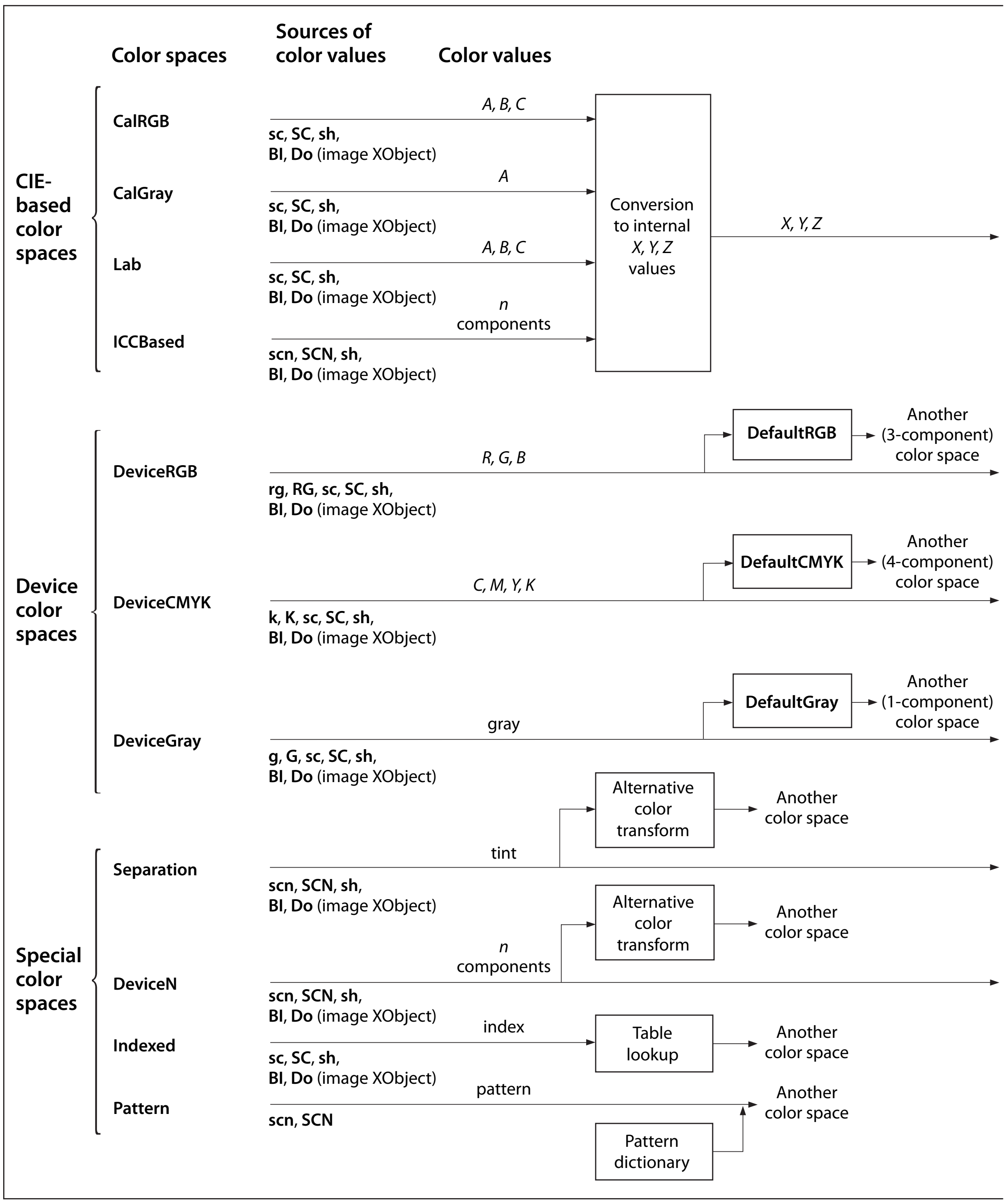

Color spaces
Sources of color values
Color values
CIE-based color spaces
CalRGB
CalGray
Lab
ICCBased
A, B, C
sc, SC, sh, BI, Do (image XObject)
A
sc, SC, sh, BI, Do (image XObject)
A, B, C
sc, SC, sh, BI, Do (image XObject)
n components
scn, SCN, sh, BI, Do (image XObject)
Conversion to internal X, Y, Z values
X, Y, Z
Device color spaces
DeviceRGB
DeviceCMYK
DeviceGray
R, G, B
rg, RG, sc, SC, sh, BI, Do (image XObject)
DefaultRGB
Another (3-component) color space
C, M, Y, K
k, K, sc, SC, sh, BI, Do (image XObject)
DefaultCMYK
Another (4-component) color space
gray
g, G, sc, SC, sh, BI, Do (image XObject)
DefaultGray
Another (1-component) color space
Special color spaces
Separation
DeviceN
Indexed
Pattern
tint
scn, SCN, sh, BI, Do (image XObject)
Alternative color transform
Another color space
n components
scn, SCN, sh, BI, Do (image XObject)
Alternative color transform
Another color space
index
sc, SC, sh, BI, Do (image XObject)
Table lookup
Another color space
pattern
scn, SCN
Pattern dictionary
Another color space

Figure 20 – Colour Specification

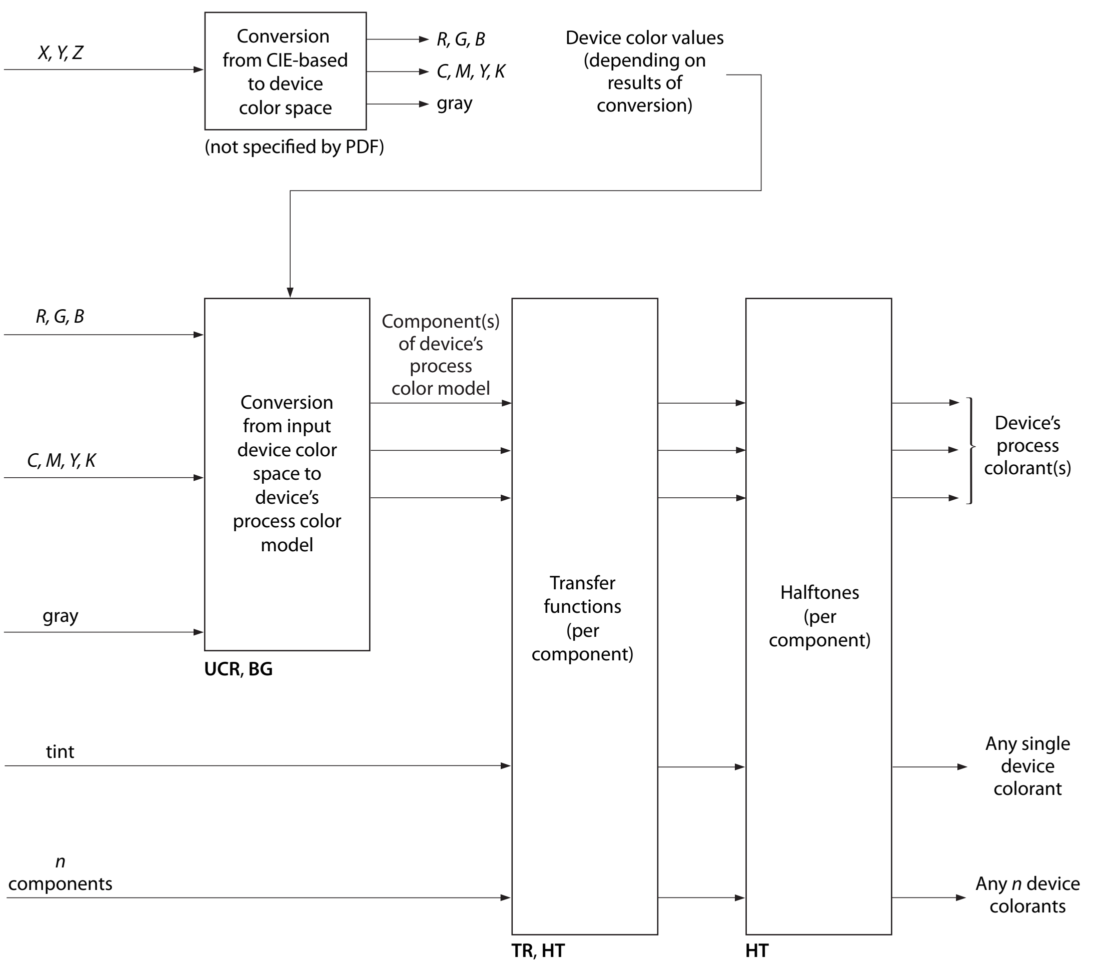

X, Y, Z
Conversion from CIE-based to device color space
(not specified by PDF)
R, G, B
C, M, Y, K
gray
Device color values (depending on results of conversion)
R, G, B
C, M, Y, K
gray
Conversion from input device color space to device's process color model
UCR, BG
Component(s) of device's process color model
tint
n components
Transfer functions (per component)
TR, HT
Halftones (per component)
HT
Device's process colorant(s)
Any single device colorant
Any n device colorants

Figure 21 – Colour Rendering

A colour space shall be defined by an array object whose first element is a name object identifying the colour space family. The remaining array elements, if any, are parameters that further characterize the colour space; their number and types vary according to the particular family. For families that do not require parameters, the colour space may be specified simply by the family name itself instead of an array.

A colour space shall be specified in one of two ways:

- Within a content stream, the **CS** or **cs** operator establishes the current colour space parameter in the graphics state. The operand shall always be name object, which either identifies one of the colour spaces that need no additional parameters (**DeviceGray**, **DeviceRGB**, **DeviceCMYK**, or some cases of **Pattern**) or shall be used as a key in the **ColorSpace** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). In the latter case, the value of the dictionary entry in turn shall be a colour space array or name. A colour space array shall never be inline within a content stream.

- Outside a content stream, certain objects, such as image XObjects, shall specify a colour space as an explicit parameter, often associated with the key **ColorSpace**. In this case, the colour space array or name shall always be defined directly as a PDF object, not by an entry in the **ColorSpace** resource subdictionary. This convention also applies when colour spaces are defined in terms of other colour spaces.

The following operators shall set the current colour space and current colour parameters in the graphics state:

- **CS** shall set the stroking colour space; **cs** shall set the nonstroking colour space.
- **SC** and **SCN** shall set the stroking colour; **sc** and **scn** shall set the nonstroking colour. Depending on the colour space, these operators shall have one or more operands, each specifying one component of the colour value.
- **G**, **RG**, and **K** shall set the stroking colour space implicitly and the stroking colour as specified by the operands; **g**, **rg**, and **k** do the same for the nonstroking colour space and colour.

8.6.4 Device Colour Spaces

8.6.4.1 General

The device colour spaces enable a page description to specify colour values that are directly related to their representation on an output device. Colour values in these spaces map directly (or by simple conversions) to the application of device colorants, such as quantities of ink or intensities of display phosphors. This enables a conforming writer to control colours precisely for a particular device, but the results might not be consistent from one device to another.

Output devices form colours either by adding light sources together or by subtracting light from an illuminating source. Computer displays and film recorders typically add colours; printing inks typically subtract them. These two ways of forming colours give rise to two complementary methods of colour specification, called *additive* and *subtractive* colour (see Figure L.1 in Annex L). The most widely used forms of these two types of colour specification are known as *RGB* and *CMYK*, respectively, for the names of the primary colours on which they are based. They correspond to the following device colour spaces:

- **DeviceGray** controls the intensity of achromatic light, on a scale from black to white.
- **DeviceRGB** controls the intensities of red, green, and blue light, the three additive primary colours used in displays.
- **DeviceCMYK** controls the concentrations of cyan, magenta, yellow, and black inks, the four subtractive process colours used in printing.

NOTE Although the notion of explicit colour spaces is a PDF 1.1 feature, the operators for specifying colours in the device colour spaces—**G**, **g**, **RG**, **rg**, **K**, and **k**—are available in all versions of PDF. Beginning with PDF 1.2, colours specified in device colour spaces can optionally be remapped systematically into other colour spaces; see 8.6.5.6, "Default Colour Spaces".

In the transparent imaging model *(PDF 1.4)*, the use of device colour spaces is subject to special treatment within a transparency group whose group colour space is CIE-based (see 11.4, "Transparency Groups" and 11.6.6, "Transparency Group XObjects"). In particular, the device colour space operators should be used only if device colour spaces have been remapped to CIE-based spaces by means of the default colour space mechanism. Otherwise, the results are implementation-dependent and unpredictable.

8.6.4.2 DeviceGray Colour Space

Black, white, and intermediate shades of gray are special cases of full colour. A grayscale value shall be represented by a single number in the range 0.0 to 1.0, where 0.0 corresponds to black, 1.0 to white, and intermediate values to different gray levels.

EXAMPLE This example shows alternative ways to select the **DeviceGray** colour space and a specific gray level within that space for stroking operations.

```
/DeviceGray CS              % Set DeviceGray colour space
gray SC                     % Set gray level
gray G                      % Set both in one operation
```

The **CS** and **SC** operators shall select the current stroking colour space and current stroking colour separately; **G** shall set them in combination. (The **cs**, **sc**, and **g** operators shall perform the same functions for nonstroking operations.) Setting either current colour space to **DeviceGray** shall initialize the corresponding current colour to 0.0.

8.6.4.3 DeviceRGB Colour Space

Colours in the **DeviceRGB** colour space shall be specified according to the additive *RGB* (red-green-blue) colour model, in which colour values shall be defined by three components representing the intensities of the additive primary colorants red, green, and blue. Each component shall be specified by a number in the range 0.0 to 1.0, where 0.0 shall denote the complete absence of a primary component and 1.0 shall denote maximum intensity.

EXAMPLE This example shows alternative ways to select the **DeviceRGB** colour space and a specific colour within that space for stroking operations.

```
/DeviceRGB CS               % Set DeviceRGB colour space
red green blue SC           % Set colour
red green blue RG           % Set both in one operation
```

The **CS** and **SC** operators shall select the current stroking colour space and current stroking colour separately; **RG** shall set them in combination. The **cs**, **sc**, and **rg** operators shall perform the same functions for nonstroking operations. Setting either current colour space to **DeviceRGB** shall initialize the red, green, and blue components of the corresponding current colour to 0.0.

8.6.4.4 DeviceCMYK Colour Space

The **DeviceCMYK** colour space allows colours to be specified according to the subtractive *CMYK* (cyan-magenta-yellow-black) model typical of printers and other paper-based output devices. The four components in a **DeviceCMYK** colour value shall represent the concentrations of these process colorants. Each component shall be a number in the range 0.0 to 1.0, where 0.0 shall denote the complete absence of a process colorant and 1.0 shall denote maximum concentration (absorbs as much as possible of the additive primary).

NOTE As much as the reflective colours (CMYK) decrease reflection with increased ink values and radiant colours (RGB) increases the intensity of colours with increased values the values work in an opposite manner.

EXAMPLE The following shows alternative ways to select the **DeviceCMYK** colour space and a specific colour within that space for stroking operations.

```
/DeviceCMYK CS                   % Set DeviceCMYK colour space
cyan magenta yellow black SC     % Set colour
cyan magenta yellow black K      % Set both in one operation
```

The **CS** and **SC** operators shall select the current stroking colour space and current stroking colour separately; **K** shall set them in combination. The **cs**, **sc**, and **k** operators shall perform the same functions for nonstroking operations. Setting either current colour space to **DeviceCMYK** shall initialize the cyan, magenta, and yellow components of the corresponding current colour to 0.0 and the black component to 1.0.

8.6.5 CIE-Based Colour Spaces

8.6.5.1 General

Calibrated colour in PDF shall be defined in terms of an international standard used in the graphic arts, television, and printing industries. *CIE-based* colour spaces enable a page description to specify colour values in a way that is related to human visual perception. The goal is for the same colour specification to produce consistent results on different output devices, within the limitations of each device; Figure L.2 in Annex L illustrates the kind of variation in colour reproduction that can result from the use of uncalibrated colour on different devices. PDF 1.1 supports three CIE-based colour space families, named **CalGray**, **CalRGB**, and **Lab**; PDF 1.3 added a fourth, named **ICCBased**.

NOTE 1 In PDF 1.1, a colour space family named ***CalCMYK*** was partially defined, with the expectation that its definition would be completed in a future version. However, this feature has been deprecated. PDF 1.3 and later versions support calibrated four-component colour spaces by means of ICC profiles (see 8.6.5.5, "ICCBased Colour Spaces"). A conforming reader should ignore ***CalCMYK*** colour space attributes and render colours specified in this family as if they had been specified using ***DeviceCMYK***.

NOTE 2 The details of the CIE colourimetric system and the theory on which it is based are beyond the scope of this specification; see the Bibliography for sources of further information. The semantics of CIE-based colour spaces are defined in terms of the relationship between the space's components and the tristimulus values *X*, *Y*, and *Z* of the CIE 1931 *XYZ* space. The **CalRGB** and **Lab** colour spaces *(PDF 1.1)* are special cases of three-component CIE-based colour spaces, known as *CIE-based ABC* colour spaces. These spaces are defined in terms of a two-stage, nonlinear transformation of the CIE 1931 *XYZ* space. The formulation of such colour spaces models a simple *zone theory* of colour vision, consisting of a nonlinear trichromatic first stage combined with a nonlinear opponent-colour second stage. This formulation allows colours to be digitized with minimum loss of fidelity, an important consideration in sampled images.

Colour values in a CIE-based *ABC* colour space shall have three components, arbitrarily named *A*, *B*, and *C.* The first stage shall transform these components by first forcing their values to a specified range, then applying *decoding functions*, and then multiplying the results by a 3-by-3 matrix, producing three intermediate components arbitrarily named *L*, *M*, and *N*. The second stage shall transform these intermediate components in a similar fashion, producing the final *X*, *Y*, and *Z* components of the CIE 1931 *XYZ* space (see Figure 22).

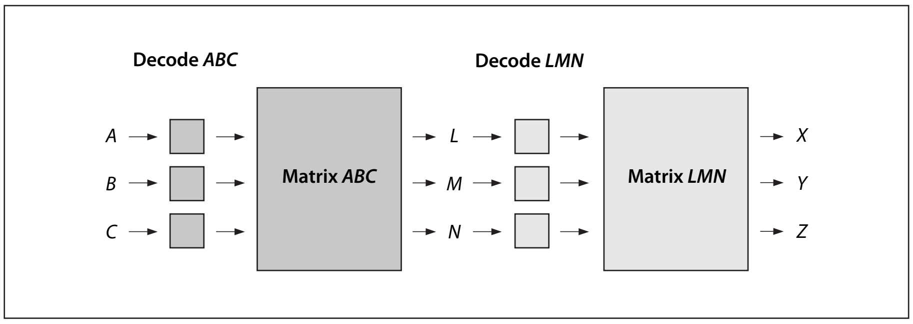


Figure 22 – Component Transformations in a CIE-based *ABC* Colour Space

Colour spaces in the CIE-based families shall be defined by an array

[*name dictionary*]

where *name* is the name of the family and *dictionary* is a dictionary containing parameters that further characterize the space. The entries in this dictionary have specific interpretations that depend on the colour space; some entries are required and some are optional. See the sub-clauses on specific colour space families for details.

Setting the current stroking or nonstroking colour space to any CIE-based colour space shall initialize all components of the corresponding current colour to 0.0 (unless the range of valid values for a given component does not include 0.0, in which case the nearest valid value shall be substituted.)

NOTE 3 The model and terminology used here—*CIE-based ABC* (above) and *CIE-based A* (below)—are derived from the PostScript language, which supports these colour space families in their full generality. PDF supports specific useful cases of CIE-based *ABC* and CIE-based *A* spaces; most others can be represented as ***ICCBased*** spaces.

8.6.5.2 CalGray Colour Spaces

A **CalGray** colour space *(PDF 1.1)* is a special case of a single-component CIE-based colour space, known as a *CIE-based A* colour space. This type of space is the one-dimensional (and usually achromatic) analog of CIE-based *ABC* spaces. Colour values in a CIE-based *A* space shall have a single component, arbitrarily named *A.* Figure 23 illustrates the transformations of the *A* component to *X*, *Y*, and *Z* components of the CIE 1931 *XYZ* space.

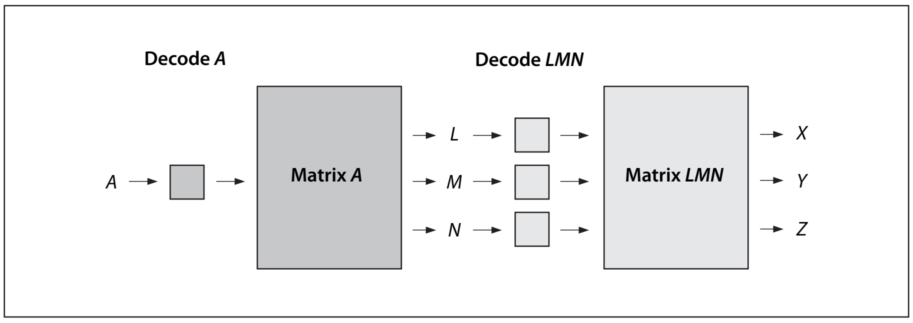


Figure 23 – Component Transformations in a CIE-based *A* Colour Space

A **CalGray** colour space shall be a CIE-based *A* colour space with only one transformation stage instead of two. In this type of space, *A* represents the gray component of a calibrated gray space. This component shall be in the range 0.0 to 1.0. The decoding function (denoted by "Decode *A*" in Figure 23) is a gamma function whose coefficient shall be specified by the **Gamma** entry in the colour space dictionary (see Table 63). The transformation matrix denoted by "Matrix *A*" in the figure is derived from the dictionary's **WhitePoint** entry, as described below. Since there is no second transformation stage, "Decode *LMN*" and "Matrix *LMN*" shall be implicitly taken to be identity transformations.

Table 63 – Entries in a CalGray Colour Space Dictionary

Key	Type	Value
WhitePoint	array	*(Required)* An array of three numbers [X_W Y_W Z_W] specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see 8.6.5.3, "CalRGB Colour Spaces", for further discussion. The numbers X_W and Z_W shall be positive, and Y_W shall be equal to 1.0.
BlackPoint	array	*(Optional)* An array of three numbers [X_B Y_B Z_B] specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see 8.6.5.3, "CalRGB Colour Spaces", for further discussion. All three of these numbers shall be non-negative. Default value: [0.0 0.0 0.0].
Gamma	number	*(Optional)* A number *G* defining the gamma for the gray *(A)* component. *G* shall be positive and is generally greater than or equal to 1. Default value: 1.

The transformation defined by the **Gamma** and **WhitePoint** entries is

$$\begin{aligned} X &= L = X_W \times A^G \\ Y &= M = Y_W \times A^G \\ Z &= N = Z_W \times A^G \end{aligned}$$

In other words, the *A* component shall be first decoded by the gamma function, and the result shall be multiplied by the components of the white point to obtain the *L*, *M*, and *N* components of the intermediate representation. Since there is no second stage, the *L*, *M*, and *N* components shall also be the *X*, *Y*, and *Z* components of the final representation.

EXAMPLE 1 The examples in this sub-clause illustrate interesting and useful special cases of **CalGray** spaces. This example establishes a space consisting of the Y dimension of the CIE 1931 XYZ space with the CCIR XA/11–recommended D65 white point.

```
[ /CalGray
      << /WhitePoint [0.9505 1.0000 1.0890] >>
]
```

EXAMPLE 2 This example establishes a calibrated gray space with the CCIR XA/11–recommended D65 white point and opto-electronic transfer function.

```
[ /CalGray
      << /WhitePoint [0.9505 1.0000 1.0890]
        /Gamma 2.222
      >>
]
```

8.6.5.3 CalRGB Colour Spaces

A **CalRGB** colour space is a CIE-based *ABC* colour space with only one transformation stage instead of two. In this type of space, *A*, *B*, and *C* represent calibrated red, green, and blue colour values. These three colour components shall be in the range 0.0 to 1.0; component values falling outside that range shall be adjusted to the nearest valid value without error indication. The decoding functions (denoted by “Decode *ABC*” in Figure 22) are gamma functions whose coefficients shall be specified by the **Gamma** entry in the colour space dictionary (see Table 64). The transformation matrix denoted by “Matrix *ABC*” in Figure 22 shall be defined by the dictionary’s **Matrix** entry. Since there is no second transformation stage, “Decode *LMN*” and “Matrix *LMN*” shall be implicitly taken to be identity transformations.

Table 64 – Entries in a CalRGB Colour Space Dictionary

Key	Type	Value
WhitePoint	array	*(Required)* An array of three numbers [X_W Y_W Z_W] specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see below for further discussion. The numbers X_W and Z_W shall be positive, and Y_W shall be equal to 1.0.
BlackPoint	array	*(Optional)* An array of three numbers [X_B Y_B Z_B] specifying the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see below for further discussion. All three of these numbers shall be non-negative. Default value: [0.0 0.0 0.0].
Gamma	array	*(Optional)* An array of three numbers [G_R G_G G_B] specifying the gamma for the red, green, and blue *(A*, *B*, and *C)* components of the colour space. Default value: [1.0 1.0 1.0].
Matrix	array	*(Optional)* An array of nine numbers [X_A Y_A Z_A X_B Y_B Z_B X_C Y_C Z_C] specifying the linear interpretation of the decoded *A*, *B*, and *C* components of the colour space with respect to the final *XYZ* representation. Default value: the identity matrix [1 0 0 0 1 0 0 0 1].

The **WhitePoint** and **BlackPoint** entries in the colour space dictionary shall control the overall effect of the CIE-based gamut mapping function described in sub-clause 10.2, "CIE-Based Colour to Device Colour". Typically, the colours specified by **WhitePoint** and **BlackPoint** shall be mapped to the nearly lightest and nearly darkest achromatic colours that the output device is capable of rendering in a way that preserves colour appearance and visual contrast.

WhitePoint represents the diffuse achromatic highlight, not a specular highlight. Specular highlights, achromatic or otherwise, are often reproduced lighter than the diffuse highlight. **BlackPoint** represents the diffuse achromatic shadow; its value is limited by the dynamic range of the input device. In images produced by a photographic system, the values of **WhitePoint** and **BlackPoint** vary with exposure, system response, and artistic intent; hence, their values are image-dependent.

The transformation defined by the **Gamma** and **Matrix** entries in the **CalRGB** colour space dictionary shall be

$$\begin{aligned} X &= L = X_A \times A^{G_R} + X_B \times B^{G_G} + X_C \times C^{G} \\ Y &= M = Y_A \times A^{G_R} + Y_B \times B^{G_G} + Y_C \times C^{G} \\ Z &= N = Z_A \times A^{G_R} + Z_B \times B^{G_G} + Z_C \times C^{G_B} \end{aligned}$$

The *A*, *B*, and *C* components shall first be decoded individually by the gamma functions. The results shall be treated as a three-element vector and multiplied by **Matrix** (a 3-by-3 matrix) to obtain the *L*, *M*, and *N* components of the intermediate representation. Since there is no second stage, these shall also be the *X*, *Y*, and *Z* components of the final representation.

EXAMPLE The following shows an example of a **CalRGB** colour space for the CCIR XA/11–recommended D65 white point with 1.8 gammas and Sony Trinitron phosphor chromaticities.

```
[ /CalRGB
      << /WhitePoint [0.9505 1.0000 1.0890]
         /Gamma [1.8000 1.8000 1.8000]
         /Matrix [ 0.4497 0.2446 0.0252
                   0.3163 0.6720 0.1412
                   0.1845 0.0833 0.9227
                 ]
      >>
]
```

The parameters of a **CalRGB** colour space may be specified in terms of the CIE 1931 chromaticity coordinates (x_R, y_R), (x_G, y_G), (x_B, y_B) of the red, green, and blue phosphors, respectively, and the chromaticity (x_W, y_W) of the diffuse white point corresponding to a linear *RGB* value (*R*, *G*, *B*), where *R*, *G*, and *B* should all equal 1.0. The standard CIE notation uses lowercase letters to specify chromaticity coordinates and uppercase letters to specify tristimulus values. Given this information, **Matrix** and **WhitePoint** shall be calculated as follows:

$$z = y_W \times ((x_G - x_B) \times y_R - (x_R - x_B) \times y_G + (x_R - x_G) \times y_B$$

$$Y_A = \frac{y_R}{R} \times \frac{(x_G - x_B) \times y_W - (x_W - x_B) \times y_G + (x_W - x_G) \times y}{z}$$

$$X_A = Y_A \times \frac{x_R}{y_R} \qquad Z_A = Y_A \times \left(\frac{1 - x_R}{y_R} - 1 \right)$$

$$Y_B = -\frac{y_G}{G} \times \frac{(x_R - x_B) \times y_W - (x_W - x_B) \times y_R + (x_W - x_R) \times y}{z}$$

$$X_B = Y_B \times \frac{x_G}{y_G} \qquad Z_B = Y_B \times \left(\frac{1 - x_G}{y_G} - 1 \right)$$

$$Y_C = \frac{y_B}{B} \times \frac{(x_R - x_G) \times y_W - (x_W - x_G) \times y_R + (x_W - x_R) \times y}{z}$$

$$X_C = Y_C \times \frac{x_B}{y_B} \qquad Z_C = Y_C \times \left(\frac{1 - x_B}{y_B} - 1 \right)$$

$$\begin{aligned} X_W &= X_A \times R + X_B \times G + X_C \times I \\ Y_W &= Y_A \times R + Y_B \times G + Y_C \times I \\ Z_W &= Z_A \times R + Z_B \times G + Z_C \times B \end{aligned}$$

8.6.5.4 Lab Colour Spaces

A **Lab** colour space is a CIE-based *ABC* colour space with two transformation stages (see Figure 22). In this type of space, *A*, *B*, and *C* represent the *L**, *a**, and *b** components of a CIE 1976 *L*a*b** space. The range of the first *(L*)* component shall be 0 to 100; the ranges of the second and third *(a** and *b*)* components shall be defined by the **Range** entry in the colour space dictionary (see Table 65).

Figure L.3 in Annex L illustrates the coordinates of a typical **Lab** colour space; Figure L.4 in Annex L compares the gamuts (ranges of representable colours) for *L*a*b**, *RGB*, and *CMYK* spaces.

Table 65 – Entries in a Lab Colour Space Dictionary

Key	Type	Value
WhitePoint	array	*(Required)* An array of three numbers [X_W Y_W Z_W] that shall specify the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse white point; see 8.6.5.3, "CalRGB Colour Spaces" for further discussion. The numbers X_W and Z_W shall be positive, and Y_W shall be 1.0.
BlackPoint	array	*(Optional)* An array of three numbers [X_B Y_B Z_B] that shall specify the tristimulus value, in the CIE 1931 *XYZ* space, of the diffuse black point; see 8.6.5.3, "CalRGB Colour Spaces" for further discussion. All three of these numbers shall be non-negative. Default value: [0.0 0.0 0.0].
Range	array	*(Optional)* An array of four numbers [a_{min} a_{max} b_{min} b_{max}] that shall specify the range of valid values for the *a** and *b** (*B* and *C*) components of the colour space—that is, $a_{min} \leq a^* \leq a_{max}$ and $b_{min} \leq b^* \leq b_{max}$ Component values falling outside the specified range shall be adjusted to the nearest valid value without error indication. Default value: [–100 100 –100 100].

A **Lab** colour space shall not specify explicit decoding functions or matrix coefficients for either stage of the transformation from *L*a*b** space to *XYZ* space (denoted by "Decode *ABC*," "Matrix *ABC*," "Decode *LMN*," and "Matrix *LMN*" in Figure 22). Instead, these parameters shall have constant implicit values. The first transformation stage shall be defined by the equations

$$L = \frac{L^* + 16}{116} + \frac{a^*}{500}$$
$$M = \frac{L^* + 16}{116}$$
$$N = \frac{L^* + 16}{116} - \frac{b^*}{200}$$

The second transformation stage shall be

$$X = X_W \times g(L)$$
$$Y = Y_W \times g(M)$$
$$Z = Z_W \times g(N)$$

where the function *g*(*x*) shall be defined as

$$g(x) = x^3 \qquad \text{if } x \geq \frac{6}{29}$$
$$g(x) = \frac{108}{841} \times \left(x - \frac{4}{29}\right) \qquad \text{otherwise}$$

EXAMPLE The following defines the CIE 1976 L*a*b* space with the CCIR XA/11–recommended D65 white point. The a* and b* components, although theoretically unbounded, are defined to lie in the useful range -128 to +127.

```
[ /Lab
      <<  /WhitePoint  [0.9505  1.0000  1.0890]
          /Range  [-128  127  -128  127]
      >>
]
```

8.6.5.5 ICCBased Colour Spaces

ICCBased colour spaces *(PDF 1.3)* shall be based on a cross-platform *colour profile* as defined by the International Color Consortium (ICC) (see, “Bibliography“). Unlike the **CalGray**, **CalRGB**, and **Lab** colour spaces, which are characterized by entries in the colour space dictionary, an **ICCBased** colour space shall be characterized by a sequence of bytes in a standard format. Details of the profile format can be found in the ICC specification (see, “Bibliography“).

An **ICCBased** colour space shall be an array:

```
[/ICCBased  stream]
```

The stream shall contain the ICC profile. Besides the usual entries common to all streams (see Table 5), the profile stream shall have the additional entries listed in Table 66.

Table 66 – Additional Entries Specific to an ICC Profile Stream Dictionary

Key	Type	Value
N	integer	*(Required)* The number of colour components in the colour space described by the ICC profile data. This number shall match the number of components actually in the ICC profile. **N** shall be 1, 3, or 4.

Table 66 – Additional Entries Specific to an ICC Profile Stream Dictionary (continued)

Key	Type	Value
Alternate	array or name	*(Optional)* An alternate colour space that shall be used in case the one specified in the stream data is not supported. Non-conforming readers may use this colour space. The alternate space may be any valid colour space (except a **Pattern** colour space) that has the number of components specified by **N**. If this entry is omitted and the conforming reader does not understand the ICC profile data, the colour space that shall be used is **DeviceGray**, **DeviceRGB**, or **DeviceCMYK**, depending on whether the value of **N** is 1, 3, or 4, respectively. There shall not be conversion of source colour values, such as a tint transformation, when using the alternate colour space. Colour values within the range of the **ICCBased** colour space might not be within the range of the alternate colour space. In this case, the nearest values within the range of the alternate space shall be substituted.
Range	array	*(Optional)* An array of 2 × **N** numbers [min_0 max_0 min_1 max_1 …] that shall specify the minimum and maximum valid values of the corresponding colour components. These values shall match the information in the ICC profile. Default value: [0.0 1.0 0.0 1.0 …].
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* that shall contain metadata for the colour space (see 14.3.2, "Metadata Streams").

The ICC specification is an evolving standard. Table 67 shows the versions of the ICC specification on which the **ICCBased** colour spaces that PDF versions 1.3 and later shall use. (Earlier versions of the ICC specification shall also be supported.)

Table 67 – ICC Specification Versions Supported by ICC Based Colour Spaces

PDF Version	ICC Specification Version
1.3	3.3
1.4	ICC.1:1998-09 and its addendum ICC.1A:1999-04
1.5	ICC.1:2001-12
1.6	ICC.1:2003-09
1.7	ICC.1:2004-10 (ISO 15076-1:2005)

Conforming writers and readers should follow these guidelines:

- A conforming reader shall support ICC.1:2004:10 as required by PDF 1.7, which will enable it to properly render all embedded ICC profiles regardless of the PDF version.
- A conforming reader shall always process an embedded ICC profile according to the corresponding version of the PDF being processed as shown in Table 67 above; it shall not substitute the Alternate colour space in these cases.
- A conforming writer should use ICC 1:2004-10 profiles. It may embed profiles conforming to a later ICC version. The conforming reader should process such profiles according to Table 67; if that is not possible, it shall substitute the Alternate colour space.
- Conforming writers shall only use the profile types shown in Table 68 for specifying calibrated colour spaces for colouring graphic objects. Each of the indicated fields shall have one of the values listed for that field in the second column of the table. Profiles shall satisfy *both* the criteria shown in the table. The terminology is taken from the ICC specifications.

NOTE 1 XYZ and 16-bit L*a*b* profiles are not listed.

Table 68 – ICC Profile Types

Header Field	Required Value
deviceClass	icSigInputClass ('scnr') icSigDisplayClass ('mntr') icSigOutputClass ('prtr') icSigColorSpaceClass ('spac')
colorSpace	icSigGrayData ('GRAY') icSigRgbData ('RGB ') icSigCmykData ('CMYK') icSigLabData ('Lab ')

The terminology used in PDF colour spaces and ICC colour profiles is similar, but sometimes the same terms are used with different meanings. The default value for each component in an **ICCBased** colour space is 0. The range of each colour component is a function of the colour space specified by the profile and is indicated in the ICC specification. The ranges for several ICC colour spaces are shown in Table 69.

Table 69 – Ranges for Typical ICC Colour Spaces

ICC Colour Space	Component Ranges
Gray	[0.0 1.0]
RGB	[0.0 1.0]
CMYK	[0.0 1.0]
L*a*b*	*L**: [0 100]; *a** and *b**: [−128 127]

Since the **ICCBased** colour space is being used as a source colour space, only the “to CIE” profile information (*AToB* in ICC terminology) shall be used; the “from CIE” (*BToA*) information shall be ignored when present. An ICC profile may also specify a *rendering intent*, but a conforming reader shall ignore this information; the rendering intent shall be specified in PDF by a separate parameter (see 8.6.5.8, "Rendering Intents").

The requirements stated above apply to an ***ICCBased*** colour space that is used to specify the source colours of graphics objects. When such a space is used as the blending colour space for a transparency group in the transparent imaging model (see 11.3.4, "Blending Colour Space"; 11.4, "Transparency Groups"; and 11.6.6, "Transparency Group XObjects"), it shall have both “to CIE” *(AToB)* and “from CIE” *(BToA)* information. This is because the group colour space shall be used as both the destination for objects being painted within the group and the source for the group’s results. ICC profiles shall also be used in specifying *output intents* for matching the colour characteristics of a PDF document with those of a target output device or production environment. When used in this context, they shall be subject to still other constraints on the “to CIE” and “from CIE” information; see 14.11.5, "Output Intents", for details.

The representations of **ICCBased** colour spaces are less compact than **CalGray**, **CalRGB**, and **Lab**, but can represent a wider range of colour spaces.

NOTE 2 One particular colour space is the “standard *RGB*” or *sRGB*, defined in the International Electrotechnical Commission (IEC) document *Color Measurement and Management in Multimedia Systems and Equipment* (see, “Bibliography“). In PDF, the *sRGB* colour space can only be expressed as an **ICCBased** space, although it can be approximated by a **CalRGB** space.

EXAMPLE The following shows an **ICCBased** colour space for a typical three-component RGB space. The profile’s data has been encoded in hexadecimal representation for readability; in actual practice, a lossless decompression filter such as **FlateDecode** should be used.

```
10 0 obj                                  % Colour space
   [/ICCBased 15 0 R]
```

```
endobj

15  0  obj                                        % ICC profile stream
    <<  /N  3
        /Alternate  /DeviceRGB
        /Length  1605
        /Filter  /ASCIIHexDecode
    >>

stream
00  00  02  0C  61  70  70  6C  02  00  00  00  6D  6E  74  72
52  47  42  20  58  59  5A  20  07  CB  00  02  00  16  00  0E
00  22  00  2C  61  63  73  70  41  50  50  4C  00  00  00  00
61  70  70  6C  00  00  04  01  00  00  00  00  00  00  00  02
00  00  00  00  00  00  F6  D4  00  01  00  00  00  00  D3  2B
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  09  64  65  73  63  00  00  00  F0  00  00  00  71
72  58  59  5A  00  00  01  64  00  00  00  14  67  58  59  5A
00  00  01  78  00  00  00  14  62  58  59  5A  00  00  01  8C
00  00  00  14  72  54  52  43  00  00  01  A0  00  00  00  0E
67  54  52  43  00  00  01  B0  00  00  00  0E  62  54  52  43
00  00  01  C0  00  00  00  0E  77  74  70  74  00  00  01  D0
00  00  00  14  63  70  72  74  00  00  01  E4  00  00  00  27
64  65  73  63  00  00  00  00  00  00  00  17  41  70  70  6C
65  20  31  33  22  20  52  47  42  20  53  74  61  6E  64  61
72  64  00  00  00  00  00  00  00  00  00  00  00  17  41  70
70  6C  65  20  31  33  22  20  52  47  42  20  53  74  61  6E
64  61  72  64  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
00  58  59  5A  58  59  5A  20  00  00  00  00  00  00  63  0A
00  00  35  0F  00  00  03  30  58  59  5A  20  00  00  00  00
00  00  53  3D  00  00  AE  37  00  00  15  76  58  59  5A  20
00  00  00  00  00  00  40  89  00  00  1C  AF  00  00  BA  82
63  75  72  76  00  00  00  00  00  00  00  01  01  CC  63  75
63  75  72  76  00  00  00  00  00  00  00  01  01  CC  63  75
63  75  72  76  00  00  00  00  00  00  00  01  01  CC  58  59
58  59  5A  20  00  00  00  00  00  00  F3  1B  00  01  00  00
00  01  67  E7  74  65  78  74  00  00  00  00  20  43  6F  70
79  72  69  67  68  74  20  41  70  70  6C  65  20  43  6F  6D
70  75  74  65  72  73  20  31  39  39  34  00  >
endstream
endobj
```

8.6.5.6 Default Colour Spaces

Colours that are specified in a device colour space (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**) are device-dependent. By setting *default colour spaces (PDF 1.1)*, a conforming writer can request that such colours shall be systematically transformed *(remapped)* into device-independent CIE-based colour spaces. This capability can be useful in a variety of circumstances:

- A document originally intended for one output device is redirected to a different device.
- A document is intended to be compatible with non-compliant readers and thus cannot specify CIE-based colours directly.
- Colour corrections or rendering intents need to be applied to device colours (see 8.6.5.8, "Rendering Intents").

A colour space is selected for painting each graphics object. This is either the current colour space parameter in the graphics state or a colour space given as an entry in an image XObject, inline image, or shading dictionary. Regardless of how the colour space is specified, it shall be subject to remapping as described below.

When a device colour space is selected, the **ColorSpace** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries") is checked for the presence of an entry designating a corresponding default colour space (**DefaultGray**, **DefaultRGB**, or **DefaultCMYK**, corresponding to **DeviceGray**, **DeviceRGB**, or **DeviceCMYK**, respectively). If such an entry is present, its value shall be used as the colour space for the operation currently being performed.

Colour values in the original device colour space shall be passed unchanged to the default colour space, which shall have the same number of components as the original space. The default colour space should be chosen to be compatible with the original, taking into account the components' ranges and whether the components are additive or subtractive. If a colour value lies outside the range of the default colour space, it shall be adjusted to the nearest valid value.

Any colour space other than a **Lab**, **Indexed**, or **Pattern** colour space may be used as a default colour space and it should be compatible with the original device colour space as described above.

If the selected space is a special colour space based on an underlying device colour space, the default colour space shall be used in place of the underlying space. This shall apply to the following colour spaces:

- The underlying colour space of a **Pattern** colour space
- The base colour space of an **Indexed** colour space
- The alternate colour space of a **Separation** or **DeviceN** colour space (but only if the alternate colour space is actually selected)

See 8.6.6, "Special Colour Spaces", for details on these colour spaces.

There is no conversion of colour values, such as a tint transformation, when using the default colour space. Colour values that are within the range of the device colour space might not be within the range of the default colour space (particularly if the default is an **ICCBased** colour space). In this case, the nearest values within the range of the default space are used. For this reason, a **Lab** colour space shall not be used as the **DefaultRGB** colour space.

8.6.5.7 Implicit Conversion of CIE-Based Colour Spaces

In cases where a source colour space accurately represents the particular output device being used, a conforming reader should avoid converting the component colour values but use the source values directly as output values. This avoids any unwanted computational error and in the case of 4 component colour spaces avoids the conversion from 4 components to 3 and back to 4, a process that loses critical colour information.

NOTE 1 In workflows in which PDF documents are intended for rendering on a specific target output device (such as a printing press with particular inks and media), it is often useful to specify the source colours for some or all of a document's objects in a CIE-based colour space that matches the calibration of the intended device. The resulting document, although tailored to the specific characteristics of the target device, remains device-independent and will produce reasonable results if retargeted to a different output device. However, the expectation is that if the document is printed on the intended target device, source colours that have been specified in a colour space matching the calibration of the device will pass through unchanged, without conversion to and from the intermediate CIE 1931 *XYZ* space as depicted in Figure 22.

NOTE 2 In particular, when colours intended for a *CMYK* output device are specified in an **ICCBased** colour space using a matching *CMYK* printing profile, converting such colours from four components to three and back is unnecessary and results in a loss of fidelity in the black component. In such cases, a conforming reader may provide the ability for the user to specify a particular calibration to use for printing, proofing, or previewing. This calibration is then considered to be that of the native colour space of the intended output device (typically **DeviceCMYK**), and colours expressed in a CIE-based source colour space matching it can be treated as if they were specified directly in the device's native colour space.

NOTE 3 The conditions under which such implicit conversion is done cannot be specified in PDF, since nothing in PDF describes the calibration of the output device (although an output intent dictionary, if present, may suggest such a calibration; see 14.11.5, "Output Intents"). The conversion is completely hidden by the conforming reader and plays no part in the interpretation of PDF colour spaces.

When this type of implicit conversion is done, all of the semantics of the device colour space shall also apply, even though they do not apply to CIE-based spaces in general. In particular:

- The nonzero overprint mode (see 8.6.7, "Overprint Control") shall determine the interpretation of colour component values in the space.
- If the space is used as the blending colour space for a transparency group in the transparent imaging model (see 11.3.4, "Blending Colour Space"; 11.4, "Transparency Groups"; and 11.6.6, "Transparency Group XObjects"), components of the space, such as **Cyan**, may be selected in a **Separation** or **DeviceN** colour space used within the group (see 8.6.6.4, "Separation Colour Spaces" and 8.6.6.5, "DeviceN Colour Spaces").
- Likewise, any uses of device colour spaces for objects within such a transparency group have well-defined conversions to the group colour space.

NOTE 4 A source colour space can be specified directly (for example, with an ***ICCBased*** colour space) or indirectly using the default colour space mechanism (for example, ***DefaultCMYK***; see 8.6.5.6, "Default Colour Spaces"). The implicit conversion of a CIE-based colour space to a device space should not depend on whether the CIE-based space is specified directly or indirectly.

8.6.5.8 Rendering Intents

Although CIE-based colour specifications are theoretically device-independent, they are subject to practical limitations in the colour reproduction capabilities of the output device. Such limitations may sometimes require compromises to be made among various properties of a colour specification when rendering colours for a given device. Specifying a *rendering intent (PDF 1.1)* allows a conforming writer to set priorities regarding which of these properties to preserve and which to sacrifice.

EXAMPLE The conforming writer might request that colours falling within the output device's gamut (the range of colours it can reproduce) be rendered exactly while sacrificing the accuracy of out-of-gamut colours, or that a scanned image such as a photograph be rendered in a perceptually pleasing manner at the cost of strict colourimetric accuracy.

Rendering intents shall be specified with the **ri** operator (see 8.4.4, "Graphics State Operators"), the **RI** entry in a graphics state parameter dictionary (see 8.4.5, "Graphics State Parameter Dictionaries"), or with the **Intent** entry in image dictionaries (see 8.9.5, "Image Dictionaries"). The value shall be a name identifying the rendering intent. Table 70 lists the standard rendering intents that shall be recognized. Figure L.5 in Annex L illustrates their effects. These intents have been chosen to correspond to those defined by the International Color Consortium (ICC), an industry organization that has developed standards for device-independent colour. If a conforming reader does not recognize the specified name, it shall use the **RelativeColorimetric** intent by default.

NOTE Note, however, that the exact set of rendering intents supported may vary from one output device to another; a particular device may not support all possible intents or may support additional ones beyond those listed in the table.

See 11.7.5, "Rendering Parameters and Transparency", and in particular 11.7.5.3, "Rendering Intent and Colour Conversions", for further discussion of the role of rendering intents in the transparent imaging model.

Table 70 – Rendering Intents

Name	Description
AbsoluteColorimetric	Colours shall be represented solely with respect to the light source; no correction shall be made for the output medium's white point (such as the colour of unprinted paper). Thus, for example, a monitor's white point, which is bluish compared to that of a printer's paper, would be reproduced with a blue cast. In-gamut colours shall be reproduced exactly; out-of-gamut colours shall be mapped to the nearest value within the reproducible gamut. NOTE 1 This style of reproduction has the advantage of providing exact colour matches from one output medium to another. It has the disadvantage of causing colours with *Y* values between the medium's white point and 1.0 to be out of gamut. A typical use might be for logos and solid colours that require exact reproduction across different media.
RelativeColorimetric	Colours shall be represented with respect to the combination of the light source and the output medium's white point (such as the colour of unprinted paper). Thus, a monitor's white point can be reproduced on a printer by simply leaving the paper unmarked, ignoring colour differences between the two media. In-gamut colours shall be reproduced exactly; out-of-gamut colours shall be mapped to the nearest value within the reproducible gamut. NOTE 2 This style of reproduction has the advantage of adapting for the varying white points of different output media. It has the disadvantage of not providing exact colour matches from one medium to another. A typical use might be for vector graphics.
Saturation	Colours shall be represented in a manner that preserves or emphasizes saturation. Reproduction of in-gamut colours may or may not be colourimetrically accurate. NOTE 3 A typical use might be for business graphics, where saturation is the most important attribute of the colour.
Perceptual	Colours shall be represented in a manner that provides a pleasing perceptual appearance. To preserve colour relationships, both in-gamut and out-of-gamut colours shall be generally modified from their precise colourimetric values. NOTE 4 A typical use might be for scanned images.

8.6.6 Special Colour Spaces

8.6.6.1 General

Special colour spaces add features or properties to an underlying colour space. There are four special colour space families: **Pattern**, **Indexed**, **Separation**, and **DeviceN**.

8.6.6.2 Pattern Colour Spaces

A **Pattern** colour space *(PDF 1.2)* specifies that an area is to be painted with a *pattern* rather than a single colour. The pattern shall be either a *tiling pattern* (type 1) or a *shading pattern* (type 2). 8.7, "Patterns", discusses patterns in detail.

8.6.6.3 Indexed Colour Spaces

An **Indexed** colour space specifies that an area is to be painted using a *colour map* or *colour table* of arbitrary colours in some other space. A conforming reader shall treat each sample value as an index into the colour table and shall use the colour value it finds there. This technique can considerably reduce the amount of data required to represent a sampled image.

An **Indexed** colour space shall be defined by a four-element array:

[/Indexed *base* *hival* *lookup*]

The first element shall be the colour space family name **Indexed**. The remaining elements shall be parameters that an **Indexed** colour space requires; their meanings are discussed below. Setting the current stroking or nonstroking colour space to an **Indexed** colour space shall initialize the corresponding current colour to 0.

The *base* parameter shall be an array or name that identifies the *base colour space* in which the values in the colour table are to be interpreted. It shall be any device or CIE-based colour space or *(PDF 1.3)* a **Separation** or **DeviceN** space, but shall not be a **Pattern** space or another **Indexed** space. If the base colour space is **DeviceRGB**, the values in the colour table shall be interpreted as red, green, and blue components; if the base colour space is a CIE-based *ABC* space such as a **CalRGB** or **Lab** space, the values shall be interpreted as *A*, *B*, and *C* components.

The *hival* parameter shall be an integer that specifies the maximum valid index value. The colour table shall be indexed by integers in the range 0 to *hival*. *hival* shall be no greater than 255, which is the integer required to index a table with 8-bit index values.

The colour table shall be defined by the *lookup* parameter, which may be either a stream or *(PDF 1.2)* a byte string. It shall provide the mapping between index values and the corresponding colours in the base colour space.

The colour table data shall be *m* ¥ (*hival* + 1) bytes long, where *m* is the number of colour components in the base colour space. Each byte shall be an unsigned integer in the range 0 to 255 that shall be scaled to the range of the corresponding colour component in the base colour space; that is, 0 corresponds to the minimum value in the range for that component, and 255 corresponds to the maximum.

The colour components for each entry in the table shall appear consecutively in the string or stream.

EXAMPLE 1 If the base colour space is **DeviceRGB** and the indexed colour space contains two colours, the order of bytes in the string or stream is R_0 G_0 B_0 R_1 G_1 B_1, where letters denote the colour component and numeric subscripts denote the table entry.

EXAMPLE 1 The following illustrates the specification of an Indexed colour space that maps 8-bit index values to three-component colour values in the **DeviceRGB** colour space.

```
[ /Indexed
        /DeviceRGB
        255
        <000000 FF0000 00FF00 0000FF B57342 …>
]
```

The example shows only the first five colour values in the *lookup* string; in all, there should be 256 colour values and the string should be 768 bytes long. Having established this colour space, the program can now specify colours as single-component values in the range 0 to 255. For example, a colour value of 4 selects an *RGB* colour whose components are coded as the hexadecimal integers B5, 73, and 42.

Dividing these by 255 and scaling the results to the range 0.0 to 1.0 yields a colour with red, green, and blue components of 0.710, 0.451, and 0.259, respectively.

Although an **Indexed** colour space is useful mainly for images, index values can also be used with the colour selection operators **SC**, **SCN**, **sc**, and **scn**.

EXAMPLE 2 The following selects the same colour as does an image sample value of 123.

```
123 sc
```

The index value should be an integer in the range 0 to *hival*. If the value is a real number, it shall be rounded to the nearest integer; if it is outside the range 0 to *hival*, it shall be adjusted to the nearest value within that range.

8.6.6.4 Separation Colour Spaces

A **Separation** colour space *(PDF 1.2)* provides a means for specifying the use of additional colorants or for isolating the control of individual colour components of a device colour space for a subtractive device. When such a space is the current colour space, the current colour shall be a single-component value, called a *tint*, that controls the application of the given colorant or colour components only.

NOTE 1 Colour output devices produce full colour by combining *primary* or *process colorants* in varying amounts. On an additive colour device such as a display, the primary colorants consist of red, green, and blue phosphors; on a subtractive device such as a printer, they typically consist of cyan, magenta, yellow, and sometimes black inks. In addition, some devices can apply special colorants, often called *spot colorants*, to produce effects that cannot be achieved with the standard process colorants alone. Examples include metallic and fluorescent colours and special textures.

NOTE 2 When printing a page, most devices produce a single *composite* page on which all process colorants (and spot colorants, if any) are combined. However, some devices, such as imagesetters, produce a separate, monochromatic rendition of the page, called a *separation*, for each colorant. When the separations are later combined—on a printing press, for example—and the proper inks or other colorants are applied to them, the result is a full-colour page.

NOTE 3 The term *separation* is often misused as a synonym for an individual device colorant. In the context of this discussion, a printing system that produces separations generates a separate piece of physical medium (generally film) for each colorant. It is these pieces of physical medium that are correctly referred to as separations. A particular colorant properly constitutes a separation only if the device is generating physical separations, one of which corresponds to the given colorant. The ***Separation*** colour space is so named for historical reasons, but it has evolved to the broader purpose of controlling the application of individual colorants in general, regardless of whether they are actually realized as physical separations.

NOTE 4 The operation of a ***Separation*** colour space itself is independent of the characteristics of any particular output device. Depending on the device, the space may or may not correspond to a true, physical separation or to an actual colorant. For example, a ***Separation*** colour space could be used to control the application of a single process colorant (such as cyan) on a composite device that does not produce physical separations, or could represent a colour (such as orange) for which no specific colorant exists on the device. A ***Separation*** colour space provides consistent, predictable behaviour, even on devices that cannot directly generate the requested colour.

A **Separation** colour space is defined as follows:

[/Separation *name alternateSpace tintTransform*]

It shall be a four-element array whose first element shall be the colour space family name **Separation**. The remaining elements are parameters that a **Separation** colour space requires; their meanings are discussed below.

A colour value in a **Separation** colour space shall consist of a single tint component in the range 0.0 to 1.0. The value 0.0 shall represent the minimum amount of colorant that can be applied; 1.0 shall represent the maximum. Tints shall always be treated as *subtractive* colours, even if the device produces output for the designated component by an additive method. Thus, a tint value of 0.0 denotes the lightest colour that can be achieved with the given colorant, and 1.0 is the darkest. The initial value for both the stroking and nonstroking colour in the graphics state shall be 1.0. The **SCN** and **scn** operators respectively shall set the current stroking and nonstroking colour to a tint value. A sampled image with single-component samples may also be used as a source of tint values.

NOTE 5 This convention is the same as for **DeviceCMYK** colour components but opposite to the one for **DeviceGray** and **DeviceRGB**.

The *name* parameter is a name object that shall specify the name of the colorant that this **Separation** colour space is intended to represent (or one of the special names **All** or **None**; see below). Such colorant names are arbitrary, and there may be any number of them, subject to implementation limits.

The special colorant name **All** shall refer collectively to all colorants available on an output device, including those for the standard process colorants. When a **Separation** space with this colorant name is the current colour space, painting operators shall apply tint values to all available colorants at once.

NOTE 6 This is useful for purposes such as painting registration targets in the same place on every separation. Such marks are typically painted as the last step in composing a page to ensure that they are not overwritten by subsequent painting operations.

The special colorant name **None** shall not produce any visible output. Painting operations in a **Separation** space with this colorant name shall have no effect on the current page.

A conforming reader shall support **Separation** colour spaces with the colorant names **All** and **None** on all devices, even if the devices are not capable of supporting any others. When processing **Separation** spaces with either of these colorant names conforming readers shall ignore the *alternateSpace* and *tintTransform* parameters (discussed below), although valid values shall still be provided.

At the moment the colour space is set to a **Separation** space, the conforming reader shall determine whether the device has an available colorant corresponding to the name of the requested space. If so, the conforming reader shall ignore the *alternateSpace* and *tintTransform* parameters; subsequent painting operations within the space shall apply the designated colorant directly, according to the tint values supplied.

The preceding paragraph applies only to subtractive output devices such as printers and imagesetters. For an additive device such as a computer display, a **Separation** colour space never applies a process colorant directly; it always reverts to the alternate colour space as described below. This is because the model of applying process colorants independently does not work as intended on an additive device.

EXAMPLE 1 Painting tints of the ***Red*** component on a white background produces a result that varies from white to cyan.

This exception applies only to colorants for additive devices, not to the specific names **Red**, **Green**, and **Blue**. In contrast, a printer might have a (subtractive) ink named **Red**, which should work as a **Separation** colour space just the same as any other supported colorant.

If the colorant name associated with a **Separation** colour space does not correspond to a colorant available on the device, the conforming reader shall arrange for subsequent painting operations to be performed in an *alternate colour space.* The intended colours may be approximated by colours in a device or CIE-based colour space, which shall then be rendered with the usual primary or process colorants:

- The *alternateSpace* parameter shall be an array or name object that identifies the alternate colour space, which may be any device or CIE-based colour space but may not be another special colour space (**Pattern**, **Indexed**, **Separation**, or **DeviceN**).

- The *tintTransform* parameter shall be a function (see 7.10, "Functions"). During subsequent painting operations, a conforming reader calls this function to transform a tint value into colour component values in the alternate colour space. The function shall be called with the tint value and shall return the corresponding colour component values. That is, the number of components and the interpretation of their values shall depend on the alternate colour space.

NOTE 7 Painting in the alternate colour space may produce a good approximation of the intended colour when only opaque objects are painted. However, it does not correctly represent the interactions between an object and its backdrop when the object is painted with transparency or when overprinting (see 8.6.7, "Overprint Control") is enabled.

EXAMPLE 2 The following illustrates the specification of a **Separation** colour space (object 5) that is intended to produce a colour named LogoGreen. If the output device has no colorant corresponding to this colour, **DeviceCMYK** is used as the alternate colour space, and the tint transformation function (object 12) maps tint values linearly into shades of a CMYK colour value approximating the LogoGreen colour.

```
5  0  obj                                            % Colour space
   [  /Separation
         /LogoGreen
         /DeviceCMYK
         12 0 R
   ]
endobj

12  0  obj                                           % Tint transformation function
   <<  /FunctionType  4
      /Domain  [0.0  1.0]
      /Range  [0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0]
      /Length  62
   >>

stream
   {  dup  0.84  mul
   exch  0.00  exch  dup  0.44  mul
   exch  0.21  mul
   }
endstream
endobj
```

See 11.7.3, "Spot Colours and Transparency", for further discussion of the role of **Separation** colour spaces in the transparent imaging model.

8.6.6.5 DeviceN Colour Spaces

DeviceN colour spaces *(PDF 1.3)* may contain an arbitrary number of colour components.

NOTE 1 They provide greater flexibility than is possible with standard device colour spaces such as **DeviceCMYK** or with individual **Separation** colour spaces.

EXAMPLE 1 It is possible to create a **DeviceN** colour space consisting of only the cyan, magenta, and yellow colour components, with the black component excluded.

NOTE 2 **DeviceN** colour spaces are used in applications such as these:

High-fidelity colour is the use of more than the standard CMYK process colorants to produce an extended gamut, or range of colours. A popular example is the PANTONE Hexachrome system, which uses six colorants: the usual cyan, magenta, yellow, and black, plus orange and green.

Multitone colour systems use a single-component image to specify multiple colour components. In a duotone, for example, a single-component image can be used to specify both the black component and a spot colour component. The tone reproduction is generally different for the different components. For example, the black component might be painted with the exact sample data from the single-component image; the spot colour component might be generated as a nonlinear function of the image data in a manner that emphasizes the shadows. Figure L.6 in Annex L shows an example that uses black and magenta colour components. In Figure L.7 in Annex L, a single-component grayscale image is used to generate a quadtone result that uses four colorants: black and three PANTONE spot colours. See EXAMPLE 5 in this sub-clause for the code used to generate this image.

DeviceN shall be used to represent colour spaces containing multiple components that correspond to colorants of some target device. As with **Separation** colour spaces, conforming readers shall be able to approximate the colorants if they are not available on the current output device, such as a display. To accomplish this, the colour space definition provides a tint transformation function that shall be used to convert all the components to an alternate colour space.

PDF 1.6 extended the meaning of **DeviceN** to include colour spaces that are referred to as ***NChannel** colour spaces*. Such colour spaces may contain an arbitrary number of spot and process components, which may or may not correspond to specific device colorants (the process components shall be from a single process colour space). They provide information about each component that allows conforming readers more flexibility in converting colours. These colour spaces shall be identified by a value of **NChannel** for the **Subtype** entry of the attributes dictionary (see Table 71). A value of **DeviceN** for the **Subtype** entry, or no value, shall mean that only the previous features shall be supported. Conforming readers that do not support PDF 1.6 shall treat these colour spaces as normal **DeviceN** colour spaces and shall use the tint transformation function as appropriate. Conforming writers using the **NChannel** features should follow certain guidelines, as noted throughout this sub-clause, to achieve good backward compatibility.

EXAMPLE 2 They may use their own blending algorithms for on-screen viewing and composite printing, rather than being required to use a specified tint transformation function.

DeviceN colour spaces shall be defined in a similar way to **Separation** colour spaces—in fact, a **Separation** colour space can be defined as a **DeviceN** colour space with only one component.

A **DeviceN** colour space shall be specified as follows:

[/DeviceN *names* *alternateSpace* *tintTransform*]

or

[/DeviceN *names* *alternateSpace* *tintTransform* *attributes*]

It is a four- or five-element array whose first element shall be the colour space family name **DeviceN**. The remaining elements shall be parameters that a **DeviceN** colour space requires.

The *names* parameter shall be an array of name objects specifying the individual colour components. The length of the array shall determine the number of components in the **DeviceN** colour space, which is subject to an implementation limit; see Annex C.The component names shall all be different from one another, except for the name **None**, which may be repeated as described later in this sub-clause. The special name **All**, used by **Separation** colour spaces, shall not be used.

Colour values shall be tint components in the range 0.0 to 1.0:

- For **DeviceN** colour spaces that do not have a subtype of **NChannel**, 0.0 shall represent the minimum amount of colorant; 1.0 shall represent the maximum. Tints shall always be treated as subtractive colours, even if the device produces output for the designated component by an additive method. Thus, a tint value of 0.0 shall denote the lightest colour that can be achieved with the given colorant, and 1.0 the darkest.

NOTE 3 This convention is the same one as for **DeviceCMYK** colour components but opposite to the one for **DeviceGray** and **DeviceRGB**.

- For **NChannel** colour spaces, values for additive process colours (such as *RGB*) shall be specified in their natural form, where 1.0 shall represent maximum intensity of colour.

When this space is set to the current colour space (using the **CS** or **cs** operators), each component shall be given an initial value of 1.0. The **SCN** and **scn** operators respectively shall set the current stroking and nonstroking colour. Operand values supplied to **SCN** or **scn** shall be interpreted as colour component values in the order in which the colours are given in the *names* array, as are the values in a sampled image that uses a **DeviceN** colour space.

The *alternateSpace* parameter shall be an array or name object that can be any device or CIE-based colour space but shall not be another special colour space (**Pattern**, **Indexed**, **Separation**, or **DeviceN**). When the colour space is set to a **DeviceN** space, if any of the component names in the colour space do not correspond to a colorant available on the device, the conforming reader shall perform subsequent painting operations in the alternate colour space specified by this parameter.

For **NChannel** colour spaces, the components shall be evaluated individually; that is, only the ones not present on the output device shall use the alternate colour space.

The *tintTransform* parameter shall specify a function (see 7.10, "Functions") that is used to transform the tint values into the alternate colour space. It shall be called with *n* tint values and returns *m* colour component values, where *n* is the number of components needed to specify a colour in the **DeviceN** colour space and *m* is the number required by the alternate colour space.

NOTE 4 Painting in the alternate colour space may produce a good approximation of the intended colour when only opaque objects are painted. However, it does not correctly represent the interactions between an object and its backdrop when the object is painted with transparency or when overprinting (see 8.6.7, "Overprint Control") is enabled.

The colour component name **None**, which may be present only for **DeviceN** colour spaces that do not have the **NChannel** subtype, indicates that the corresponding colour component shall never be painted on the page, as in a **Separation** colour space for the **None** colorant. When a **DeviceN** colour space is painting the named device colorants directly, colour components corresponding to **None** colorants shall be discarded. However, when the **DeviceN** colour space reverts to its alternate colour space, those components shall be passed to the tint transformation function, which may use them as desired.

A **DeviceN** colour space whose component colorant names are all **None** shall always discard its output, just the same as a **Separation** colour space for **None**; it shall never revert to the alternate colour space. Reversion shall occur only if at least one colour component (other than **None**) is specified and is not available on the device.

The optional *attributes* parameter shall be a dictionary (see Table 71) containing additional information about the components of colour space that conforming readers may use. Conforming readers need not use the *alternateSpace* and *tintTransform* parameters, and may instead use custom blending algorithms, along with other information provided in the attributes dictionary if present. (If the value of the **Subtype** entry in the attributes dictionary is **NChannel**, such information shall be present.) However, *alternateSpace* and *tintTransform* shall always be provided for conforming readers that want to use them or do not support PDF 1.6.

Table 71 – Entries in a DeviceN Colour Space Attributes Dictionary

Key	Type	Value
Subtype	name	*(Optional; PDF 1.6)* A name specifying the preferred treatment for the colour space. Values shall be **DeviceN** or **NChannel**. Default value: **DeviceN**.

Table 71 – Entries in a DeviceN Colour Space Attributes Dictionary (continued)

Key	Type	Value
Colorants	dictionary	*(Required if* ***Subtype*** *is* ***NChannel*** *and the colour space includes spot colorants; otherwise optional)* A dictionary describing the individual colorants that shall be used in the **DeviceN** colour space. For each entry in this dictionary, the key shall be a colorant name and the value shall be an array defining a **Separation** colour space for that colorant (see 8.6.6.4, "Separation Colour Spaces"). The key shall match the colorant name given in that colour space. This dictionary provides information about the individual colorants that may be useful to some conforming readers. In particular, the alternate colour space and tint transformation function of a **Separation** colour space describe the appearance of that colorant alone, whereas those of a **DeviceN** colour space describe only the appearance of its colorants in combination. If **Subtype** is **NChannel**, this dictionary shall have entries for all spot colorants in this colour space. This dictionary may also include additional colorants not used by this colour space.
Process	dictionary	*(Required if* ***Subtype*** *is* ***NChannel*** *and the colour space includes components of a process colour space, otherwise optional; PDF 1.6)* A dictionary (see Table 72) that describes the process colour space whose components are included in this colour space.
MixingHints	dictionary	*(Optional; PDF 1.6)* A dictionary (see Table 73) that specifies optional attributes of the inks that shall be used in blending calculations when used as an alternative to the tint transformation function.

A value of **NChannel** for the **Subtype** entry indicates that some of the other entries in this dictionary are required rather than optional. The **Colorants** entry specifies a *colorants dictionary* that contains entries for all the spot colorants in the colour space; they shall be defined using individual **Separation** colour spaces. The **Process** entry specifies a *process dictionary* (see Table 72) that identifies the process colour space that is used by this colour space and the names of its components. It shall be present if **Subtype** is **NChannel** and the colour space has process colour components. An **NChannel** colour space shall contain components from at most one process colour space.

For colour spaces that have a value of **NChannel** for the **Subtype** entry in the attributes dictionary (see Table 71), the following restrictions apply to process colours:

- There may be colour components from at most one process colour space, which may be any device or CIE-based colour space.

- For a non-*CMYK* colour space, the names of the process components shall appear sequentially in the *names* array, in the normal colour space order (for example, **Red**, **Green**, and **Blue**). However, the names in the *names* array need not match the actual colour space names (for example, a **Red** component need not be named **Red**).The mapping of names is specified in the process dictionary (see Table 72 and discussion below), which shall be present.

- Definitions for process colorants should not appear in the colorants dictionary. Any such definition shall be ignored if the colorant is also present in the process dictionary. Any component not specified in the process dictionary shall be considered to be a spot colorant.

- For a *CMYK* colour space, a subset of the components may be present, and they may appear in any order in the *names* array. The reserved names **Cyan**, **Magenta**, **Yellow**, and **Black** shall always be considered to be process colours, which do not necessarily correspond to the colorants of a specific device; they need not have entries in the process dictionary.

- The values associated with the process components shall be stored in their natural form (that is, subtractive colour values for *CMYK* and additive colour values for *RGB*), since they shall be interpreted directly as process values by consumers making use of the process dictionary. (For additive colour spaces, this is the reverse of how colour values are specified for **DeviceN**, as described above in the discussion of the *names* parameter.)

The **MixingHints** entry in the attributes dictionary specifies a *mixing hints dictionary* (see Table 73) that provides information about the characteristics of colorants that may be used in blending calculations when the actual colorants are not available on the target device. Conforming readers need not use this information.

Table 72 – Entries in a DeviceN Process Dictionary

KEY	TYPE	VALUE
ColorSpace	name or array	*(Required)* A name or array identifying the process colour space, which may be any device or CIE-based colour space. If an **ICCBased** colour space is specified, it shall provide calibration information appropriate for the process colour components specified in the *names* array of the **DeviceN** colour space.
Components	array	*(Required)* An array of component names that correspond, in order, to the components of the process colour space specified in **ColorSpace**. For example, an RGB colour space shall have three names corresponding to red, green, and blue. The names may be arbitrary (that is, not the same as the standard names for the colour space components) and shall match those specified in the *names* array of the **DeviceN** colour space, even if all components are not present in the *names* array.

Table 73 – Entries in a DeviceN Mixing Hints Dictionary

Key	Type	Value
Solidities	dictionary	*(Optional)* A dictionary specifying the solidity of inks that shall be used in blending calculations when used as an alternative to the tint transformation function. For each entry, the key shall be a colorant name, and the value shall be a number between 0.0 and 1.0. This dictionary need not contain entries for all colorants used in this colour space; it may also include additional colorants not used by this colour space. A value of 1.0 simulates an ink that completely covers the inks beneath; a value of 0.0 simulates a transparent ink that completely reveals the inks beneath. An entry with a key of **Default** specifies a value that shall be used by all components in the associated **DeviceN** colour space for which a solidity value is not explicitly provided. If **Default** is not present, the default value for unspecified colorants shall be 0.0; conforming readers may choose to use other values. If this entry is present, **PrintingOrder** shall also be present.
PrintingOrder	array	*(Required if* ***Solidities*** *is present)* An array of colorant names, specifying the order in which inks shall be laid down. Each component in the *names* array of the **DeviceN** colour space shall appear in this array (although the order is unrelated to the order specified in the *names* array). This entry may also list colorants unused by this specific **DeviceN** instance.

Table 73 – Entries in a DeviceN Mixing Hints Dictionary (continued)

Key	Type	Value
DotGain	dictionary	*(Optional)* A dictionary specifying the *dot gain* of inks that shall be used in blending calculations when used as an alternative to the tint transformation function. Dot gain (or loss) represents the amount by which a printer's halftone dots change as the ink spreads and is absorbed by paper. For each entry, the key shall be a colorant name, and the value shall be a function that maps values in the range 0 to 1 to values in the range 0 to 1. The dictionary may list colorants unused by this specific **DeviceN** instance and need not list all colorants. An entry with a key of **Default** shall specify a function to be used by all colorants for which a dot gain function is not explicitly specified. Conforming readers may ignore values in this dictionary when other sources of dot gain information are available, such as ICC profiles associated with the process colour space or tint transformation functions associated with individual colorants.

Each entry in the mixing hints dictionary refers to colorant names, which include spot colorants referenced by the **Colorants** dictionary. Under some circumstances, they may also refer to one or more individual process components called **Cyan**, **Magenta**, **Yellow**, or **Black** when **DeviceCMYK** is specified as the process colour space in the process dictionary. However, applications shall ignore these process component entries if they can obtain the information from an ICC profile.

NOTE 5 The mixing hints subdictionaries (as well as the colorants dictionary) may specify colorants that are not used in any given instance of a ***DeviceN*** colour space. This allows them to be referenced from multiple ***DeviceN*** colour spaces, which can produce smaller file sizes as well as consistent colour definitions across instances.

For consistency of colour, conforming readers should follow these guidelines:

- The conforming reader shall apply either the specified tint transformation function or invoke the same alternative blending algorithm for all ***DeviceN*** instances in the document.

NOTE 6 When the tint transformation function is used, the burden is on the conforming writer to guarantee that the individual function definitions chosen for all ***DeviceN*** instances produce similar colour appearances throughout the document.

- Blending algorithms should produce a similar appearance for colours when they are used as separation colours or as a component of a **DeviceN** colour space.

EXAMPLE 3 This example shows a **DeviceN** colour space consisting of three colour components named **Orange**, **Green**, and **None**. In this example, the **DeviceN** colour space, object 30, has an attributes dictionary whose **Colorants** entry is an indirect reference to object 45 (which might also be referenced by attributes dictionaries of other **DeviceN** colour spaces). *tintTransform1*, whose definition is not shown, maps three colour components (tints of the colorants **Orange**, **Green**, and **None**) to four colour components in the alternate colour space, **DeviceCMYK**. *tintTransform2* maps a single colour component (an orange tint) to four components in **DeviceCMYK**. Likewise, *tintTransform3* maps a green tint to **DeviceCMYK**, and *tintTransform4* maps a tint of PANTONE 131 to **DeviceCMYK**.

```
30 0 obj                                        % Colour space
   [ /DeviceN
         [/Orange /Green /None]
         /DeviceCMYK
         tintTransform1
         << /Colorants 45 0 R >>
   ]
endobj
```

EXAMPLE 4

```
45  0  obj                                          % Colorants dictionary
    <<   /Orange   [    /Separation
                        /Orange
                        /DeviceCMYK
                        tintTransform2
                   ]

        /Green  [  /Separation
                        /Green
                        /DeviceCMYK
                        tintTransform3
                ]
        /PANTONE#20131  [  /Separation
                                /PANTONE#20131
                                /DeviceCMYK
                                tintTransform4
                        ]
    >>
endobj
```

NOTE 7 EXAMPLE 5 through EXAMPLE 8 show the use of **NChannel** colour spaces.

EXAMPLE 5 This example shows the use of calibrated *CMYK* process components. EXAMPLE 6 shows the use of **Lab** process components.

```
10  0  obj                                      % Colour space
    [   /DeviceN
            [/Magenta /Spot1 /Yellow /Spot2]
            alternateSpace
            tintTransform1
            <<                                      % Attributes dictionary
                /Subtype /NChannel
                /Process
                    << /ColorSpace [/ICCBased CMYK_ICC profile ]
                        /Components [/Cyan /Magenta /Yellow /Black]
                    >>
                /Colorants
                    << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2]
                        /Spot2 [/Separation /Spot2 alternateSpace tintTransform3]
                    >>
            >>
    ]
endobj
```

EXAMPLE 6

```
10  0  obj                                          %Colour space
    [ /DeviceN
        [/L /a /b /Spot1 /Spot2]
        alternateSpace
        tintTransform1
        <<                                          % Attributes dictionary
            /Subtype /NChannel
            /Process
                << /ColorSpace [ /Lab << /WhitePoint ... /Range ... >> ]
                    /Components [/L /a /b]
                >>
            /Colorants
                << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2 ]
                    /Spot2 [/Separation /Spot2 alternateSpace tintTransform3]
                >>
        >>
    ]
```

EXAMPLE 7 This example shows the recommended convention for dealing with situations where a spot colorant and a process colour component have the same name. Since the *names* array may not have duplicate names, the process colours should be given different names, which are mapped to process components in the **Components** entry of the process dictionary. In this case, **Red** refers to a spot colorant; **ProcessRed**, **ProcessGreen**, and **ProcessBlue** are mapped to the components of an *RGB* colour space.

```
10  0  obj                                          % Colour space
[ /DeviceN
    [/ProcessRed /ProcessGreen /ProcessBlue /Red]
    alternateSpace
    tintTransform1
    <<                                              % Attributes dictionary
        /Subtype /NChannel
        /Process
            << /ColorSpace [ /ICCBased RGB_ICC profile ]
                /Components [/ProcessRed /ProcessGreen /ProcessBlue]
            >>
        /Colorants
            << /Red [/Separation /Red alternateSpace tintTransform2 ] >>
    >>
]
```

EXAMPLE 8 This example shows the use of a mixing hints dictionary.

```
10  0  obj                                          % Colour space
    [/DeviceN
        [/Magenta /Spot1 /Yellow /Spot2]
        alternateSpace
        tintTransform1
        <<
            /Subtype /NChannel
            /Process
                << /ColorSpace [ /ICCBased CMYK_ICC profile ]
                    /Components [/Cyan /Magenta /Yellow /Black]
                >>
            /Colorants
                << /Spot1 [/Separation /Spot1 alternateSpace tintTransform2 ]
                    /Spot2 [/Separation /Spot2 alternateSpace tintTransform2 ]
                >>
            /MixingHints
                <<
                    /Solidities
                        << /Spot1 1.0
                            /Spot2 0.0
                        >>
                    /DotGain
                        << /Spot1 function1
                            /Spot2 function2
                            /Magenta function3
                            /Yellow function4
                        >>
                    /PrintingOrder [/Magenta /Yellow /Spot1 /Spot2]
                >>
        >>
        ]
```

See 11.7.3, "Spot Colours and Transparency", for further discussion of the role of **DeviceN** colour spaces in the transparent imaging model.

8.6.6.6 Multitone Examples

NOTE 1 The following examples illustrate various interesting and useful special cases of the use of **Indexed** and **DeviceN** colour spaces in combination to produce multitone colours.

NOTE 2 EXAMPLE 1 and EXAMPLE 2 in this sub-clause illustrate the use of **DeviceN** to create duotone colour spaces.

EXAMPLE 1 In this example, an **Indexed** colour space maps index values in the range 0 to 255 to a duotone **DeviceN** space in cyan and black. In effect, the index values are treated as if they were tints of the duotone space, which are then mapped into tints of the two underlying colorants. Only the beginning of the lookup table string for the **Indexed** colour space is shown; the full table would contain 256 two-byte entries, each specifying a tint value for cyan and black, for a total of 512 bytes. If the alternate colour space of the **DeviceN** space is selected, the tint transformation function (object 15 in the example) maps the two tint components for cyan and black to the four components for a **DeviceCMYK** colour space by supplying zero values for the other two components.

```
10  0  obj                                          %Colour space
   [  /Indexed
         [ /DeviceN
               [/Cyan  /Black]
               /DeviceCMYK
               15 0 R
         ]

         255
         <6605  6806  6907  6B09  6C0A  …>
   ]
endobj

15  0  obj                                          % Tint transformation function
   <<  /FunctionType  4
      /Domain  [0.0  1.0   0.0  1.0]
      /Range  [0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0]
      /Length  16
   >>

stream
   {0  0  3  -1  roll}
endstream
endobj
```

EXAMPLE 2 This example shows the definition of another duotone colour space, this time using black and gold colorants (where gold is a spot colorant) and using a **CalRGB** space as the alternate colour space. This could be defined in the same way as in the preceding example, with a tint transformation function that converts from the two tint components to colours in the alternate **CalRGB** colour space.

```
30  0  obj                                          % Colour space
   [  /Indexed
         [ /DeviceN
               [/Black  /Gold]
               [    /CalRGB
                     <<  /WhitePoint  [1.0  1.0  1.0]
                        /Gamma  [2.2  2.2  2.2]
                     >>
               ]
               35 0 R                                  % Tint transformation function
         ]

         255
         …Lookup table…
   ]
endobj
```

NOTE 3 Given a formula for converting any combination of black and gold tints to calibrated *RGB*, a 2-in, 3-out type 4 (PostScript calculator) function could be used for the tint transformation. Alternatively, a type 0 (sampled) function could be used, but this would require a large number of sample points to represent the function accurately; for example, sampling each input variable for 256 tint values between 0.0 and 1.0 would require 256^2 = 65,536 samples. But since the **DeviceN** colour space is being used as the base of an **Indexed** colour space, there are actually only 256 possible combinations of black and gold tint values.

EXAMPLE 3 This example shows a more compact way to represent this information is to put the alternate colour values directly into the lookup table alongside the **DeviceN** colour values.

```
10  0  obj                                        % Colour space
    [  /Indexed
            [  /DeviceN
                    [/Black  /Gold  /None  /None  /None]
                    [    /CalRGB
                            <<  /WhitePoint  [1.0  1.0  1.0]
                                /Gamma  [2.2  2.2  2.2]
                            >>
                    ]
                    20 0 R                                    % Tint transformation function
            ]

            255
            …Lookup table…
    ]
endobj
```

NOTE 4 In EXAMPLE 3 in this sub-clause, each entry in the lookup table has *five* components: two for the black and gold colorants and three more (specified as **None**) for the equivalent **CalRGB** colour components. If the black and gold colorants are available on the output device, the **None** components are ignored; if black and gold are not available, the tint transformation function is used to convert a five-component colour into a three-component equivalent in the alternate **CalRGB** colour space. But because, by construction, the third, fourth, and fifth components are the **CalRGB** components, the tint transformation function can merely discard the first two components and return the last three. This can be readily expressed with a type 4 (PostScript calculator) function (see EXAMPLE 4 in this sub-clause).

EXAMPLE 4 This example shows a type 4 (PostScript calculator) function.

```
20  0  obj                                        % Tint transformation function
    <<  /FunctionType  4
        /Domain  [0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0]
        /Range  [0.0  1.0   0.0  1.0   0.0  1.0]
        /Length  27
    >>

stream
    {5  3  roll  pop  pop}
endstream
endobj
```

EXAMPLE 5 This example uses an extension of the techniques described above to produce the quadtone (four-component) image shown in Figure L.7 in Annex L.

```
5  0  obj                             % Image XObject
    <<  /Type  /XObject
        /Subtype  /Image
        /Width  288
        /Height  288
        /ColorSpace  10 0 R
        /BitsPerComponent  8
        /Length  105278
        /Filter  /ASCII85Decode
    >>

stream
…Data for grayscale image…
endstream
endobj

10  0  obj                            % Indexed colour space for image
    [  /Indexed
            15 0 R                        % Base colour space
            255                         % Table has 256 entries
            30 0 R                      % Lookup table
      ]
   endobj

   15  0  obj                        % Base colour space (DeviceN) for Indexed space
      [  /DeviceN
            [   /Black                  % Four colorants (black plus three spot colours)
            /PANTONE#20216#20CVC
            /PANTONE#20409#20CVC
            /PANTONE#202985#20CVC
            /None                       % Three components for alternate space
            /None
            /None
            ]

            16 0 R                      % Alternate colour space
            20 0 R                      % Tint transformation function
      ]
   endobj

   16  0  obj                        % Alternate colour space for DeviceN space
      [  /CalRGB
            <<  /WhitePoint  [1.0  1.0  1.0]  >>
      ]
   endobj

   20  0  obj                        % Tint transformation function for DeviceN space
      <<   /FunctionType  4
         /Domain  [0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0   0.0  1.0]
         /Range  [0.0  1.0   0.0  1.0   0.0  1.0]
         /Length  44
      >>

   stream
      {  7  3  roll                     % Just discard first four values
      pop  pop  pop  pop
      }
   endstream
   endobj

   30  0  obj                        % Lookup table for Indexed colour space
      <<   /Length  1975
         /Filter  [/ASCII85Decode  /FlateDecode]
      >>

   stream
   8;T1BB2"M7*!"psYBt1k\gY1T<D&tO]r*F7Hga*
   …Additional data (seven components for each table entry)…
   endstream
   endobj
```

NOTE 5 As in the preceding examples, an **Indexed** colour space based on a **DeviceN** space is used to paint the grayscale image shown on the left in the plate with four colorants: black and three PANTONE spot colours. The alternate colour space is a simple calibrated *RGB*. Thus, the **DeviceN** colour space has seven components: the four desired colorants plus the three components of the alternate space. The example shows the image XObject (see 8.9.5, "Image Dictionaries") representing the quadtone image, followed by the colour space used to interpret the image data.

8.6.7 Overprint Control

The graphics state contains an *overprint parameter*, controlled by the **OP** and **op** entries in a graphics state parameter dictionary. Overprint control is useful mainly on devices that produce true physical separations, but it is available on some composite devices as well. Although the operation of this parameter is device-dependent, it is described here rather than in the sub-clause on colour rendering, because it pertains to an aspect of painting in device colour spaces that is important to many applications.

Any painting operation marks some specific set of device colorants, depending on the colour space in which the painting takes place. In a **Separation** or **DeviceN** colour space, the colorants to be marked shall be specified explicitly; in a device or CIE-based colour space, they shall be implied by the process colour model of the output device (see clause 10, "Rendering"). The overprint parameter is a boolean flag that determines how painting operations affect colorants other than those explicitly or implicitly specified by the current colour space.

If the overprint parameter is **false** (the default value), painting a colour in any colour space shall cause the corresponding areas of unspecified colorants to be erased (painted with a tint value of 0.0). The effect is that the colour at any position on the page is whatever was painted there last, which is consistent with the normal painting behaviour of the opaque imaging model.

If the overprint parameter is **true** and the output device supports overprinting, erasing actions shall not be performed; anything previously painted in other colorants is left undisturbed. Consequently, the colour at a given position on the page may be a combined result of several painting operations in different colorants. The effect produced by such overprinting is device-dependent and is not defined here.

NOTE 1 Not all devices support overprinting. Furthermore, many PostScript printers support it only when separations are being produced, and not for composite output.

If overprinting is not supported, the value of the overprint parameter shall be ignored.

An additional graphics state parameter, the *overprint mode (PDF 1.3)*, shall affect the interpretation of a tint value of 0.0 for a colour component in a **DeviceCMYK** colour space when overprinting is enabled. This parameter is controlled by the **OPM** entry in a graphics state parameter dictionary; it shall have an effect only when the overprint parameter is **true**, as described above.

When colours are specified in a **DeviceCMYK** colour space and the native colour space of the output device is also **DeviceCMYK**, each of the source colour components controls the corresponding device colorant directly. Ordinarily, each source colour component value replaces the value previously painted for the corresponding device colorant, no matter what the new value is; this is the default behaviour, specified by overprint mode 0.

When the overprint mode is 1 (also called *nonzero overprint mode*), a tint value of 0.0 for a source colour component shall leave the corresponding component of the previously painted colour unchanged. The effect is equivalent to painting in a **DeviceN** colour space that includes only those components whose values are nonzero.

EXAMPLE If the overprint parameter is **true** and the overprint mode is 1, the operation

```
0.2 0.3 0.0 1.0 k
```

is equivalent to

```
0.2 0.3 1.0 scn
```

in the colour space shown in this example.

```
10 0 obj                                          %Colour space
   [ /DeviceN
         [/Cyan /Magenta /Black]
         /DeviceCMYK
         15 0 R
   ]
endobj

15 0 obj                                          % Tint transformation function
      << /FunctionType 4
         /Domain [0.0 1.0  0.0 1.0  0.0 1.0]
         /Range [0.0 1.0  0.0 1.0  0.0 1.0  0.0 1.0]
```

```
            /Length 13
         >>

stream
   {0 exch}
endstream
endobj
```

Nonzero overprint mode shall apply only to painting operations that use the current colour in the graphics state when the current colour space is **DeviceCMYK** (or is implicitly converted to **DeviceCMYK**; see 8.6.5.7, "Implicit Conversion of CIE-Based Colour Spaces"). It shall not apply to the painting of images or to any colours that are the result of a computation, such as those in a shading pattern or conversions from some other colour space. It also shall not apply if the device's native colour space is not **DeviceCMYK**; in that case, source colours shall be converted to the device's native colour space, and all components participate in the conversion, whatever their values.

NOTE 2 This is shown explicitly in the alternate colour space and tint transformation function of the **DeviceN** colour space (see EXAMPLE 3 in 8.6.6, "Special Colour Spaces").

See 11.7.4, "Overprinting and Transparency", for further discussion of the role of overprinting in the transparent imaging model.

8.6.8 Colour Operators

Table 74 lists the PDF operators that control colour spaces and colour values. Also colour-related is the graphics state operator **ri**, listed in Table 57 and discussed under 8.6.5.8, "Rendering Intents". Colour operators may appear at the page description level or inside text objects (see Figure 9 in Annex L).

Table 74 – Colour Operators

Operands	Operator	Description
name	**CS**	*(PDF 1.1)* Set the current colour space to use for stroking operations. The operand *name* shall be a name object. If the colour space is one that can be specified by a name and no additional parameters (**DeviceGray**, **DeviceRGB**, **DeviceCMYK**, and certain cases of **Pattern**), the name may be specified directly. Otherwise, it shall be a name defined in the **ColorSpace** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"); the associated value shall be an array describing the colour space (see 8.6.3, "Colour Space Families"). The names ***DeviceGray***, ***DeviceRGB***, ***DeviceCMYK***, and ***Pattern*** always identify the corresponding colour spaces directly; they never refer to resources in the ***ColorSpace*** subdictionary. The **CS** operator shall also set the current stroking colour to its initial value, which depends on the colour space: In a **DeviceGray**, **DeviceRGB**, **CalGray**, or **CalRGB** colour space, the initial colour shall have all components equal to 0.0. In a **DeviceCMYK** colour space, the initial colour shall be [0.0 0.0 0.0 1.0]. In a **Lab** or **ICCBased** colour space, the initial colour shall have all components equal to 0.0 unless that falls outside the intervals specified by the space's **Range** entry, in which case the nearest valid value shall be substituted. In an **Indexed** colour space, the initial colour value shall be 0. In a **Separation** or **DeviceN** colour space, the initial tint value shall be 1.0 for all colorants. In a **Pattern** colour space, the initial colour shall be a pattern object that causes nothing to be painted.
name	**cs**	*(PDF 1.1)* Same as **CS** but used for nonstroking operations.

Table 74 – Colour Operators (continued)

Operands	Operator	Description
$c_1 \ldots c_n$	**SC**	*(PDF 1.1)* Set the colour to use for stroking operations in a device, CIE-based (other than **ICCBased**), or **Indexed** colour space. The number of operands required and their interpretation depends on the current stroking colour space: For **DeviceGray**, **CalGray**, and **Indexed** colour spaces, one operand shall be required ($n = 1$). For **DeviceRGB**, **CalRGB**, and **Lab** colour spaces, three operands shall be required ($n = 3$). For **DeviceCMYK**, four operands shall be required ($n = 4$).
$c_1 \ldots c_n$ $c_1 \ldots c_n$ *name*	**SCN** **SCN**	*(PDF 1.2)* Same as **SC** but also supports **Pattern, Separation, DeviceN** and **ICCBased** colour spaces. If the current stroking colour space is a **Separation**, **DeviceN**, or **ICCBased** colour space, the operands $c_1 \ldots c_n$ shall be numbers. The number of operands and their interpretation depends on the colour space. If the current stroking colour space is a **Pattern** colour space, *name* shall be the name of an entry in the **Pattern** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). For an uncoloured tiling pattern (**PatternType** = 1 and **PaintType** = 2), $c_1 \ldots c_n$ shall be component values specifying a colour in the pattern's underlying colour space. For other types of patterns, these operands shall not be specified.
$c_1 \ldots c_n$	**sc**	*(PDF 1.1)* Same as **SC** but used for nonstroking operations.
$c_1 \ldots c_n$ $c_1 \ldots c_n$ *name*	**scn** **scn**	*(PDF 1.2)* Same as **SCN** but used for nonstroking operations.
gray	**G**	Set the stroking colour space to **DeviceGray** (or the **DefaultGray** colour space; see 8.6.5.6, "Default Colour Spaces") and set the gray level to use for stroking operations. *gray* shall be a number between 0.0 (black) and 1.0 (white).
gray	**g**	Same as **G** but used for nonstroking operations.
r g b	**RG**	Set the stroking colour space to **DeviceRGB** (or the **DefaultRGB** colour space; see 8.6.5.6, "Default Colour Spaces") and set the colour to use for stroking operations. Each operand shall be a number between 0.0 (minimum intensity) and 1.0 (maximum intensity).
r g b	**rg**	Same as **RG** but used for nonstroking operations.
c m y k	**K**	Set the stroking colour space to **DeviceCMYK** (or the **DefaultCMYK** colour space; see 8.6.5.6, "Default Colour Spaces") and set the colour to use for stroking operations. Each operand shall be a number between 0.0 (zero concentration) and 1.0 (maximum concentration). The behaviour of this operator is affected by the overprint mode (see 8.6.7, "Overprint Control").
c m y k	**k**	Same as **K** but used for nonstroking operations.

Invoking operators that specify colours or other colour-related parameters in the graphics state is restricted in certain circumstances. This restriction occurs when defining graphical figures whose colours shall be specified separately each time they are used. Specifically, the restriction applies in these circumstances:

- In any glyph description that uses the **d1** operator (see 9.6.5, "Type 3 Fonts")
- In the content stream of an uncoloured tiling pattern (see 8.7.3.3, "Uncoloured Tiling Patterns")

In these circumstances, the following actions cause an error:

- Invoking any of the following operators:

CS	**scn**	**K**
cs	**G**	**k**
SC	**g**	**ri**
SCN	**RG**	**sh**
sc	**rg**	

- Invoking the **gs** operator with any of the following entries in the graphics state parameter dictionary:

TR	**BG**	**UCR**
TR2	**BG2**	**UCR2**
HT		

- Painting an image. However, painting an *image mask* (see 8.9.6.2, "Stencil Masking") shall be permitted because it does not specify colours; instead, it designates places where the current colour shall be painted.

8.7 Patterns

8.7.1 General

Patterns come in two varieties:

- *Tiling patterns* consist of a small graphical figure (called a *pattern cell*) that is replicated at fixed horizontal and vertical intervals to fill the area to be painted. The graphics objects to use for tiling shall be described by a content stream.

- *Shading patterns* define a *gradient fill* that produces a smooth transition between colours across the area. The colour to use shall be specified as a function of position using any of a variety of methods.

NOTE 1 When operators such as **S** (stroke), **f** (fill), and **Tj** (show text) paint an area of the page with the current colour, they ordinarily apply a single colour that covers the area uniformly. However, it is also possible to apply "paint" that consists of a repeating graphical figure or a smoothly varying colour gradient instead of a simple colour. Such a repeating figure or smooth gradient is called a *pattern.* Patterns are quite general, and have many uses; for example, they can be used to create various graphical textures, such as weaves, brick walls, sunbursts, and similar geometrical and chromatic effects.

NOTE 2 Older techniques such as defining a pattern by using character glyphs in a special font and painting them repeatedly with the ***Tj*** operator should not be used. Another technique, defining patterns as halftone screens, should not be used because the effects produced are device-dependent.

Patterns shall be specified in a special family of colour spaces named **Pattern.** These spaces shall use *pattern objects* as the equivalent of colour values instead of the numeric component values used with other spaces. A pattern object shall be a dictionary or a stream, depending on the type of pattern; the term *pattern dictionary* is used generically throughout this sub-clause to refer to either a dictionary object or the dictionary portion of a stream object. (Those pattern objects that are streams are specifically identified as such in the descriptions of particular pattern types; unless otherwise stated, they are understood to be simple dictionaries instead.) This sub-clause describes **Pattern** colour spaces and the specification of colour values within them.

NOTE 3 See 8.6, "Colour Spaces", for information about colour spaces and colour values in general and 11.6.7, "Patterns and Transparency", for further discussion of the treatment of patterns in the transparent imaging model.

8.7.2 General Properties of Patterns

A pattern dictionary contains descriptive information defining the appearance and properties of a pattern. All pattern dictionaries shall contain an entry named **PatternType**, whose value identifies the kind of pattern the dictionary describes: type 1 for a tiling pattern or type 2 for a shading pattern. The remaining contents of the dictionary depend on the pattern type and are detailed in the sub-clauses on individual pattern types.

All patterns shall be treated as colours; a **Pattern** colour space shall be established with the **CS** or **cs** operator just like other colour spaces, and a particular pattern shall be installed as the current colour with the **SCN** or **scn** operator (see Table 74).

A pattern's appearance is described with respect to its own internal coordinate system. Every pattern has a *pattern matrix*, a transformation matrix that maps the pattern's internal coordinate system to the default coordinate system of the pattern's *parent content stream* (the content stream in which the pattern is defined as a resource). The concatenation of the pattern matrix with that of the parent content stream establishes the *pattern coordinate space*, within which all graphics objects in the pattern shall be interpreted.

NOTE 1 If a pattern is used on a page, the pattern appears in the **Pattern** subdictionary of that page's resource dictionary, and the pattern matrix maps pattern space to the default (initial) coordinate space of the page. Changes to the page's transformation matrix that occur within the page's content stream, such as rotation and scaling, have no effect on the pattern; it maintains its original relationship to the page no matter where on the page it is used. Similarly, if a pattern is used within a form XObject (see 8.10, "Form XObjects"), the pattern matrix maps pattern space to the form's default user space (that is, the form coordinate space at the time the form is painted with the **Do** operator). A pattern may be used within another pattern; the inner pattern's matrix defines its relationship to the pattern space of the outer pattern.

NOTE 2 PostScript allows a pattern to be defined in one context but used in another. For example, a pattern might be defined on a page (that is, its pattern matrix maps the pattern coordinate space to the user space of the page) but be used in a form on that page, so that its relationship to the page is independent of each individual placement of the form. PDF does not support this feature; in PDF, all patterns shall be local to the context in which they are defined.

8.7.3 Tiling Patterns

8.7.3.1 General

A *tiling pattern* consists of a small graphical figure called a *pattern cell*. Painting with the pattern replicates the cell at fixed horizontal and vertical intervals to fill an area. The effect is as if the figure were painted on the surface of a clear glass tile, identical copies of which were then laid down in an array covering the area and trimmed to its boundaries. This process is called *tiling* the area.

The pattern cell can include graphical elements such as filled areas, text, and sampled images. Its shape need not be rectangular, and the spacing of tiles can differ from the dimensions of the cell itself. When performing painting operations such as **S** (stroke) or **f** (fill), the conforming reader shall paint the cell on the current page as many times as necessary to fill an area. The order in which individual tiles (instances of the cell) are painted is unspecified and unpredictable; figures on adjacent tiles should not overlap.

The appearance of the pattern cell shall be defined by a content stream containing the painting operators needed to paint one instance of the cell. Besides the usual entries common to all streams (see Table 5), this stream's dictionary may contain the additional entries listed in Table 75.

Table 75 – Additional Entries Specific to a Type 1 Pattern Dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Pattern** for a pattern dictionary.
PatternType	integer	*(Required)* A code identifying the type of pattern that this dictionary describes; shall be 1 for a tiling pattern.

Table 75 – Additional Entries Specific to a Type 1 Pattern Dictionary (continued)

Key	Type	Value
PaintType	integer	*(Required)* A code that determines how the colour of the pattern cell shall be specified: a) *Coloured tiling pattern*. The pattern's content stream shall specify the colours used to paint the pattern cell. When the content stream begins execution, the current colour is the one that was initially in effect in the pattern's parent content stream. This is similar to the definition of the pattern matrix; see 8.7.2, "General Properties of Patterns". b) *Uncoloured tiling pattern*. The pattern's content stream shall not specify any colour information. Instead, the entire pattern cell is painted with a separately specified colour each time the pattern is used. Essentially, the content stream describes a *stencil* through which the current colour shall be poured. The content stream shall not invoke operators that specify colours or other colour-related parameters in the graphics state; otherwise, an error occurs (see 8.6.8, "Colour Operators"). The content stream may paint an image mask, however, since it does not specify any colour information (see 8.9.6.2, "Stencil Masking").
TilingType	integer	*(Required)* A code that controls adjustments to the spacing of tiles relative to the device pixel grid: a) *Constant spacing*. Pattern cells shall be spaced consistently—that is, by a multiple of a device pixel. To achieve this, the conforming reader may need to distort the pattern cell slightly by making small adjustments to **XStep**, **YStep**, and the transformation matrix. The amount of distortion shall not exceed 1 device pixel. b) *No distortion*. The pattern cell shall not be distorted, but the spacing between pattern cells may vary by as much as 1 device pixel, both horizontally and vertically, when the pattern is painted. This achieves the spacing requested by **XStep** and **YStep** *on average* but not necessarily for each individual pattern cell. c) *Constant spacing and faster tiling*. Pattern cells shall be spaced consistently as in tiling type 1 but with additional distortion permitted to enable a more efficient implementation.
BBox	rectangle	*(Required)* An array of four numbers in the pattern coordinate system giving the coordinates of the left, bottom, right, and top edges, respectively, of the pattern cell's bounding box. These boundaries shall be used to clip the pattern cell.
XStep	number	*(Required)* The desired horizontal spacing between pattern cells, measured in the pattern coordinate system.
YStep	number	*(Required)* The desired vertical spacing between pattern cells, measured in the pattern coordinate system. NOTE **XStep** and **YStep** may differ from the dimensions of the pattern cell implied by the **BBox** entry. This allows tiling with irregularly shaped figures. **XStep** and **YStep** may be either positive or negative but shall not be zero.
Resources	dictionary	*(Required)* A resource dictionary that shall contain all of the named resources required by the pattern's content stream (see 7.8.3, "Resource Dictionaries").
Matrix	array	*(Optional)* An array of six numbers specifying the pattern matrix (see 8.7.2, "General Properties of Patterns"). Default value: the identity matrix [1 0 0 1 0 0].

The pattern dictionary's **BBox**, **XStep**, and **YStep** values shall be interpreted in the pattern coordinate system, and the graphics objects in the pattern's content stream shall be defined with respect to that coordinate system. The placement of pattern cells in the tiling is based on the location of one *key pattern cell*, which is then displaced by multiples of **XStep** and **YStep** to replicate the pattern. The origin of the key pattern cell coincides with the origin of the pattern coordinate system. The phase of the tiling can be controlled by the translation components of the **Matrix** entry in the pattern dictionary.

Prior to painting with a tiling pattern, the conforming writer shall establish the pattern as the current colour in the graphics state. Subsequent painting operations tile the painted areas with the pattern cell described by the pattern's content stream. To obtain the pattern cell, the conforming reader shall perform these steps:

a) Saves the current graphics state (as if by invoking the **q** operator)

b) Installs the graphics state that was in effect at the beginning of the pattern's parent content stream, with the current transformation matrix altered by the pattern matrix as described in 8.7.2, "General Properties of Patterns"

c) Paints the graphics objects specified in the pattern's content stream

d) Restores the saved graphics state (as if by invoking the **Q** operator)

NOTE The pattern's content stream should not set any of the device-dependent parameters in the graphics state (see Table 53) because it may result in incorrect output.

8.7.3.2 Coloured Tiling Patterns

A *coloured tiling pattern* is a pattern whose colour is self-contained. In the course of painting the pattern cell, the pattern's content stream explicitly sets the colour of each graphical element it paints. A single pattern cell may contain elements that are painted different colours; it may also contain sampled grayscale or colour images. This type of pattern is identified by a pattern type of 1 and a paint type of 1 in the pattern dictionary.

When the current colour space is a **Pattern** space, a coloured tiling pattern shall be selected as the current colour by supplying its name as the single operand to the **SCN** or **scn** operator. This name shall be the key of an entry in the **Pattern** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"), whose value shall be the stream object representing the pattern. Since the pattern defines its own colour information, no additional operands representing colour components shall be specified to **SCN** or **scn**.

EXAMPLE 1 If P1 is the name of a pattern resource in the current resource dictionary, the following code establishes it as the current nonstroking colour:

```
/Pattern cs
/P1 scn
```

NOTE 1 Subsequent executions of nonstroking painting operators, such as **f** (fill), **Tj** (show text), or **Do** (paint external object) with an image mask, use the designated pattern to tile the areas to be painted.

NOTE 2 The following defines a page (object 5) that paints three circles and a triangle using a coloured tiling pattern (object 15) over a yellow background. The pattern consists of the symbols for the four suits of playing cards (spades, hearts, diamonds, and clubs), which are character glyphs taken from the ZapfDingbats font (see D.6, "ZapfDingbats Set and Encoding"); the pattern's content stream specifies the colour of each glyph. Figure L.8 in Annex L shows the results.

EXAMPLE 2

```
5  0  obj                                          % Page object
   <<  /Type  /Page
       /Parent  2 0 R
       /Resources  10 0 R
       /Contents  30 0 R
       /CropBox  [0  0  225  225]
   >>
endobj

10  0  obj                                       % Resource dictionary for page
   <<  /Pattern  <<  /P1  15 0 R  >>
```

```
    >>
endobj

15  0  obj                                            % Pattern definition
    <<  /Type  /Pattern
        /PatternType  1                                   % Tiling pattern
        /PaintType  1                                     % Coloured
        /TilingType  2
        /BBox  [0  0  100  100]
        /XStep  100
        /YStep  100
        /Resources  16 0 R
        /Matrix  [0.4  0.0  0.0  0.4  0.0  0.0]
        /Length  183
    >>

stream
BT                                                    % Begin text object
    /F1  1  Tf                                        % Set text font and size
    64  0  0  64  7.1771  2.4414  Tm                  % Set text matrix
    0  Tc                                             % Set character spacing
    0  Tw                                             % Set word spacing

    1.0  0.0  0.0  rg                                 % Set nonstroking colour to red
    (\001)  Tj                                        % Show spade glyph

    0.7478  -0.007  TD                                % Move text position
    0.0  1.0  0.0  rg                                 % Set nonstroking colour to green
    (\002)  Tj                                        % Show heart glyph

    -0.7323  0.7813  TD                               % Move text position
    0.0  0.0  1.0  rg                                 % Set nonstroking colour to blue
    (\003)  Tj                                        % Show diamond glyph

    0.6913  0.007  TD                                 % Move text position
    0.0  0.0  0.0  rg                                 % Set nonstroking colour to black
    (\004)  Tj                                        % Show club glyph

ET                                                    % End text object
endstream
endobj

16  0  obj                                            % Resource dictionary for pattern
    <<  /Font  <<  /F1  20 0 R  >>
    >>
endobj

20  0  obj                                            % Font for pattern
    <<  /Type  /Font
        /Subtype  /Type1
        /Encoding  21 0 R
        /BaseFont  /ZapfDingbats
    >>
endobj

21  0  obj                                            % Font encoding
    <<  /Type  /Encoding
        /Differences  [1  /a109  /a110  /a111  /a112]
    >>
endobj

30  0  obj                                            % Contents of page
    <<  /Length  1252  >>
stream
    0.0  G                                                % Set stroking colour to black
```

```
        1.0  1.0  0.0  rg                               % Set nonstroking colour to yellow
        25  175  175  -150  re                          % Construct rectangular path
        f                                               % Fill path

        /Pattern  cs                                    % Set pattern colour space
        /P1  scn                                        % Set pattern as nonstroking colour

        99.92  49.92  m                                 % Start new path
        99.92  77.52  77.52  99.92  49.92  99.92  c     % Construct lower-left circle
        22.32  99.92  -0.08  77.52  -0.08  49.92  c
        -0.08  22.32  22.32  -0.08  49.92  -0.08  c
        77.52  -0.08  99.92  22.32  99.92  49.92  c
        B                                               % Fill and stroke path

        224.96  49.92  m                                % Start new path
        224.96  77.52  202.56  99.92  174.96  99.92  c  % Construct lower-right circle
        147.36  99.92  124.96  77.52  124.96  49.92  c
        124.96  22.32  147.36  -0.08  174.96  -0.08  c
        202.56  -0.08  224.96  22.32  224.96  49.92  c
        B                                               % Fill and stroke path

        87.56  201.70  m                                % Start new path
        63.66  187.90  55.46  157.32  69.26  133.40  c  % Construct upper circle
        83.06  109.50  113.66  101.30  137.56  115.10  c
        161.46  128.90  169.66  159.50  155.86  183.40  c
        142.06  207.30  111.46  215.50  87.56  201.70  c
        B                                               % Fill and stroke path

        50  50  m                                       % Start new path
        175  50  l                                      % Construct triangular path
        112.5  158.253  l
        b                                               % Close, fill, and stroke path
    endstream
    endobj
```

NOTE 3 Several features of EXAMPLE 2 in this sub-clause are noteworthy:

The three circles and the triangle are painted with the same pattern. The pattern cells align, even though the circles and triangle are not aligned with respect to the pattern cell. For example, the position of the blue diamonds varies relative to the three circles.

The pattern cell does not completely cover the tile: it leaves the spaces between the glyphs unpainted. When the tiling pattern is used as a colour, the existing background (the yellow rectangle) shows through these unpainted areas.

8.7.3.3 Uncoloured Tiling Patterns

An *uncoloured tiling pattern* is a pattern that has no inherent colour: the colour shall be specified separately whenever the pattern is used. It provides a way to tile different regions of the page with pattern cells having the same shape but different colours. This type of pattern shall be identified by a pattern type of 1 and a paint type of 2 in the pattern dictionary. The pattern's content stream shall not explicitly specify any colours; it may paint an image mask (see 8.9.6.2, "Stencil Masking") but no other kind of image.

A **Pattern** colour space representing an uncoloured tiling pattern shall have a parameter: an object identifying the *underlying colour space* in which the actual colour of the pattern shall be specified. The underlying colour space shall be given as the second element of the array that defines the **Pattern** colour space.

EXAMPLE 1 The array

```
[/Pattern  /DeviceRGB]
```

defines a **Pattern** colour space with **DeviceRGB** as its underlying colour space.

NOTE The underlying colour space cannot be another ***Pattern*** colour space.

Operands supplied to the **SCN** or **scn** operator in such a colour space shall include a colour value in the underlying colour space, specified by one or more numeric colour components, as well as the name of a pattern object representing an uncoloured tiling pattern.

EXAMPLE 2 If the current resource dictionary (see 7.8.3, "Resource Dictionaries") defines Cs3 as the name of a **ColorSpace** resource whose value is the **Pattern** colour space shown above and P2 as a **Pattern** resource denoting an uncoloured tiling pattern, the code

```
/Cs3  cs
0.30  0.75  0.21  /P2  scn
```

establishes Cs3 as the current nonstroking colour space and P2 as the current nonstroking colour, to be painted in the colour represented by the specified components in the **DeviceRGB** colour space. Subsequent executions of nonstroking painting operators, such as **f** (fill), **Tj** (show text), and **Do** (paint external object) with an image mask, use the designated pattern and colour to tile the areas to be painted. The same pattern can be used repeatedly with a different colour each time.

EXAMPLE 3 This example is similar to EXAMPLE 2 in 8.7.3.2, except that it uses an uncoloured tiling pattern to paint the three circles and the triangle, each in a different colour (see Figure L.9 in Annex L). To do so, it supplies four operands each time it invokes the scn operator: three numbers denoting the colour components in the underlying **DeviceRGB** colour space, along with the name of the pattern.

```
5  0  obj                                   % Page object
   <<  /Type  /Page
       /Parent  2 0 R
       /Resources  10 0 R
       /Contents  30 0 R
       /CropBox  [0  0  225  225]
   >>
endobj

10  0  obj                                  % Resource dictionary for page
   <<  /ColorSpace  <<  /Cs12  12 0 R  >>
       /Pattern  <<  /P1  15 0 R  >>
   >>
endobj

12  0  obj                                  %Colour space
   [/Pattern  /DeviceRGB]
endobj

15  0  obj                                  % Pattern definition
   <<  /Type  /Pattern
       /PatternType  1                         % Tiling pattern
       /PaintType  2                           % Uncoloured
       /TilingType  2
       /BBox  [0  0  100  100]
       /XStep  100
       /YStep  100
       /Resources  16 0 R
       /Matrix  [0.4  0.0  0.0  0.4  0.0  0.0]
       /Length  127
   >>

stream
   BT                                          % Begin text object
       /F1  1  Tf                              % Set text font and size
       64  0  0  64  7.1771  2.4414  Tm        % Set text matrix
       0  Tc                                   % Set character spacing
       0  Tw                                   % Set word spacing

       (\001)  Tj                              % Show spade glyph
```

```
            0.7478  -0.007  TD                              % Move text position
            (\002)  Tj                                      % Show heart glyph

            -0.7323  0.7813  TD                             % Move text position
            (\003)  Tj                                      % Show diamond glyph

            0.6913  0.007  TD                               % Move text position
            (\004)  Tj                                      % Show club glyph
        ET                                                  % End text object
    endstream
    endobj

    16  0  obj                                          % Resource dictionary for pattern
        <<   /Font  <<  /F1  20 0 R  >>
        >>
    endobj

    20  0  obj                                          % Font for pattern
        <<  /Type  /Font
            /Subtype  /Type1
            /Encoding  21 0 R
            /BaseFont  /ZapfDingbats
        >>
    endobj

    21  0  obj                                          % Font encoding
        <<  /Type  /Encoding
            /Differences  [1  /a109  /a110  /a111  /a112]
        >>
    endobj

    30  0  obj                                          % Contents of page
        <<  /Length  1316  >>
    stream
        0.0  G                                              % Set stroking colour to black
        1.0  1.0  0.0  rg                                   % Set nonstroking colour to yellow
        25  175  175  -150  re                              % Construct rectangular path
        f                                                   % Fill path

        /Cs12  cs                                           % Set pattern colour space
        0.77  0.20  0.00  /P1  scn                          % Set nonstroking colour and pattern
        99.92  49.92  m                                     % Start new path
        99.92  77.52  77.52  99.92  49.92  99.92  c         % Construct lower-left circle
        22.32  99.92  -0.08  77.52  -0.08  49.92  c
        -0.08  22.32  22.32  -0.08  49.92  -0.08  c
        77.52  -0.08  99.92  22.32  99.92  49.92  c

        B                                                   % Fill and stroke path

        0.2  0.8  0.4  /P1  scn                             % Change nonstroking colour
        224.96  49.92  m                                    % Start new path
        224.96  77.52  202.56  99.92  174.96  99.92  c      % Construct lower-right circle
        147.36  99.92  124.96  77.52  124.96  49.92  c
        124.96  22.32  147.36  -0.08  174.96  -0.08  c
        202.56  -0.08  224.96  22.32  224.96  49.92  c
        B                                                   % Fill and stroke path

        0.3  0.7  1.0  /P1  scn                             % Change nonstroking colour
        87.56  201.70  m                                    % Start new path
        63.66  187.90  55.46  157.30  69.26  133.40  c      % Construct upper circle
        83.06  109.50  113.66  101.30  137.56  115.10  c
        161.46  128.90  169.66  159.50  155.86  183.40  c
        142.06  207.30  111.46  215.50  87.56  201.70  c
        B                                                   % Fill and stroke path
    0.5  0.2  1.0  /P1  scn                      % Change nonstroking colour
    50  50  m                                    % Start new path
    175  50  l                                   % Construct triangular path
    112.5  158.253  l
    b                                            % Close, fill, and stroke path
endstream
endobj
```

8.7.4 Shading Patterns

8.7.4.1 General

Shading patterns (PDF 1.3) provide a smooth transition between colours across an area to be painted, independent of the resolution of any particular output device and without specifying the number of steps in the colour transition. Patterns of this type shall be described by pattern dictionaries with a pattern type of 2. Table 76 shows the contents of this type of dictionary.

Table 76 – Entries in a Type 2 Pattern Dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Pattern** for a pattern dictionary.
PatternType	integer	*(Required)* A code identifying the type of pattern that this dictionary describes; shall be 2 for a shading pattern.
Shading	dictionary or stream	*(Required)* A shading object (see below) defining the shading pattern's gradient fill. The contents of the dictionary shall consist of the entries in Table 78 and those in one of Tables 79 to 84.
Matrix	array	*(Optional)* An array of six numbers specifying the pattern matrix (see 8.7.2, "General Properties of Patterns"). Default value: the identity matrix [1 0 0 1 0 0].
ExtGState	dictionary	*(Optional)* A graphics state parameter dictionary (see 8.4.5, "Graphics State Parameter Dictionaries") containing graphics state parameters to be put into effect temporarily while the shading pattern is painted. Any parameters that are so specified shall be inherited from the graphics state that was in effect at the beginning of the content stream in which the pattern is defined as a resource.

The most significant entry is **Shading**, whose value shall be a *shading object* defining the properties of the shading pattern's *gradient fill*. This is a complex "paint" that determines the type of colour transition the shading pattern produces when painted across an area. A shading object shall be a dictionary or a stream, depending on the type of shading; the term *shading dictionary* is used generically throughout this sub-clause to refer to either a dictionary object or the dictionary portion of a stream object. (Those shading objects that are streams are specifically identified as such in the descriptions of particular shading types; unless otherwise stated, they are understood to be simple dictionaries instead.)

By setting a shading pattern as the current colour in the graphics state, a PDF content stream may use it with painting operators such as **f** (fill), **S** (stroke), **Tj** (show text), or **Do** (paint external object) with an image mask to paint a path, character glyph, or mask with a smooth colour transition. When a shading is used in this way, the geometry of the gradient fill is independent of that of the object being painted.

8.7.4.2 Shading Operator

When the area to be painted is a relatively simple shape whose geometry is the same as that of the gradient fill itself, the **sh** operator may be used instead of the usual painting operators. **sh** accepts a shading dictionary as an operand and applies the corresponding gradient fill directly to current user space. This operator does not require the creation of a pattern dictionary or a path and works without reference to the current colour in the graphics state. Table 77 describes the **sh** operator.

NOTE Patterns defined by type 2 pattern dictionaries do not tile. To create a tiling pattern containing a gradient fill, invoke the ***sh*** operator from within the content stream of a type 1 (tiling) pattern.

Table 77 – Shading Operator

Operands	Operator	Description
name	**sh**	*(PDF 1.3)* Paint the shape and colour shading described by a shading dictionary, subject to the current clipping path. The current colour in the graphics state is neither used nor altered. The effect is different from that of painting a path using a shading pattern as the current colour. *name* is the name of a shading dictionary resource in the **Shading** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). All coordinates in the shading dictionary are interpreted relative to the current user space. (By contrast, when a shading dictionary is used in a type 2 pattern, the coordinates are expressed in pattern space.) All colours are interpreted in the colour space identified by the shading dictionary's **ColorSpace** entry (see Table 78). The **Background** entry, if present, is ignored. This operator should be applied only to bounded or geometrically defined shadings. If applied to an unbounded shading, it paints the shading's gradient fill across the entire clipping region, which may be time-consuming.

8.7.4.3 Shading Dictionaries

A shading dictionary specifies details of a particular gradient fill, including the type of shading to be used, the geometry of the area to be shaded, and the geometry of the gradient fill. Various shading types are available, depending on the value of the dictionary's **ShadingType** entry:

- Function-based shadings (type 1) define the colour of every point in the domain using a mathematical function (not necessarily smooth or continuous).
- Axial shadings (type 2) define a colour blend along a line between two points, optionally extended beyond the boundary points by continuing the boundary colours.
- Radial shadings (type 3) define a blend between two circles, optionally extended beyond the boundary circles by continuing the boundary colours. This type of shading is commonly used to represent three-dimensional spheres and cones.
- Free-form Gouraud-shaded triangle meshes (type 4) define a common construct used by many three-dimensional applications to represent complex coloured and shaded shapes. Vertices are specified in free-form geometry.
- Lattice-form Gouraud-shaded triangle meshes (type 5) are based on the same geometrical construct as type 4 but with vertices specified as a pseudorectangular lattice.
- Coons patch meshes (type 6) construct a shading from one or more colour patches, each bounded by four cubic Bézier curves.
- Tensor-product patch meshes (type 7) are similar to type 6 but with additional control points in each patch, affording greater control over colour mapping.

NOTE 1 Table 78 shows the entries that all shading dictionaries share in common; entries specific to particular shading types are described in the relevant sub-clause.

NOTE 2 The term *target coordinate space*, used in many of the following descriptions, refers to the coordinate space into which a shading is painted. For shadings used with a type 2 pattern dictionary, this is the pattern coordinate space, discussed in 8.7.2, "General Properties of Patterns". For shadings used directly with the ***sh*** operator, it is the current user space.

Table 78 – Entries Common to All Shading Dictionaries

Key	Type	Value
ShadingType	integer	*(Required)* The shading type: *1* Function-based shading *2* Axial shading *3* Radial shading *4* Free-form Gouraud-shaded triangle mesh *5* Lattice-form Gouraud-shaded triangle mesh *6* Coons patch mesh *7* Tensor-product patch mesh
ColorSpace	name or array	*(Required)* The colour space in which colour values shall be expressed. This may be any device, CIE-based, or special colour space except a **Pattern** space. See 8.7.4.4, "Colour Space: Special Considerations" for further information.
Background	array	*(Optional)* An array of colour components appropriate to the colour space, specifying a single background colour value. If present, this colour shall be used, before any painting operation involving the shading, to fill those portions of the area to be painted that lie outside the bounds of the shading object. NOTE In the opaque imaging model, the effect is as if the painting operation were performed twice: first with the background colour and then with the shading. NOTE The background colour is applied only when the shading is used as part of a shading pattern, not when it is painted directly with the ***sh*** operator.
BBox	rectangle	*(Optional)* An array of four numbers giving the left, bottom, right, and top coordinates, respectively, of the shading's bounding box. The coordinates shall be interpreted in the shading's target coordinate space. If present, this bounding box shall be applied as a temporary clipping boundary when the shading is painted, in addition to the current clipping path and any other clipping boundaries in effect at that time.
AntiAlias	boolean	*(Optional)* A flag indicating whether to filter the shading function to prevent *aliasing* artifacts. NOTE The shading operators sample shading functions at a rate determined by the resolution of the output device. Aliasing can occur if the function is not smooth—that is, if it has a high spatial frequency relative to the sampling rate. Anti-aliasing can be computationally expensive and is usually unnecessary, since most shading functions are smooth enough or are sampled at a high enough frequency to avoid aliasing effects. Anti-aliasing may not be implemented on some output devices, in which case this flag is ignored. Default value: **false**.

Shading types 4 to 7 shall be defined by a stream containing descriptive data characterizing the shading's gradient fill. In these cases, the shading dictionary is also a stream dictionary and may contain any of the standard entries common to all streams (see Table 5). In particular, shall include a **Length** entry.

In addition, some shading dictionaries also include a **Function** entry whose value shall be a function object (dictionary or stream) defining how colours vary across the area to be shaded. In such cases, the shading dictionary usually defines the geometry of the shading, and the function defines the colour transitions across that geometry. The function is required for some types of shading and optional for others. Functions are described in detail in 7.10, "Functions".

NOTE 3 Discontinuous colour transitions, or those with high spatial frequency, may exhibit aliasing effects when painted at low effective resolutions.

8.7.4.4 Colour Space: Special Considerations

8.7.4.4.1 General

Conceptually, a shading determines a colour value for each individual point within the area to be painted. In practice, however, the shading may actually be used to compute colour values only for some subset of the points in the target area, with the colours of the intervening points determined by interpolation between the ones computed. Conforming readers are free to use this strategy as long as the interpolated colour values approximate those defined by the shading to within the smoothness tolerance specified in the graphics state (see 10.6.3, "Smoothness Tolerance"). The **ColorSpace** entry common to all shading dictionaries not only defines the colour space in which the shading specifies its colour values but also determines the colour space in which colour interpolation is performed.

NOTE 1 Some types of shading (4 to 7) perform interpolation on a parametric value supplied as *input* to the shading's colour function, as described in the relevant sub-clause. This form of interpolation is conceptually distinct from the interpolation described here, which operates on the *output* colour values produced by the colour function and takes place within the shading's target colour space.

Gradient fills between colours defined by most shadings may be implemented using a variety of interpolation algorithms, and these algorithms may be sensitive to the characteristics of the colour space.

NOTE 2 Linear interpolation, for example, may have observably different results when applied in a **DeviceCMYK** colour space than in a **Lab** colour space, even if the starting and ending colours are perceptually identical. The difference arises because the two colour spaces are not linear relative to each other.

Shadings shall be rendered according to the following rules:

- If **ColorSpace** is a device colour space different from the native colour space of the output device, colour values in the shading shall be converted to the native colour space using the standard conversion formulas described in 10.3, "Conversions among Device Colour Spaces". To optimize performance, these conversions may take place at any time (before or after any interpolation on the colour values in the shading). Thus, shadings defined with device colour spaces may have colour gradient fills that are less accurate and somewhat device-dependent. (This does not apply to axial and radial shadings—shading types 2 and 3—because those shading types perform gradient fill calculations on a single variable and then convert to parametric colours.)

- If **ColorSpace** is a CIE-based colour space, all gradient fill calculations shall be performed in that space. Conversion to device colours shall occur only after all interpolation calculations have been performed. Thus, the colour gradients are device-independent for the colours generated at each point.

- If **ColorSpace** is a **Separation** or **DeviceN** colour space, a colour conversion (to the alternate colour space) occurs only if one or more of the specified colorants is not supported by the device. In that case, gradient fill calculations shall be performed in the designated **Separation** or **DeviceN** colour space before conversion to the alternate space. Thus, nonlinear tint transformation functions shall be accommodated for the best possible representation of the shading.

- If **ColorSpace** is an **Indexed** colour space, all colour values specified in the shading shall be immediately converted to the base colour space. Depending on whether the base colour space is a device or CIE-based space, gradient fill calculations shall be performed as stated above. Interpolation shall never occur in an **Indexed** colour space, which is quantized and therefore inappropriate for calculations that assume a continuous range of colours. For similar reasons, an **Indexed** colour space shall not be used in any shading whose colour values are generated by a function; this rule applies to any shading dictionary that contains a **Function** entry.

8.7.4.5 Shading Types

8.7.4.5.1 General

In addition to the entries listed in Table 78, all shading dictionaries have entries specific to the type of shading they represent, as indicated by the value of their **ShadingType** entry. The following sub-clauses describe the available shading types and the dictionary entries specific to each.

8.7.4.5.2 Type 1 (Function-Based) Shadings

In Type 1 (function-based) shadings, the colour at every point in the domain is defined by a specified mathematical function. The function need not be smooth or continuous. This type is the most general of the available shading types and is useful for shadings that cannot be adequately described with any of the other types. Table 79 shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see Table 78).

This type of shading shall not be used with an ***Indexed*** colour space.

Table 79 – Additional Entries Specific to a Type 1 Shading Dictionary

Key	Type	Value
Domain	array	*(Optional)* An array of four numbers [x_{min} x_{max} y_{min} y_{max}] specifying the rectangular domain of coordinates over which the colour function(s) are defined. Default value: [0.0 1.0 0.0 1.0].
Matrix	array	*(Optional)* An array of six numbers specifying a transformation matrix mapping the coordinate space specified by the **Domain** entry into the shading's target coordinate space. NOTE To map the domain rectangle [0.0 1.0 0.0 1.0] to a 1-inch square with lower-left corner at coordinates (100, 100) in default user space, the **Matrix** value would be [72 0 0 72 100 100]. Default value: the identity matrix [1 0 0 1 0 0].
Function	function	*(Required)* A 2-in, *n*-out function or an array of *n* 2-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). Each function's domain shall be a superset of that of the shading dictionary. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value.

The domain rectangle (**Domain**) establishes an internal coordinate space for the shading that is independent of the target coordinate space in which it shall be painted. The colour function(s) (**Function**) specify the colour of the shading at each point within this domain rectangle. The transformation matrix (**Matrix**) then maps the domain rectangle into a corresponding rectangle or parallelogram in the target coordinate space. Points within the shading's bounding box (**BBox**) that fall outside this transformed domain rectangle shall be painted with the shading's background colour (**Background**); if the shading dictionary has no **Background** entry, such points shall be left unpainted. If the function is undefined at any point within the declared domain rectangle, an error may occur, even if the corresponding transformed point falls outside the shading's bounding box.

8.7.4.5.3 Type 2 (Axial) Shadings

Type 2 (axial) shadings define a colour blend that varies along a linear axis between two endpoints and extends indefinitely perpendicular to that axis. The shading may optionally be extended beyond either or both endpoints by continuing the boundary colours indefinitely. Table 80 shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see Table 78).

This type of shading shall not be used with an ***Indexed*** colour space.

Table 80 – Additional Entries Specific to a Type 2 Shading Dictionary

Key	Type	Value
Coords	array	*(Required)* An array of four numbers [x_0 y_0 x_1 y_1] specifying the starting and ending coordinates of the axis, expressed in the shading's target coordinate space.
Domain	array	*(Optional)* An array of two numbers [t_0 t_1] specifying the limiting values of a parametric variable *t.* The variable is considered to vary linearly between these two values as the colour gradient varies between the starting and ending points of the axis. The variable *t* becomes the input argument to the colour function(s). Default value: [0.0 1.0].
Function	function	*(Required)* A 1-in, *n*-out function or an array of *n* 1-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). The function(s) shall be called with values of the parametric variable *t* in the domain defined by the **Domain** entry. Each function's domain shall be a superset of that of the shading dictionary. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value.
Extend	array	*(Optional)* An array of two boolean values specifying whether to extend the shading beyond the starting and ending points of the axis, respectively. Default value: [**false false**].

The colour blend shall be accomplished by linearly mapping each point (*x*, *y*) along the axis between the endpoints (x_0, y_0) and (x_1, y_1) to a corresponding point in the domain specified by the shading dictionary's **Domain** entry. The points (0, 0) and (1, 0) in the domain correspond respectively to (x_0, y_0) and (x_1, y_1) on the axis. Since all points along a line in domain space perpendicular to the line from (0, 0) to (1, 0) have the same colour, only the new value of *x* needs to be computed:

$$x' = \frac{(x_1 - x_0) \times (x - x_0) + (y_1 - y_0) \times (y - y_0)}{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

The value of the parametric variable *t* is then determined from *x*¢ as follows:

- For 0 £ *x*¢ £ 1, *t* = t_0 + (t_1 - t_0) ¥ *x*¢.
- For *x*¢ < 0, if the first element of the **Extend** array is **true**, then *t* = t_0; otherwise, *t* is undefined and the point shall be left unpainted.
- For *x*¢ > 1, if the second element of the **Extend** array is **true**, then *t* = t_1; otherwise, *t* is undefined and the point shall be left unpainted.

The resulting value of *t* shall be passed as input to the function(s) defined by the shading dictionary's **Function** entry, yielding the component values of the colour with which to paint the point (*x*, *y*).

NOTE Figure L.10 in Annex L shows three examples of the use of an axial shading to fill a rectangle and display text. The area to be filled extends beyond the shading's bounding box. The shading is the same in all three cases, except for the values of the **Background** and **Extend** entries in the shading dictionary. In the first example, the shading is not extended at either end and no background colour is specified; therefore, the shading is clipped to its bounding box at both ends. The second example still has no background colour specified, but the shading is extended at both ends; the result is to fill the remaining portions of the filled area with the colours defined at the ends of the shading. In the third example, the shading is extended at both ends and a background colour is specified; therefore, the background colour is used for the portions of the filled area beyond the ends of the shading.

8.7.4.5.4 Type 3 (Radial) Shadings

Type 3 (radial) shadings define a colour blend that varies between two circles. Shadings of this type are commonly used to depict three-dimensional spheres and cones. Shading dictionaries for this type of shading contain the entries shown in Table 81, as well as those common to all shading dictionaries (see Table 78).

This type of shading shall not be used with an ***Indexed*** colour space.

Table 81 – Additional Entries Specific to a Type 3 Shading Dictionary

Key	Type	Value
Coords	array	*(Required)* An array of six numbers [x_0 y_0 r_0 x_1 y_1 r_1] specifying the centres and radii of the starting and ending circles, expressed in the shading's target coordinate space. The radii r_0 and r_1 shall both be greater than or equal to 0. If one radius is 0, the corresponding circle shall be treated as a point; if both are 0, nothing shall be painted.
Domain	array	*(Optional)* An array of two numbers [t_0 t_1] specifying the limiting values of a parametric variable *t*. The variable is considered to vary linearly between these two values as the colour gradient varies between the starting and ending circles. The variable *t* becomes the input argument to the colour function(s). Default value: [0.0 1.0].
Function	function	*(Required)* A 1-in, *n*-out function or an array of *n* 1-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). The function(s) shall be called with values of the parametric variable *t* in the domain defined by the shading dictionary's **Domain** entry. Each function's domain shall be a superset of that of the shading dictionary. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value.
Extend	array	*(Optional)* An array of two boolean values specifying whether to extend the shading beyond the starting and ending circles, respectively. Default value: [**false** **false**].

The colour blend is based on a family of *blend circles* interpolated between the starting and ending circles that shall be defined by the shading dictionary's **Coords** entry. The blend circles shall be defined in terms of a subsidiary parametric variable

$$s = \frac{t - t_0}{t_1 - t_0}$$

which varies linearly between 0.0 and 1.0 as *t* varies across the domain from t_0 to t_1, as specified by the dictionary's **Domain** entry. The centre and radius of each blend circle shall be given by the following parametric equations:

$$\begin{aligned} x_c(s) &= x_0 + s \times (x_1 - x_0) \\ y_c(s) &= y_0 + s \times (y_1 - y_0) \\ r(s) &= r_0 + s \times (r_1 - r_0) \end{aligned}$$

Each value of *s* between 0.0 and 1.0 determines a corresponding value of *t*, which is passed as the input argument to the function(s) defined by the shading dictionary's **Function** entry. This yields the component values of the colour with which to fill the corresponding blend circle. For values of *s* not lying between 0.0 and

1.0, the boolean elements of the shading dictionary's **Extend** array determine whether and how the shading is extended. If the first of the two elements is **true**, the shading shall be extended beyond the defined starting circle to values of *s* less than 0.0; if the second element is **true**, the shading shall be extended beyond the defined ending circle to *s* values greater than 1.0.

NOTE 1 Either of the starting and ending circles may be larger than the other. If the shading is extended at the smaller end, the family of blend circles continues as far as that value of *s* for which the radius of the blend circle *r*(*s*) = 0. If the shading is extended at the larger end, the blend circles continue as far as that *s* value for which *r*(*s*) is large enough to encompass the shading's entire bounding box (**BBox**). Extending the shading can thus cause painting to extend beyond the areas defined by the two circles themselves. The two examples in the rightmost column of Figure L.11 in Annex L depict the results of extending the shading at the smaller and larger ends, respectively.

Conceptually, all of the blend circles shall be painted in order of increasing values of *s*, from smallest to largest. Blend circles extending beyond the starting circle shall be painted in the same colour defined by the shading dictionary's **Function** entry for the starting circle ($t = t_0$, $s = 0.0$). Blend circles extending beyond the ending circle shall be painted in the colour defined for the ending circle ($t = t_1$, $s = 1.0$). The painting is opaque, with the colour of each circle completely overlaying those preceding it. Therefore, if a point lies within more than one blend circle, its final colour shall be that of the last of the enclosing circles to be painted, corresponding to the greatest value of *s*.

NOTE 2 If one of the starting and ending circles entirely contains the other, the shading depicts a sphere, as in Figure L.12 and Figure L.13 in Annex L. In Figure L.12 in Annex L, the inner circle has zero radius; it is the starting circle in the figure on the left and the ending circle in the figure on the right. Neither shading is extended at either the smaller or larger end. In Figure L.13 in Annex L, the inner circle in both figures has a nonzero radius and the shading is extended at the larger end. In each plate, a background colour is specified for the figure on the right but not for the figure on the left.

NOTE 3 If neither circle contains the other, the shading depicts a cone. If the starting circle is larger, the cone appears to point out of the page. If the ending circle is larger, the cone appears to point into the page (see Figure L.11 in Annex L).

EXAMPLE 1 This example paints the leaf-covered branch shown in Figure L.14 in Annex L. Each leaf is filled with the same radial shading (object number 5). The colour function (object 10) is a stitching function (described in 7.10.4, "Type 3 (Stitching) Functions") whose two subfunctions (objects 11 and 12) are both exponential interpolation functions (see 7.10.3, "Type 2 (Exponential Interpolation) Functions").

```
5  0  obj                                              % Shading dictionary
   <<  /ShadingType  3
       /ColorSpace  /DeviceCMYK
       /Coords  [0.0  0.0  0.096  0.0  0.0  1.000]        % Concentric circles
       /Function  10 0 R
       /Extend  [true  true]
   >>
endobj

10  0  obj                                             %Colour function
   <<  /FunctionType  3
       /Domain  [0.0  1.0]
       /Functions  [11 0 R  12 0 R]
       /Bounds  [0.708]
       /Encode  [1.0  0.0  0.0  1.0]
   >>
endobj

11  0  obj                                             % First subfunction
   <<  /FunctionType  2
       /Domain  [0.0  1.0]
       /C0  [0.929  0.357  1.000  0.298]
       /C1  [0.631  0.278  1.000  0.027]
       /N  1.048
   >>
endobj
```

```
12 0 obj                                                    % Second subfunction
    << /FunctionType 2
        /Domain [0.0 1.0]
        /C0 [0.929 0.357 1.000 0.298]
        /C1 [0.941 0.400 1.000 0.102]
        /N 1.374
    >>
endobj
```

EXAMPLE 2 This example shows how each leaf shown in Figure L.14 in Annex L is drawn as a path and then filled with the shading (where the name Sh1 is associated with object 5 by the **Shading** subdictionary of the current resource dictionary; see 7.8.3, "Resource Dictionaries").

```
316.789 140.311 m                                           % Move to start of leaf
303.222 146.388 282.966 136.518 279.122 121.983 c           % Curved segment
277.322 120.182 l                                           % Straight line
285.125 122.688 291.441 121.716 298.156 119.386 c           % Curved segment
336.448 119.386 l                                           % Straight line
331.072 128.643 323.346 137.376 316.789 140.311 c           % Curved segment
W n                                                         % Set clipping path
q                                                           % Save graphics state
27.7843 0.0000 0.0000 -27.7843 310.2461 121.1521 cm            % Set matrix
/Sh1 sh                                                        % Paint shading
Q                                                           % Restore graphics state
```

8.7.4.5.5 Type 4 Shadings (Free-Form Gouraud-Shaded Triangle Meshes)

Type 4 shadings (free-form Gouraud-shaded triangle meshes) are commonly used to represent complex coloured and shaded three-dimensional shapes. The area to be shaded is defined by a path composed entirely of triangles. The colour at each vertex of the triangles is specified, and a technique known as *Gouraud interpolation* is used to colour the interiors. The interpolation functions defining the shading may be linear or nonlinear. Table 82 shows the entries specific to this type of shading dictionary, in addition to those common to all shading dictionaries (see Table 78) and stream dictionaries (see Table 5).

Table 82 – Additional Entries Specific to a Type 4 Shading Dictionary

Key	Type	Value
BitsPerCoordinate	integer	*(Required)* The number of bits used to represent each vertex coordinate. The value shall be 1, 2, 4, 8, 12, 16, 24, or 32.
BitsPerComponent	integer	*(Required)* The number of bits used to represent each colour component. The value shall be 1, 2, 4, 8, 12, or 16.
BitsPerFlag	integer	*(Required)* The number of bits used to represent the edge flag for each vertex (see below). The value of **BitsPerFlag** shall be 2, 4, or 8, but only the least significant 2 bits in each flag value shall be used. The value for the edge flag shall be 0, 1, or 2.
Decode	array	*(Required)* An array of numbers specifying how to map vertex coordinates and colour components into the appropriate ranges of values. The decoding method is similar to that used in image dictionaries (see 8.9.5.2, "Decode Arrays"). The ranges shall be specified as follows: $[x_{min}\ x_{max}\ y_{min}\ y_{max}\ c_{1,min}\ c_{1,max}\ \dots\ c_{n,min}\ c_{n,max}]$ Only one pair of *c* values shall be specified if a **Function** entry is present.

Table 82 – Additional Entries Specific to a Type 4 Shading Dictionary (continued)

Key	Type	Value
Function	function	*(Optional)* A 1-in, *n*-out function or an array of *n* 1-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). If this entry is present, the colour data for each vertex shall be specified by a single parametric variable rather than by *n* separate colour components. The designated function(s) shall be called with each interpolated value of the parametric variable to determine the actual colour at each point. Each input value shall be forced into the range interval specified for the corresponding colour component in the shading dictionary's **Decode** array. Each function's domain shall be a superset of that interval. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value. This entry shall not be used with an **Indexed** colour space.

Unlike shading types 1 to 3, types 4 to 7 shall be represented as streams. Each stream contains a sequence of vertex coordinates and colour data that defines the triangle mesh. In a type 4 shading, each vertex is specified by the following values, in the order shown:

$f\ x\ y\ c_1 \dots c_n$

where

f is the vertex's edge flag (discussed below)

x and *y* are its horizontal and vertical coordinates

$c_1 \dots c_n$ are its colour components

All vertex coordinates shall be expressed in the shading's target coordinate space. If the shading dictionary includes a **Function** entry, only a single parametric value, *t*, shall be specified for each vertex in place of the colour components $c_1 \dots c_n$.

The *edge flag* associated with each vertex determines the way it connects to the other vertices of the triangle mesh. A vertex v_a with an edge flag value $f_a = 0$ begins a new triangle, unconnected to any other. At least two more vertices (v_b and v_c) shall be provided, but their edge flags shall be ignored. These three vertices define a triangle (v_a, v_b, v_c), as shown in Figure 24.

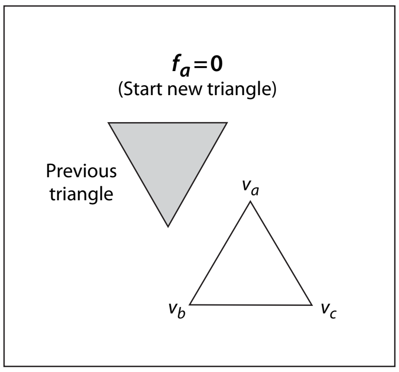


Figure 24 – Starting a New Triangle in a Free-form Gouraud-shaded Triangle Mesh

Subsequent triangles shall be defined by a single new vertex combined with two vertices of the preceding triangle. Given triangle (v_a, v_b, v_c), where vertex v_a precedes vertex v_b in the data stream and v_b precedes v_c, a new vertex v_d can form a new triangle on side v_{bc} or side v_{ac}, as shown in Figure 25. (Side v_{ab} is assumed to be shared with a preceding triangle and therefore is not available for continuing the mesh.) 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). An edge flag of $f_d = 0$ starts a new triangle, as described above.

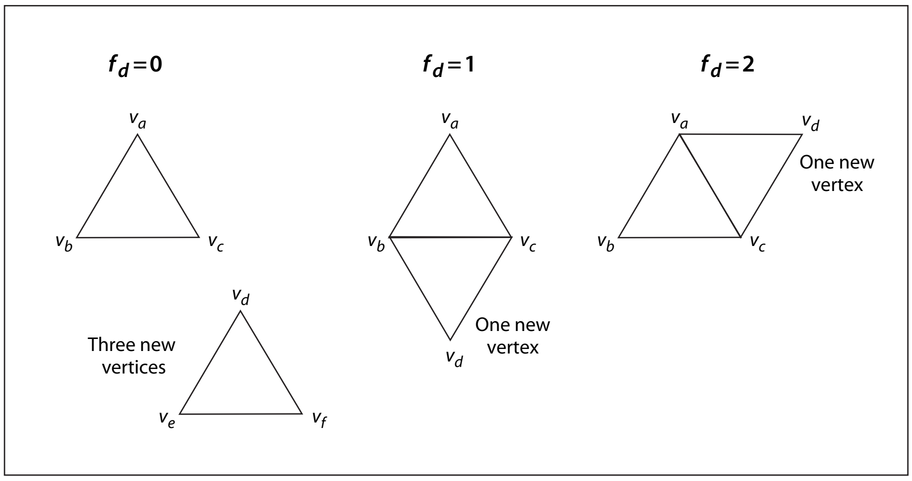


Figure 25 – Connecting Triangles in a Free-form Gouraud-shaded Triangle Mesh

Complex shapes can be created by using the edge flags to control the edge on which subsequent triangles are formed.

EXAMPLE Figure 26 shows two simple examples. Mesh 1 begins with triangle 1 and uses the following edge flags to draw each succeeding triangle:

1 ($f_a = f_b = f_c = 0$)	7 ($f_i = 2$)
2 ($f_d = 1$)	8 ($f_j = 2$)
3 ($f_e = 1$)	9 ($f_k = 2$)
4 ($f_f = 1$)	10 ($f_l = 1$)
5 ($f_g = 1$)	11 ($f_m = 1$)
6 ($f_h = 1$)	

Mesh 2 again begins with triangle 1 and uses the following edge flags:

1 ($f_a = f_b = f_c = 0$)	4 ($f_f = 2$)
2 ($f_d = 1$)	5 ($f_g = 2$)
3 ($f_e = 2$)	6 ($f_h = 2$)

The stream shall provide vertex data for a whole number of triangles with appropriate edge flags; otherwise, an error occurs.

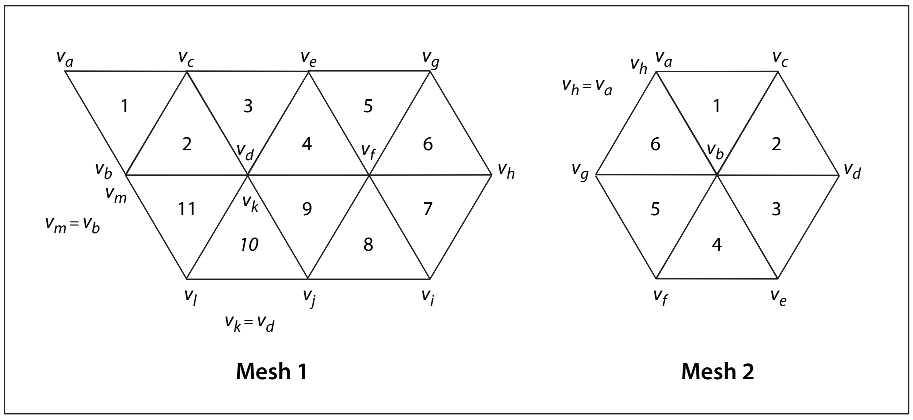


Figure 26 – Varying the Value of the Edge Flag to Create Different Shapes

The data for each vertex consists of the following items, reading in sequence from higher-order to lower-order bit positions:

- An edge flag, expressed in **BitsPerFlag** bits
- A pair of horizontal and vertical coordinates, expressed in **BitsPerCoordinate** bits each
- A set of *n* colour components (where *n* is the number of components in the shading's colour space), expressed in **BitsPerComponent** bits each, in the order expected by the **sc** operator

Each set of vertex data shall occupy a whole number of bytes. If the total number of bits required is not divisible by 8, the last data byte for each vertex is padded at the end with extra bits, which shall be ignored. The coordinates and colour values shall be decoded according to the **Decode** array in the same way as in an image dictionary (see 8.9.5.2, "Decode Arrays").

If the shading dictionary contains a **Function** entry, the colour data for each vertex shall be specified by a single parametric value *t* rather than by *n* separate colour components. All linear interpolation within the triangle mesh shall be done using the *t* values. After interpolation, the results shall be passed to the function(s) specified in the **Function** entry to determine the colour at each point.

8.7.4.5.6 Type 5 Shadings (Lattice-Form Gouraud-Shaded Triangle Meshes)

Type 5 shadings (lattice-form Gouraud-shaded triangle meshes) are similar to type 4, but instead of using free-form geometry, their vertices are arranged in a *pseudorectangular lattice*, which is topologically equivalent to a rectangular grid. The vertices are organized into rows, which need not be geometrically linear (see Figure 27).

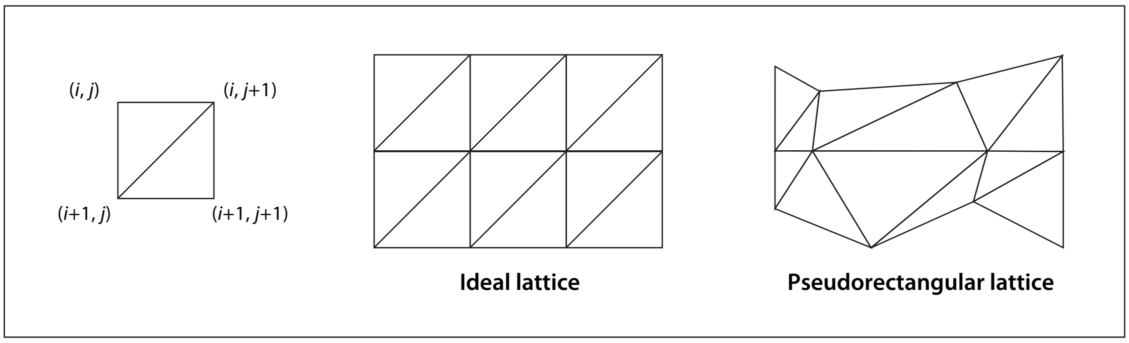


Figure 27 – Lattice-form Triangle Meshes

Table 83 shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see Table 78) and stream dictionaries (see Table 5).

The data stream for a type 5 shading has the same format as for type 4, except that type 5 does not use edge flags to define the geometry of the triangle mesh. The data for each vertex thus consists of the following values, in the order shown:

$x \ y \ c_1 \ldots c_n$

where

x and y shall be the vertex's horizontal and vertical coordinates

$c_1 \ldots c_n$ shall be its colour components

Table 83 – Additional Entries Specific to a Type 5 Shading Dictionary

Key	Type	Value
BitsPerCoordinate	integer	*(Required)* The number of bits used to represent each vertex coordinate. The value shall be 1, 2, 4, 8, 12, 16, 24, or 32.
BitsPerComponent	integer	*(Required)* The number of bits used to represent each colour component. The value shall be 1, 2, 4, 8, 12, or 16.
VerticesPerRow	integer	*(Required)* The number of vertices in each row of the lattice; the value shall be greater than or equal to 2. The number of rows need not be specified.
Decode	array	*(Required)* An array of numbers specifying how to map vertex coordinates and colour components into the appropriate ranges of values. The decoding method is similar to that used in image dictionaries (see 8.9.5.2, "Decode Arrays"). The ranges shall be specified as follows: $[x_{\min} \ x_{\max} \ y_{\min} \ y_{\max} \ c_{1,\min} \ c_{1,\max} \ \ldots \ c_{n,\min} \ c_{n,\max}]$ Only one pair of *c* values shall be specified if a **Function** entry is present.
Function	function	*(Optional)* A 1-in, *n*-out function or an array of *n* 1-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). If this entry is present, the colour data for each vertex shall be specified by a single parametric variable rather than by *n* separate colour components. The designated function(s) shall be called with each interpolated value of the parametric variable to determine the actual colour at each point. Each input value shall be forced into the range interval specified for the corresponding colour component in the shading dictionary's **Decode** array. Each function's domain shall be a superset of that interval. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value. This entry shall not be used with an **Indexed** colour space.

All vertex coordinates are expressed in the shading's target coordinate space. If the shading dictionary includes a **Function** entry, only a single parametric value, *t*, shall be present for each vertex in place of the colour components $c_1 \ldots c_n$.

The **VerticesPerRow** entry in the shading dictionary gives the number of vertices in each row of the lattice. All of the vertices in a row shall be specified sequentially, followed by those for the next row. Given *m* rows of *k* vertices each, the triangles of the mesh shall be constructed using the following triplets of vertices, as shown in Figure 27:

$$(V_{i,j}, V_{i,j+1}, V_{i+1,j}) \qquad \text{for } 0 \le i \le m-2,\ 0 \le j \le k-2$$
$$(V_{i,j+1}, V_{i+1,j}, V_{i+1,j+1})$$

See 8.7.4.5.5, "Type 4 Shadings (Free-Form Gouraud-Shaded Triangle Meshes)" for further details on the format of the vertex data.

8.7.4.5.7 Type 6 Shadings (Coons Patch Meshes)

Type 6 shadings (Coons patch meshes) are constructed from one or more *colour patches*, each bounded by four cubic Bézier curves. Degenerate Bézier curves are allowed and are useful for certain graphical effects. At least one complete patch shall be specified.

A Coons patch generally has two independent aspects:

- Colours are specified for each corner of the unit square, and bilinear interpolation is used to fill in colours over the entire unit square (see the upper figure in Figure L.15 in Annex L).
- Coordinates are mapped from the unit square into a four-sided patch whose sides are not necessarily linear (see the lower figure in Figure L.15 in Annex L). 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 28.

The sides of the patch are given by four cubic Bézier curves, C_1, C_2, D_1, and D_2, defined over a pair of parametric variables, u and v, that vary horizontally and vertically across the unit square. The four corners of the Coons patch satisfy the following equations:

$$C_1(0) = D_1(0)$$

$$C_1(1) = D_2(0)$$

$$C_2(0) = D_1(1)$$

$$C_2(1) = D_2(1)$$

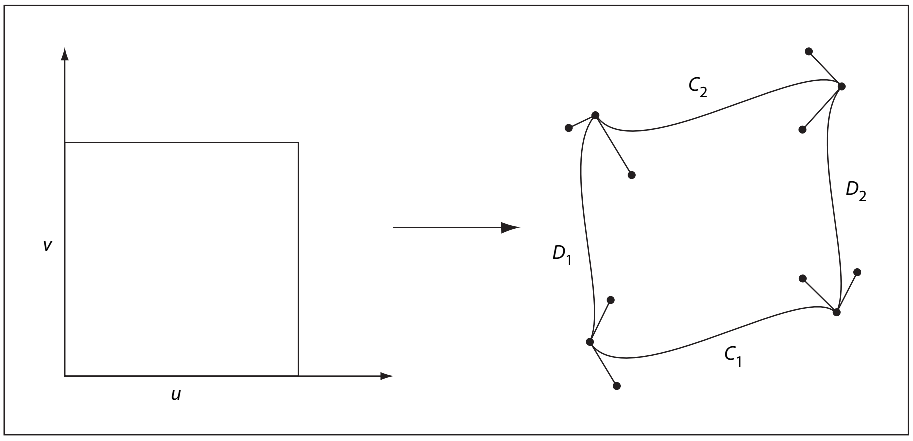


Figure 28 – Coordinate Mapping from a Unit Square to a Four-sided Coons Patch

Two surfaces can be described that are linear interpolations between the boundary curves. Along the u axis, the surface S_C is defined by

$$S_C(u, v) = (1 - v) \times C_1(u) + v \times C_2(u)$$

Along the *v* axis, the surface S_D is given by

$$S_D(u, v) = (1 - u) \times D_1(v) + u \times D_2(v)$$

A third surface is the bilinear interpolation of the four corners:

$$\begin{aligned} S_B(u, v) = (1 - v) &\times [(1 - u) \times C_1(0) + u \times C_1(1)] \\ + v\ &\times [(1 - u) \times C_2(0) + u \times C_2(1)] \end{aligned}$$

The coordinate mapping for the shading is given by the surface *S*, defined as

$$S = S_C + S_D - S_B$$

This defines the geometry of each patch. A patch mesh is constructed from a sequence of one or more such coloured patches.

Patches can sometimes appear to fold over on themselves—for example, if a boundary curve intersects itself. As the value of parameter *u* or *v* increases in parameter space, the location of the corresponding pixels in device space may change direction so that new pixels are mapped onto previous pixels already mapped. If more than one point (*u*, *v*) in parameter space is mapped to the same point in device space, the point selected shall be the one with the largest value of *v*. If multiple points have the same *v*, the one with the largest value of *u* shall be selected. If one patch overlaps another, the patch that appears later in the data stream shall paint over the earlier one.

NOTE The patch is a control surface rather than a painting geometry. The outline of a projected square (that is, the painted area) might not be the same as the patch boundary if, for example, the patch folds over on itself, as shown in Figure 29.

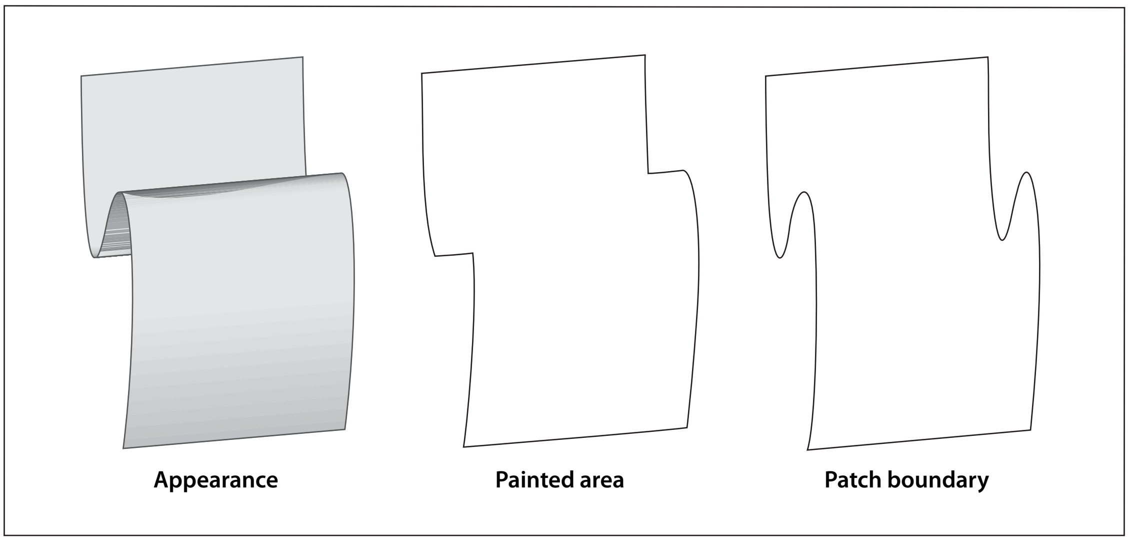


Figure 29 – Painted Area and Boundary of a Coons Patch

Table 84 shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see Table 78) and stream dictionaries (see Table 5).

Table 84 – Additional Entries Specific to a Type 6 Shading Dictionary

Key	Type	Value
BitsPerCoordinate	integer	*(Required)* The number of bits used to represent each geometric coordinate. The value shall be 1, 2, 4, 8, 12, 16, 24, or 32.
BitsPerComponent	integer	*(Required)* The number of bits used to represent each colour component. The value shall be 1, 2, 4, 8, 12, or 16.
BitsPerFlag	integer	*(Required)* The number of bits used to represent the edge flag for each patch (see below). The value shall be 2, 4, or 8, but only the least significant 2 bits in each flag value shall be used. Valid values for the edge flag shall be 0, 1, 2, and 3.
Decode	array	*(Required)* An array of numbers specifying how to map coordinates and colour components into the appropriate ranges of values. The decoding method is similar to that used in image dictionaries (see 8.9.5.2, "Decode Arrays"). The ranges shall be specified as follows: $[x_{\min}\ x_{\max}\ y_{\min}\ y_{\max}\ c_{1,\min}\ c_{1,\max}\ \dots\ c_{n,\min}\ c_{n,\max}]$ Only one pair of *c* values shall be specified if a **Function** entry is present.
Function	function	*(Optional)* A 1-in, *n*-out function or an array of *n* 1-in, 1-out functions (where *n* is the number of colour components in the shading dictionary's colour space). If this entry is present, the colour data for each vertex shall be specified by a single parametric variable rather than by *n* separate colour components. The designated function(s) shall be called with each interpolated value of the parametric variable to determine the actual colour at each point. Each input value shall be forced into the range interval specified for the corresponding colour component in the shading dictionary's **Decode** array. Each function's domain shall be a superset of that interval. If the value returned by the function for a given colour component is out of range, it shall be adjusted to the nearest valid value. This entry shall not be used with an **Indexed** colour space.

The data stream provides a sequence of Bézier control points and colour values that define the shape and colours of each patch. All of a patch's control points are given first, followed by the colour values for its corners. This differs from a triangle mesh (shading types 4 and 5), in which the coordinates and colour of each vertex are given together. All control point coordinates are expressed in the shading's target coordinate space. See 8.7.4.5.5, "Type 4 Shadings (Free-Form Gouraud-Shaded Triangle Meshes)" for further details on the format of the data.

As in free-form triangle meshes (type 4), each patch has an *edge flag* that indicates which edge, if any, it shares with the previous patch. An edge flag of 0 begins a new patch, unconnected to any other. This shall be followed by 12 pairs of coordinates, $x_1\ y_1\ \ x_2\ y_2\ \dots\ x_{12}\ y_{12}$, which specify the Bézier control points that define the four boundary curves. Figure 30 shows how these control points correspond to the cubic Bézier curves C_1, C_2, D_1, and D_2 identified in Figure 28. Colour values shall be given for the four corners of the patch, in the same order as the control points corresponding to the corners. Thus, c_1 is the colour at coordinates (x_1, y_1), c_2 at (x_4, y_4), c_3 at (x_7, y_7), and c_4 at (x_{10}, y_{10}), as shown in the figure.

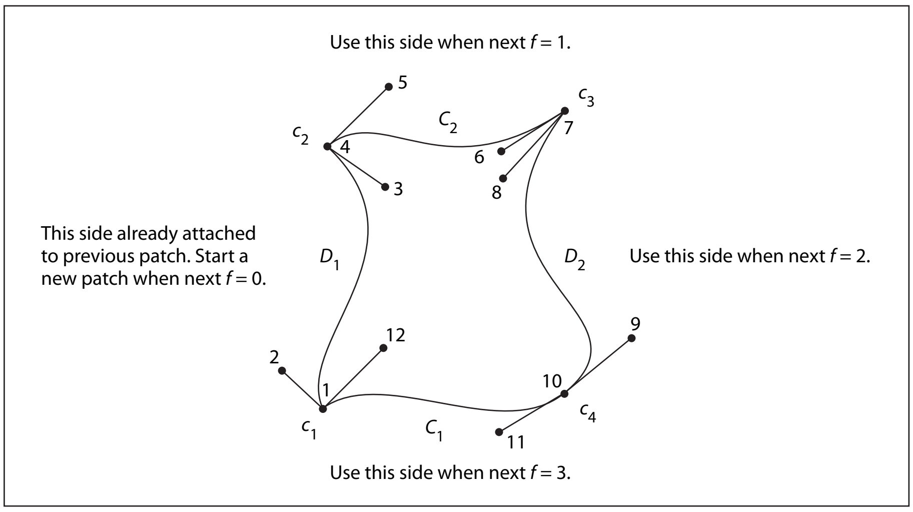

Use this side when next f = 1.
This side already attached to previous patch. Start a new patch when next f = 0.
Use this side when next f = 2.
Use this side when next f = 3.
c_1
c_2
c_3
c_4
C_1
C_2
D_1
D_2
1
2
3
4
5
6
7
8
9
10
11
12

Figure 30 – Colour Values and Edge Flags in Coons Patch Meshes

Figure 30 also shows how nonzero values of the edge flag (f = 1, 2, or 3) connect a new patch to one of the edges of the previous patch. In this case, some of the previous patch's control points serve implicitly as control points for the new patch as well (see Figure 31), and therefore shall not be explicitly repeated in the data stream. Table 85 summarizes the required data values for various values of the edge flag.

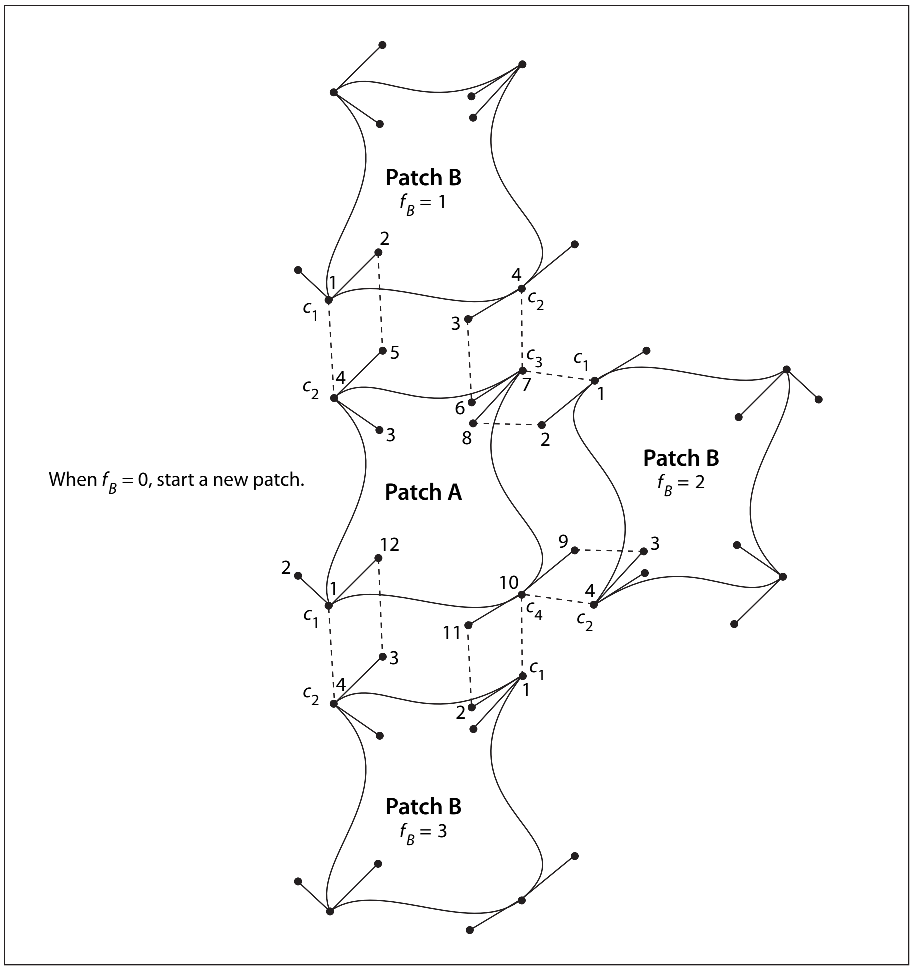


Figure 31 – Edge Connections in a Coons Patch Mesh

If the shading dictionary contains a **Function** entry, the colour data for each corner of a patch shall be specified by a single parametric value *t* rather than by *n* separate colour components $c_1 \ldots c_n$. All linear interpolation within the mesh shall be done using the *t* values. After interpolation, the results shall be passed to the function(s) specified in the **Function** entry to determine the colour at each point.

Table 85 – Data Values in a Coons Patch Mesh

Edge Flag	Next Set of Data Values
$f = 0$	$x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4\ x_5\ y_5\ x_6\ y_6$ $x_7\ y_7\ x_8\ y_8\ x_9\ y_9\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}$ $c_1\ c_2\ c_3\ c_4$ 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 $c_1 = c_2$ previous $(x_2, y_2) = (x_5, y_5)$ previous $c_2 = c_3$ previous $(x_3, y_3) = (x_6, y_6)$ previous $(x_4, y_4) = (x_7, y_7)$ 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 $c_1 = c_3$ previous $(x_2, y_2) = (x_8, y_8)$ previous $c_2 = c_4$ previous $(x_3, y_3) = (x_9, y_9)$ previous $(x_4, y_4) = (x_{10}, y_{10})$ 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 $c_1 = c_4$ previous $(x_2, y_2) = (x_{11}, y_{11})$ previous $c_2 = c_1$ previous $(x_3, y_3) = (x_{12}, y_{12})$ previous $(x_4, y_4) = (x_1, y_1)$ previous

8.7.4.5.8 Type 7 Shadings (Tensor-Product Patch Meshes)

Type 7 shadings (tensor-product patch meshes) are identical to type 6, except that they are based on a bicubic tensor-product patch defined by 16 control points instead of the 12 control points that define a Coons patch. The shading dictionaries representing the two patch types differ only in the value of the **ShadingType** entry and in the number of control points specified for each patch in the data stream.

NOTE Although the Coons patch is more concise and easier to use, the tensor-product patch affords greater control over colour mapping.

Like the Coons patch mapping, the tensor-product patch mapping is controlled by the location and shape of four cubic Bézier curves marking the boundaries of the patch. However, the tensor-product patch has four additional, “internal” control points to adjust the mapping. The 16 control points can be arranged in a 4-by-4 array indexed by row and column, as follows (see Figure 32):

$$\begin{array}{cccc} p_{03} & p_{13} & p_{23} & p_{33} \\ p_{02} & p_{12} & p_{22} & p_{32} \\ p_{01} & p_{11} & p_{21} & p_{31} \\ p_{00} & p_{10} & p_{20} & p_{30} \end{array}$$

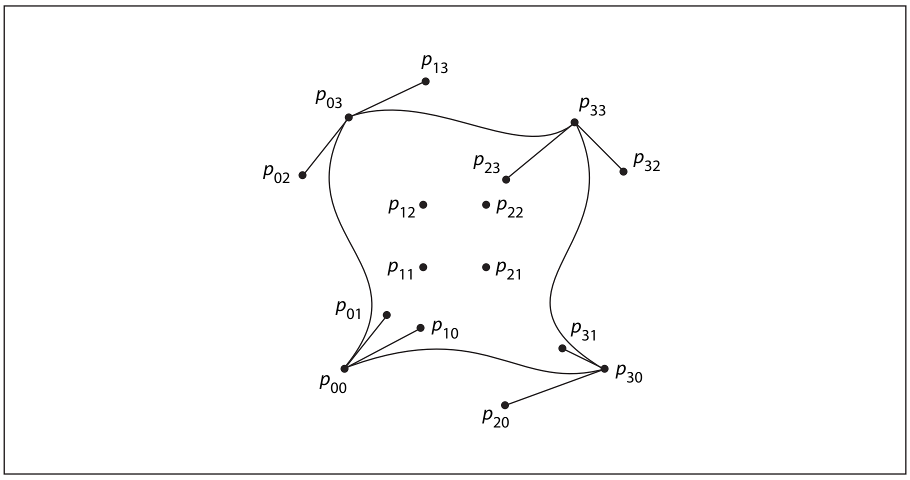


Figure 32 – Control Points in a Tensor-product Patch

As in a Coons patch mesh, the geometry of the tensor-product patch is described by a surface defined over a pair of parametric variables, *u* and *v*, which vary horizontally and vertically across the unit square. The surface is defined by the equation

$$S(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{ij} \times B_i(u) \times B_j(v)$$

where p_{ij} is the control point in column *i* and row *j* of the tensor, and B_i and B_j are the *Bernstein polynomials*

$$B_0(t) = (1 - t)^3$$

$$B_1(t) = 3t \times (1 - t)^2$$

$$B_2(t) = 3t^2 \times (1 - t)$$

$$B_3(t) = t^3$$

Since each point p_{ij} is actually a pair of coordinates (x_{ij}, y_{ij}), the surface can also be expressed as

$$x(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} x_{ij} \times B_i(u) \times B_j(v)$$

$$y(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} y_{ij} \times B_i(u) \times B_j(v)$$

The geometry of the tensor-product patch can be visualized in terms of a cubic Bézier curve moving from the bottom boundary of the patch to the top. At the bottom and top, the control points of this curve coincide with those of the patch's bottom ($p_{00} \dots p_{30}$) and top ($p_{03} \dots p_{33}$) boundary curves, respectively. As the curve moves from the bottom edge of the patch to the top, each of its four control points follows a trajectory that is in turn a cubic Bézier curve defined by the four control points in the corresponding column of the array. That is, the starting point of the moving curve follows the trajectory defined by control points $p_{00} \dots p_{03}$, the trajectory of the ending point is defined by points $p_{30} \dots p_{33}$, and those of the two intermediate control points by $p_{10} \dots p_{13}$ and $p_{20} \dots p_{23}$. Equivalently, the patch can be considered to be traced by a cubic Bézier curve moving from the left edge to the right, with its control points following the trajectories defined by the rows of the coordinate array instead of the columns.

The Coons patch (type 6) is actually a special case of the tensor-product patch (type 7) in which the four internal control points (p_{11}, p_{12}, p_{21}, p_{22}) are implicitly defined by the boundary curves. The values of the internal control points are given by these equations:

$$p_{11} = 1/9 \times [-4 \times p_{00} + 6 \times (p_{01} + p_{10}) - 2 \times (p_{03} + p_{30}) + 3 \times (p_{31} + p_{13}) - 1 \times p_{33}]$$

$$p_{12} = 1/9 \times [-4 \times p_{03} + 6 \times (p_{02} + p_{13}) - 2 \times (p_{00} + p_{33}) + 3 \times (p_{32} + p_{10}) - 1 \times p_{30}]$$

$$p_{21} = 1/9 \times [-4 \times p_{30} + 6 \times (p_{31} + p_{20}) - 2 \times (p_{33} + p_{00}) + 3 \times (p_{01} + p_{23}) - 1 \times p_{03}]$$

$$p_{22} = 1/9 \times [-4 \times p_{33} + 6 \times (p_{32} + p_{23}) - 2 \times (p_{30} + p_{03}) + 3 \times (p_{02} + p_{20}) - 1 \times p_{00}]$$

In the more general tensor-product patch, the values of these four points are unrestricted.

The coordinates of the control points in a tensor-product patch shall be specified in the shading's data stream in the following order:

4	5	6	7
3	14	15	8
2	13	16	9
1	12	11	10

All control point coordinates shall be expressed in the shading's target coordinate space. These shall be followed by the colour values for the four corners of the patch, in the same order as the corners themselves. If the patch's edge flag *f* is 0, all 16 control points and four corner colours shall be explicitly specified in the data stream. If *f* is 1, 2, or 3, the control points and colours for the patch's shared edge are implicitly understood to be the same as those along the specified edge of the previous patch and shall not be repeated in the data stream. Table 86 summarizes the data values for various values of the edge flag *f*, expressed in terms of the row and column indices used in Figure 32. See 8.7.4.5.5, "Type 4 Shadings (Free-Form Gouraud-Shaded Triangle Meshes)" for further details on the format of the data.

Table 86 – Data values in a tensor-product patch mesh

Edge Flag	Next Set of Data Values
$f = 0$	$x_{00}\ y_{00}\ x_{01}\ y_{01}\ x_{02}\ y_{02}\ x_{03}\ y_{03}\ x_{13}\ y_{13}\ x_{23}\ y_{23}\ x_{33}\ y_{33}\ x_{32}\ y_{32}$ $x_{31}\ y_{31}\ x_{30}\ y_{30}\ x_{20}\ y_{20}\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}\ x_{22}\ y_{22}\ x_{21}\ y_{21}$ $c_{00}\ c_{03}\ c_{33}\ c_{30}$ New patch; no implicit values
$f = 1$	$x_{13}\ y_{13}\ x_{23}\ y_{23}\ x_{33}\ y_{33}\ x_{32}\ y_{32}\ x_{31}\ y_{31}\ x_{30}\ y_{30}$ $x_{20}\ y_{20}\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}\ x_{22}\ y_{22}\ x_{21}\ y_{21}$ $c_{33}\ c_{30}$ Implicit values: $(x_{00}, y_{00}) = (x_{03}, y_{03})$ previous $c_{00} = c_{03}$ previous $(x_{01}, y_{01}) = (x_{13}, y_{13})$ previous $c_{03} = c_{33}$ previous $(x_{02}, y_{02}) = (x_{23}, y_{23})$ previous $(x_{03}, y_{03}) = (x_{33}, y_{33})$ previous
$f = 2$	$x_{13}\ y_{13}\ x_{23}\ y_{23}\ x_{33}\ y_{33}\ x_{32}\ y_{32}\ x_{31}\ y_{31}\ x_{30}\ y_{30}$ $x_{20}\ y_{20}\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}\ x_{22}\ y_{22}\ x_{21}\ y_{21}$ $c_{33}\ c_{30}$ Implicit values: $(x_{00}, y_{00}) = (x_{33}, y_{33})$ previous $c_{00} = c_{33}$ previous $(x_{01}, y_{01}) = (x_{32}, y_{32})$ previous $c_{03} = c_{30}$ previous $(x_{02}, y_{02}) = (x_{31}, y_{31})$ previous $(x_{03}, y_{03}) = (x_{30}, y_{30})$ previous
$f = 3$	$x_{13}\ y_{13}\ x_{23}\ y_{23}\ x_{33}\ y_{33}\ x_{32}\ y_{32}\ x_{31}\ y_{31}\ x_{30}\ y_{30}$ $x_{20}\ y_{20}\ x_{10}\ y_{10}\ x_{11}\ y_{11}\ x_{12}\ y_{12}\ x_{22}\ y_{22}\ x_{21}\ y_{21}$ $c_{33}\ c_{30}$ Implicit values: $(x_{00}, y_{00}) = (x_{30}, y_{30})$ previous $c_{00} = c_{30}$ previous $(x_{01}, y_{01}) = (x_{20}, y_{20})$ previous $c_{03} = c_{00}$ previous $(x_{02}, y_{02}) = (x_{10}, y_{10})$ previous $(x_{03}, y_{03}) = (x_{00}, y_{00})$ previous

8.8 External Objects

8.8.1 General

An *external object* (commonly called an *XObject*) is a graphics object whose contents are defined by a self-contained stream, separate from the content stream in which it is used. There are three types of external objects:

- An *image XObject* (8.9.5, "Image Dictionaries") represents a sampled visual image such as a photograph.
- A *form XObject* (8.10, "Form XObjects") is a self-contained description of an arbitrary sequence of graphics objects.
- A *PostScript XObject* (8.8.2, "PostScript XObjects") contains a fragment of code expressed in the PostScript page description language. PostScript XObjects should not be used.

Two further categories of external objects, *group XObjects* and *reference XObjects (both PDF 1.4)*, are actually specialized types of form XObjects with additional properties. See 8.10.3, "Group XObjects" and 8.10.4, "Reference XObjects" for additional information.

Any XObject can be painted as part of another content stream by means of the **Do** operator (see Table 87). This operator applies to any type of XObject—image, form, or PostScript. The syntax is the same in all cases, although details of the operator's behaviour differ depending on the type.

Table 87 – XObject Operator

Operands	Operator	Description
name	**Do**	Paint the specified XObject. The operand *name* shall appear as a key in the **XObject** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). The associated value shall be a stream whose **Type** entry, if present, is **XObject**. The effect of **Do** depends on the value of the XObject's **Subtype** entry, which may be **Image** (see 8.9.5, "Image Dictionaries"), **Form** (see 8.10, "Form XObjects"), or **PS** (see 8.8.2, "PostScript XObjects").

8.8.2 PostScript XObjects

Beginning with PDF 1.1, a content stream may include PostScript language fragments. These fragments may be used only when printing to a PostScript output device; they shall have no effect either when viewing the document on-screen or when printing it to a non-PostScript device. In addition, conforming readers may not be able to interpret the PostScript fragments. Hence, this capability should be used with extreme caution and only if there is no other way to achieve the same result. Inappropriate use of PostScript XObjects can cause PDF files to print incorrectly.

A *PostScript XObject* is an XObject stream whose **Subtype** entry has the value **PS**. A PostScript XObject dictionary may contain the entries shown in Table 88 in addition to the usual entries common to all streams (see Table 5).

Table 88 – Additional Entries Specific to a PostScript XObject Dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **XObject** for a PostScript XObject.
Subtype	name	*(Required)* The type of XObject that this dictionary describes; shall be **PS** for a PostScript XObject. Alternatively, the value of this entry may be **Form**, with an additional **Subtype2** entry whose value shall be **PS**.
Level1	stream	*(Optional)* A stream whose contents shall be used in place of the PostScript XObject's stream when the target PostScript interpreter is known to support only LanguageLevel 1.

If a PDF content stream is translated by a conforming reader into the PostScript language, any **Do** operation that references a PostScript XObject may be replaced by the contents of the XObject stream itself. The stream shall be copied without interpretation. The PostScript fragment may use Type 1 and TrueType fonts listed in the **Font** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"), accessing them by their **BaseFont** names using the PostScript **findfont** operator. The fragment shall not use other types of fonts listed in the **Font** subdictionary. It should not reference the PostScript definitions corresponding to PDF procedure sets (see 14.2, "Procedure Sets"), which are subject to change.

8.9 Images

8.9.1 General

PDF’s painting operators include general facilities for dealing with sampled images. A *sampled image* (or just *image* for short) is a rectangular array of *sample values*, each representing a colour. The image may approximate the appearance of some natural scene obtained through an input scanner or a video camera, or it may be generated synthetically.

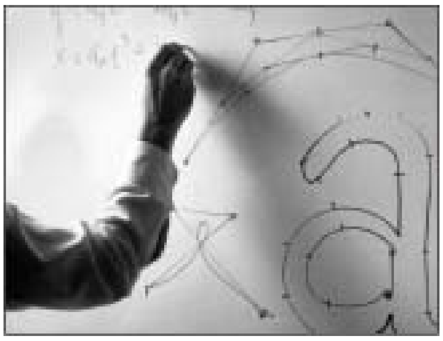


Figure 33 – Typical Sampled Image

NOTE 1 An image is defined by a sequence of samples obtained by scanning the image array in row or column order. Each sample in the array consists of as many colour components as are needed for the colour space in which they are specified—for example, one component for **DeviceGray**, three for **DeviceRGB**, four for **DeviceCMYK**, or whatever number is required by a particular **DeviceN** space. Each component is a 1-, 2-, 4-, 8-, or *(PDF 1.5)* 16-bit integer, permitting the representation of 2, 4, 16, 256, or *(PDF 1.5)* 65536 distinct values for each component. Other component sizes can be accommodated when a **JPXDecode** filter is used; see 7.4.9, "JPXDecode Filter".

NOTE 2 PDF provides two means for specifying images:

An *image XObject* (described in 8.9.5, "Image Dictionaries") is a stream object whose dictionary specifies attributes of the image and whose data contains the image samples. Like all external objects, it is painted on the page by invoking the **Do** operator in a content stream (see 8.8, "External Objects"). Image XObjects have other uses as well, such as for alternate images (see 8.9.5.4, "Alternate Images"), image masks (8.9.6, "Masked Images"), and thumbnail images (12.3.4, "Thumbnail Images").

An *inline image* is a small image that is completely defined—both attributes and data—directly inline within a content stream. The kinds of images that can be represented in this way are limited; see 8.9.7, "Inline Images" for details.

8.9.2 Image Parameters

The properties of an image—resolution, orientation, scanning order, and so forth—are entirely independent of the characteristics of the raster output device on which the image is to be rendered. A conforming reader usually renders images by a sampling technique that attempts to approximate the colour values of the source as accurately as possible. The actual accuracy achieved depends on the resolution and other properties of the output device.

To paint an image, four interrelated items shall be specified:

- The format of the image: number of columns (width), number of rows (height), number of colour components per sample, and number of bits per colour component
- The sample data constituting the image’s visual content

- The correspondence between coordinates in user space and those in the image's own internal coordinate space, defining the region of user space that will receive the image
- The mapping from colour component values in the image data to component values in the image's colour space

All of these items shall be specified explicitly or implicitly by an image XObject or an inline image.

NOTE For convenience, the following sub-clauses refer consistently to the object defining an image as an *image dictionary*. Although this term properly refers only to the dictionary portion of the stream object representing an image XObject, it should be understood to apply equally to the stream's data portion or to the parameters and data of an inline image.

8.9.3 Sample Representation

The source format for an image shall be described by four parameters:

- The width of the image in samples
- The height of the image in samples
- The number of colour components per sample
- The number of bits per colour component

The image dictionary shall specify the width, height, and number of bits per component explicitly. The number of colour components shall be inferred from the colour space specified in the dictionary.

NOTE For images using the *JPXDecode* filter (see 7.4.9, "JPXDecode Filter"), the number of bits per component is determined from the image data and not specified in the image dictionary. The colour space may or may not be specified in the dictionary.

Sample data shall be represented as a stream of bytes, interpreted as 8-bit unsigned integers in the range 0 to 255. The bytes constitute a continuous bit stream, with the high-order bit of each byte first. This bit stream, in turn, is divided into units of *n* bits each, where *n* is the number of bits per component. Each unit encodes a colour component value, given with high-order bit first; units of 16 bits shall be given with the most significant byte first. Byte boundaries shall be ignored, except that each row of sample data shall begin on a byte boundary. If the number of data bits per row is not a multiple of 8, the end of the row is padded with extra bits to fill out the last byte. A conforming reader shall ignore these padding bits.

Each *n*-bit unit within the bit stream shall be interpreted as an unsigned integer in the range 0 to 2^n- 1, with the high-order bit first. The image dictionary's **Decode** entry maps this integer to a colour component value, equivalent to what could be used with colour operators such as **sc** or **g**. Colour components shall be interleaved sample by sample; for example, in a three-component *RGB* image, the red, green, and blue components for one sample are followed by the red, green, and blue components for the next.

If the image dictionary's **ImageMask** entry is **false** or absent, the colour samples in an image shall be interpreted according to the colour space specified in the image dictionary (see 8.6, "Colour Spaces"), without reference to the colour parameters in the graphics state. However, if the image dictionary's **ImageMask** entry is **true**, the sample data shall be interpreted as a *stencil mask* for applying the graphics state's nonstroking colour parameters (see 8.9.6.2, "Stencil Masking").

8.9.4 Image Coordinate System

Each image has its own internal coordinate system, or *image space*. The image occupies a rectangle in image space *w* units wide and *h* units high, where *w* and *h* are the width and height of the image in samples. Each sample occupies one square unit. The coordinate origin (0, 0) is at the upper-left corner of the image, with coordinates ranging from 0 to *w* horizontally and 0 to *h* vertically.

The image's sample data is ordered by row, with the horizontal coordinate varying most rapidly. This is shown in Figure 34, where the numbers inside the squares indicate the order of the samples, counting from 0. The upper-left corner of the first sample is at coordinates (0, 0), the second at (1, 0), and so on through the last sample of the first row, whose upper-left corner is at (*w* - 1, 0) and whose upper-right corner is at (*w*, 0). The next samples after that are at coordinates (0, 1), (1, 1), and so on to the final sample of the image, whose upper-left corner is at (*w* - 1, *h* - 1) and whose lower-right corner is at (*w*, *h*).

NOTE The image coordinate system and scanning order imposed by PDF do not preclude using different conventions in the actual image. Coordinate transformations can be used to map from other conventions to the PDF convention.

The correspondence between image space and user space is constant: the unit square of user space, bounded by user coordinates (0, 0) and (1, 1), corresponds to the boundary of the image in image space (see Figure 35). Following the normal convention for user space, the coordinate (0, 0) is at the *lower-left* corner of this square, corresponding to coordinates (0, *h*) in image space. The implicit transformation from image space to user space, if specified explicitly, would be described by the matrix [1⁄*w* 0 0 -1⁄*h* 0 1].

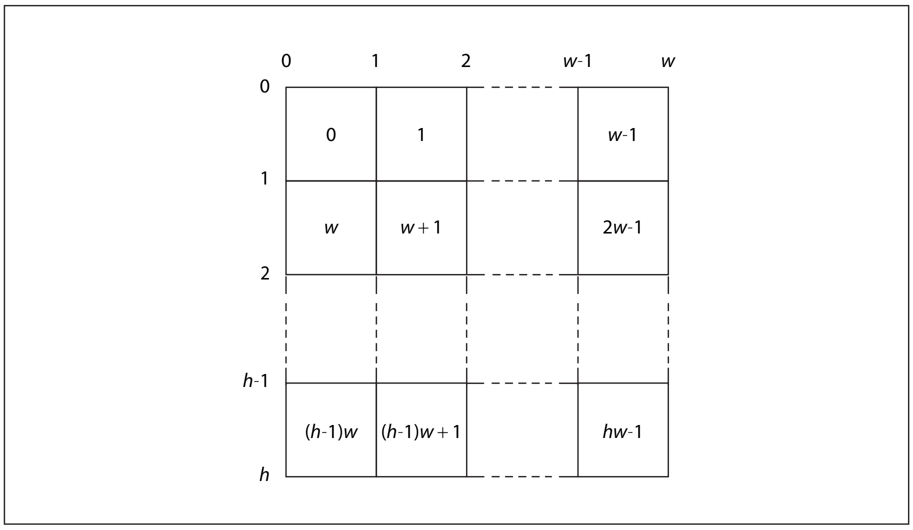


Figure 34 – Source Image Coordinate System

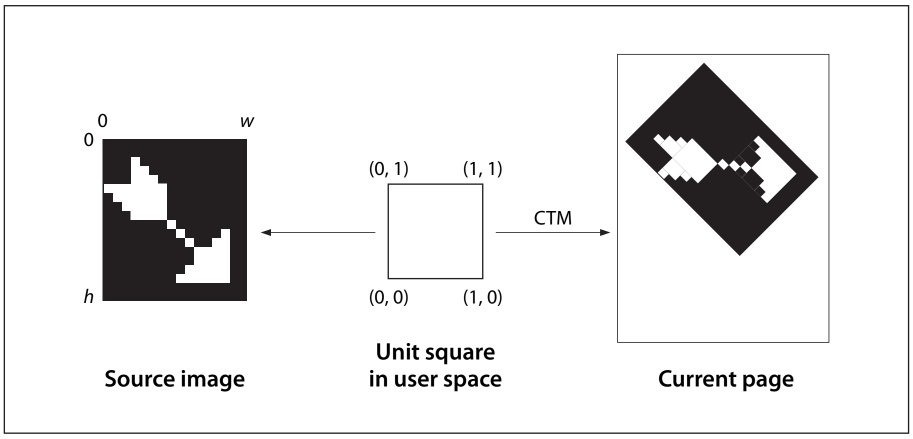


Figure 35 – Mapping the Source Image

An image can be placed on the output page in any position, orientation, and size by using the **cm** operator to modify the current transformation matrix (CTM) so as to map the unit square of user space to the rectangle or parallelogram in which the image shall be painted. Typically, this is done within a pair of **q** and **Q** operators to isolate the effect of the transformation, which can include translation, rotation, reflection, and skew (see 8.3, "Coordinate Systems").

EXAMPLE If the **XObject** subdictionary of the current resource dictionary defines the name Image1 to denote an image XObject, the code shown in this example paints the image in a rectangle whose lower-left corner is at coordinates (100, 200), that is rotated 45 degrees counter clockwise, and that is 150 units wide and 80 units high.

```
q                                                  % Save graphics state
   1  0  0  1  100  200  cm                        % Translate
   0.7071  0.7071  -0.7071  0.7071  0  0  cm       % Rotate
   150  0  0  80  0  0  cm                         % Scale
   /Image1  Do                                     % Paint image
Q                                                  % Restore graphics state
```

As discussed in 8.3.4, "Transformation Matrices", these three transformations could be combined into one. Of course, if the aspect ratio (width to height) of the original image in this example is different from 150:80, the result will be distorted.

8.9.5 Image Dictionaries

8.9.5.1 General

An image dictionary—that is, the dictionary portion of a stream representing an image XObject—may contain the entries listed in Table 89 in addition to the usual entries common to all streams (see Table 5). There are many relationships among these entries, and the current colour space may limit the choices for some of them. Attempting to use an image dictionary whose entries are inconsistent with each other or with the current colour space shall cause an error.

The entries described here are appropriate for a *base image*—one that is invoked directly with the ***Do*** operator. Some of the entries should not be used for images used in other ways, such as for alternate images (see 8.9.5.4, "Alternate Images"), image masks (see 8.9.6, "Masked Images"), or thumbnail images (see 12.3.4, "Thumbnail Images"). Except as noted, such irrelevant entries are simply ignored by a conforming reader

Table 89 – Additional Entries Specific to an Image Dictionary

Key	**Type**	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **XObject** for an image XObject.
Subtype	name	*(Required)* The type of XObject that this dictionary describes; shall be **Image** for an image XObject.
Width	integer	*(Required)* The width of the image, in samples.
Height	integer	*(Required)* The height of the image, in samples.
ColorSpace	name or array	*(Required for images, except those that use the **JPXDecode** filter; not allowed forbidden for image masks)* The colour space in which image samples shall be specified; it can be any type of colour space except **Pattern**. If the image uses the **JPXDecode** filter, this entry may be present: • If **ColorSpace** is present, any colour space specifications in the JPEG2000 data shall be ignored. • If **ColorSpace** is absent, the colour space specifications in the JPEG2000 data shall be used. The **Decode** array shall also be ignored unless **ImageMask** is **true**.

Table 89 – Additional Entries Specific to an Image Dictionary (continued)

Key	**Type**	Value
BitsPerComponent	integer	*(Required except for image masks and images that use the **JPXDecode** filter)* The number of bits used to represent each colour component. Only a single value shall be specified; the number of bits shall be the same for all colour components. The value shall be 1, 2, 4, 8, or (in PDF 1.5) 16. If **ImageMask** is **true**, this entry is optional, but if specified, its value shall be 1. If the image stream uses a filter, the value of **BitsPerComponent** shall be consistent with the size of the data samples that the filter delivers. In particular, a **CCITTFaxDecode** or **JBIG2Decode** filter shall always deliver 1-bit samples, a **RunLengthDecode** or **DCTDecode** filter shall always deliver 8-bit samples, and an **LZWDecode** or **FlateDecode** filter shall deliver samples of a specified size if a predictor function is used. If the image stream uses the **JPXDecode** filter, this entry is optional and shall be ignored if present. The bit depth is determined by the conforming reader in the process of decoding the JPEG2000 image.
Intent	name	*(Optional; PDF 1.1)* The name of a colour rendering intent to be used in rendering the image (see 8.6.5.8, "Rendering Intents"). Default value: the current rendering intent in the graphics state.
ImageMask	boolean	*(Optional)* A flag indicating whether the image shall be treated as an image mask (see 8.9.6, "Masked Images"). If this flag is **true**, the value of **BitsPerComponent** shall be 1 and **Mask** and **ColorSpace** shall not be specified; unmasked areas shall be painted using the current nonstroking colour. Default value: **false**.
Mask	stream or array	*(Optional except for image masks; not allowed for image masks; PDF 1.3)* An image XObject defining an image mask to be applied to this image (see 8.9.6.3, "Explicit Masking"), or an array specifying a range of colours to be applied to it as a colour key mask (see 8.9.6.4, "Colour Key Masking"). If **ImageMask** is **true**, this entry shall not be present.
Decode	array	*(Optional)* An array of numbers describing how to map image samples into the range of values appropriate for the image's colour space (see 8.9.5.2, "Decode Arrays"). If **ImageMask** is **true**, the array shall be either [0 1] or [1 0]; otherwise, its length shall be twice the number of colour components required by **ColorSpace**. If the image uses the **JPXDecode** filter and **ImageMask** is **false**, **Decode** shall be ignored by a conforming reader. Default value: see 8.9.5.2, "Decode Arrays".
Interpolate	boolean	*(Optional)* A flag indicating whether image interpolation shall be performed by a conforming reader (see 8.9.5.3, "Image Interpolation"). Default value: **false**.
Alternates	array	*(Optional; PDF 1.3)* An array of alternate image dictionaries for this image (see 8.9.5.4, "Alternate Images"). The order of elements within the array shall have no significance. This entry shall not be present in an image XObject that is itself an alternate image.

Table 89 – Additional Entries Specific to an Image Dictionary (continued)

Key	**Type**	Value
SMask	stream	*(Optional; PDF 1.4)* A subsidiary image XObject defining a *soft-mask image* (see 11.6.5.3, "Soft-Mask Images") that shall be used as a source of mask shape or mask opacity values in the transparent imaging model. The alpha source parameter in the graphics state determines whether the mask values shall be interpreted as shape or opacity. If present, this entry shall override the current soft mask in the graphics state, as well as the image's **Mask** entry, if any. However, the other transparency-related graphics state parameters—blend mode and alpha constant—shall remain in effect. If **SMask** is absent, the image shall have no associated soft mask (although the current soft mask in the graphics state may still apply).
SMaskInData	integer	*(Optional for images that use the* ***JPXDecode*** *filter, meaningless otherwise; PDF 1.5)* A code specifying how soft-mask information (see 11.6.5.3, "Soft-Mask Images") encoded with image samples shall be used: 0 If present, encoded soft-mask image information shall be ignored. 1 The image's data stream includes encoded soft-mask values. A conforming reader may create a soft-mask image from the information to be used as a source of mask shape or mask opacity in the transparency imaging model. 2 The image's data stream includes colour channels that have been preblended with a background; the image data also includes an opacity channel. A conforming reader may create a soft-mask image with a **Matte** entry from the opacity channel information to be used as a source of mask shape or mask opacity in the transparency model. If this entry has a nonzero value, **SMask** shall not be specified. See also 7.4.9, "JPXDecode Filter". Default value: 0.
Name	name	*(Required in PDF 1.0; optional otherwise)* The name by which this image XObject is referenced in the **XObject** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). This entry is obsolescent and shall no longer be used.
StructParent	integer	*(Required if the image is a structural content item; PDF 1.3)* The integer key of the image's entry in the structural parent tree (see 14.7.4.4, "Finding Structure Elements from Content Items").
ID	byte string	*(Optional; PDF 1.3; indirect reference preferred)* The digital identifier of the image's parent Web Capture content set (see 14.10.6, "Object Attributes Related to Web Capture").
OPI	dictionary	*(Optional; PDF 1.2)* An OPI version dictionary for the image; see 14.11.7, "Open Prepress Interface (OPI)". If **ImageMask** is **true**, this entry shall be ignored.
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* containing metadata for the image (see 14.3.2, "Metadata Streams").
OC	dictionary	*(Optional; PDF 1.5)* An optional content group or optional content membership dictionary (see 8.11, "Optional Content"), specifying the optional content properties for this image XObject. Before the image is processed by a conforming reader, its visibility shall be determined based on this entry. If it is determined to be invisible, the entire image shall be skipped, as if there were no **Do** operator to invoke it.

EXAMPLE This example defines an image 256 samples wide by 256 high, with 8 bits per sample in the DeviceGray colour space. It paints the image on a page with its lower-left corner positioned at coordinates (45, 140) in current user space and scaled to a width and height of 132 user space units.

```
20  0  obj                                  % Page object
    <<  /Type  /Page
        /Parent  1 0 R
        /Resources  21 0 R
        /MediaBox  [0  0  612  792]
        /Contents  23 0 R
    >>
endobj

21  0  obj                                  % Resource dictionary for page
    <<  /ProcSet  [/PDF  /ImageB]
        /XObject  <<  /Im1  22 0 R  >>
    >>
endobj

22  0  obj                                  % Image XObject
    <<  /Type  /XObject
        /Subtype  /Image
        /Width  256
        /Height  256
        /ColorSpace  /DeviceGray
        /BitsPerComponent  8
        /Length  83183
        /Filter  /ASCII85Decode
    >>

stream
9LhZI9h\GY9i+bb;,p:e;G9SP92/)X9MJ>^:f14d;,U(X8P;cO;G9e];c$=k9Mn\]
…Image data representing 65,536 samples…
8P;cO;G9e];c$=k9Mn\]~>
endstream
endobj

23  0  obj                                      % Contents of page
    <<  /Length  56  >>
stream
    q                                           % Save graphics state
        132  0  0  132  45  140  cm             % Translate to (45,140) and scale by 132
        /Im1  Do                                % Paint image
    Q                                           % Restore graphics state
endstream
endobj
```

8.9.5.2 Decode Arrays

An image's data stream is initially decomposed into integers in the domain 0 to 2^n -1, where *n* is the value of the image dictionary's **BitsPerComponent** entry. The image's **Decode** array specifies a linear mapping of each integer component value to a number that would be appropriate as a component value in the image's colour space.

Each pair of numbers in a **Decode** array specifies the lower and upper values to which the domain of sample values in the image is mapped. A **Decode** array shall contain one pair of numbers for each component in the colour space specified by the image's **ColorSpace** entry. The mapping for each colour component, by a conforming reader shall be a linear transformation; that is, it shall use the following formula for linear interpolation:

$$\begin{aligned} y &= \text{Interpolate}(x, x_{min}, x_{max}, y_{min}, y_{max}) \\ &= y_{min} + \left((x - x_{min}) \times \frac{y_{max} - y_{min}}{x_{max} - x_{min}} \right) \end{aligned}$$

This formula is used to convert a value x between x_{min} and x_{max} to a corresponding value y between y_{min} and y_{max}, projecting along the line defined by the points (x_{min}, y_{min}) and (x_{max}, y_{max}).

NOTE 1 While this formula applies to values outside the domain x_{min} to x_{max} and does not require that $x_{min} < x_{max}$, note that interpolation used for colour conversion, such as the **Decode** array, does require that $x_{min} < x_{max}$ and clips x values to this domain so that $y = y_{min}$ for all x £ x_{min}, and $y = y_{max}$ for all x Š x_{max}.

For a **Decode** array of the form [D_{min} D_{max}], this can be written as

$$\begin{aligned} y &= \text{Interpolate}\,(x, 0, 2^n - 1, D_{min}, D_{max}) \\ &= D_{min} + \left(x \times \frac{D_{max} - D_{min}}{2^n - 1} \right) \end{aligned}$$

where

n shall be the value of **BitsPerComponent**

x shall be the input value, in the domain 0 to 2^n - 1

D_{min} and D_{max} shall be the values specified in the **Decode** array

y is the output value, which shall be interpreted in the image's colour space

Samples with a value of 0 shall be mapped to D_{min}, those with a value of 2^n - 1 shall be mapped to D_{max}, and those with intermediate values shall be mapped linearly between D_{min} and D_{max}. Table 90 lists the default **Decode** arrays which shall be used with the various colour spaces by a conforming reader.

NOTE 2 For most colour spaces, the **Decode** arrays listed in the table map into the full range of allowed component values. For an **Indexed** colour space, the default **Decode** array ensures that component values that index a colour table are passed through unchanged.

Table 90 – Default Decode Arrays

Colour Space	**Decode Array**
DeviceGray	[0.0 1.0]
DeviceRGB	[0.0 1.0 0.0 1.0 0.0 1.0]
DeviceCMYK	[0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0]
CalGray	[0.0 1.0]
CalRGB	[0.0 1.0 0.0 1.0 0.0 1.0]
Lab	[0 100 a_{min} a_{max} b_{min} b_{max}] where a_{min}, a_{max}, b_{min}, and b_{max} correspond to the values in the **Range** array of the image's colour space

Table 90 – Default Decode Arrays (continued)

Colour Space	Decode Array
ICCBased	Same as the value of **Range** in the ICC profile of the image's colour space
Indexed	[0 *N*], where $N = 2^n - 1$
Pattern	(Not permitted with images)
Separation	[0.0 1.0]
DeviceN	[0.0 1.0 0.0 1.0 … 0.0 1.0] (one pair of elements for each colour component)

NOTE 3 It is possible to specify a mapping that *inverts* sample colour intensities by specifying a D_{min} value greater than D_{max}. For example, if the image's colour space is **DeviceGray** and the **Decode** array is [1.0 0.0], an input value of 0 is mapped to 1.0 (white); an input value of $2^n - 1$ is mapped to 0.0 (black).

The D_{min} and D_{max} parameters for a colour component need not fall within the range of values allowed for that component.

NOTE 4 For instance, if an application uses 6-bit numbers as its native image sample format, it can represent those samples in PDF in 8-bit form, setting the two unused high-order bits of each sample to 0. The image dictionary should then specify a **Decode** array of [0.00000 4.04762], which maps input values from 0 to 63 into the range 0.0 to 1.0 (4.04762 being approximately equal to 255 ³ 63).

If an output value falls outside the range allowed for a component, it shall be automatically adjusted to the nearest allowed value.

8.9.5.3 Image Interpolation

When the resolution of a source image is significantly lower than that of the output device, each source sample covers many device pixels. As a result, images can appear jaggy or blocky. These visual artifacts can be reduced by applying an *image interpolation* algorithm during rendering. Instead of painting all pixels covered by a source sample with the same colour, image interpolation attempts to produce a smooth transition between adjacent sample values.

Image interpolation is enabled by setting the **Interpolate** entry in the image dictionary to **true**. It shall be disabled by default because it may increase the time required to render the image.

NOTE A conforming Reader may choose to not implement this feature of PDF, or may use any specific implementation of interpolation that it wishes.

8.9.5.4 Alternate Images

Alternate images (PDF 1.3) provide a straightforward and backward-compatible way to include multiple versions of an image in a PDF file for different purposes. These variant representations of the image may differ, for example, in resolution or in colour space. The primary goal is to reduce the need to maintain separate versions of a PDF document for low-resolution on-screen viewing and high-resolution printing.

A *base image* (that is, the image XObject referred to in a resource dictionary) may contain an **Alternates** entry. The value of this entry shall be an array of *alternate image dictionaries* specifying variant representations of the base image. Each alternate image dictionary shall contain an image XObject for one variant and shall specify its properties. Table 91 shows the contents of an alternate image dictionary.

Table 91 – Entries in an Alternate Image Dictionary

Key	Type	Value
Image	stream	*(Required)* The image XObject for the alternate image.
DefaultForPrinting	boolean	*(Optional)* A flag indicating whether this alternate image shall be the default version to be used for printing. At most one alternate for a given base image shall be so designated. If no alternate has this entry set to **true**, the base image shall be used for printing by a conforming reader.
OC	dictionary	*(Optional; PDF 1.5)* An optional content group (see 8.11.2, "Optional Content Groups") or optional content membership dictionary (see 8.11.2.2, "Optional Content Membership Dictionaries") that facilitates the selection of which alternate image to use.

EXAMPLE The following shows an image with a single alternate. The base image is a grayscale image, and the alternate is a high-resolution RGB image stored on a Web server.

```
10  0  obj                          % Image XObject
   <<  /Type  /XObject
       /Subtype  /Image
       /Width  100
       /Height  200
       /ColorSpace  /DeviceGray
       /BitsPerComponent  8
       /Alternates  15 0 R
       /Length  2167
       /Filter  /DCTDecode
   >>

stream
…Image data…
endstream
endobj

15  0  obj                          % Alternate images array
   [  <<  /Image  16 0 R
          /DefaultForPrinting  true
   >>
   ]
endobj

16  0  obj                          % Alternate image
   <<  /Type  /XObject
       /Subtype  /Image
       /Width  1000
       /Height  2000
       /ColorSpace  /DeviceRGB
       /BitsPerComponent  8
       /Length  0                   % This is an external stream

       /F  <<  /FS  /URL
               /F  (http://www.myserver.mycorp.com/images/exttest.jpg)
           >>
       /FFilter  /DCTDecode
   >>

stream
endstream
endobj
```

In PDF 1.5, optional content (see 8.11, "Optional Content") may be used to facilitate selection between alternate images. If an image XObject contains both an **Alternates** entry and an **OC** entry, the choice of which image to use shall be determined as follows:

a) If the image's **OC** entry specifies that the base image is visible, that image shall be displayed.

b) Otherwise, the list of alternates specified by the **Alternates** entry is examined, and the first alternate containing an **OC** entry specifying that its content should be visible shall be shown. (Alternate images that have no **OC** entry shall not be shown.)

8.9.6 Masked Images

8.9.6.1 General

Ordinarily, in the opaque imaging model, images mark all areas they occupy on the page as if with opaque paint. All portions of the image, whether black, white, gray, or colour, completely obscure any marks that may previously have existed in the same place on the page. In the graphic arts industry and page layout applications, however, it is common to crop or mask out the background of an image and then place the masked image on a different background so that the existing background shows through the masked areas. A number of PDF features are available for achieving such masking effects:

- The **ImageMask** entry in the image dictionary, specifies that the image data shall be used as a *stencil mask* for painting in the current colour.

- The **Mask** entry in the image dictionary *(PDF 1.3)* specifies a separate image XObject which shall be used as an *explicit mask* specifying which areas of the image to paint and which to mask out.

- Alternatively, the **Mask** entry *(PDF 1.3)* specifies a range of colours which shall be masked out wherever they occur within the image. This technique is known as *colour key masking*.

NOTE 5 Earlier versions of PDF commonly simulated masking by defining a clipping path enclosing only those of an image's samples that are to be painted. However, if the clipping path is very complex (or if there is more than one clipping path) not all conforming Readers will render the results in the same way. An alternative way to achieve the effect of an explicit mask is to define the image being clipped as a pattern, make it the current colour, and then paint the explicit mask as an image whose **ImageMask** entry is **true**.

In the transparent imaging model, a fourth type of masking effect, *soft masking*, is available through the **SMask** entry *(PDF 1.4)* or the **SMaskInData** entry *(PDF 1.5)* in the image dictionary; see 11.6.5, "Specifying Soft Masks", for further discussion.

8.9.6.2 Stencil Masking

An *image mask* (an image XObject whose **ImageMask** entry is **true**) is a monochrome image in which each sample is specified by a single bit. However, instead of being painted in opaque black and white, the image mask is treated as a *stencil mask* that is partly opaque and partly transparent. Sample values in the image do not represent black and white pixels; rather, they designate places on the page that should either be marked with the current colour or masked out (not marked at all). Areas that are masked out retain their former contents. The effect is like applying paint in the current colour through a cut-out stencil, which lets the paint reach the page in some places and masks it out in others.

An image mask differs from an ordinary image in the following significant ways:

- The image dictionary shall not contain a **ColorSpace** entry because sample values represent masking properties (1 bit per sample) rather than colours.

- The value of the **BitsPerComponent** entry shall be 1.

- The **Decode** entry determines how the source samples shall be interpreted. If the **Decode** array is [0 1] (the default for an image mask), a sample value of 0 shall mark the page with the current colour, and a 1 shall leave the previous contents unchanged. If the **Decode** array is [1 0], these meanings shall be reversed.

NOTE 6 One of the most important uses of stencil masking is for painting character glyphs represented as bitmaps. Using such a glyph as a stencil mask transfers only its “black” bits to the page, leaving the “white” bits (which are really just background) unchanged. For reasons discussed in 9.6.5, "Type 3 Fonts", an image mask, rather than an image, should almost always be used to paint glyph bitmaps.

If image interpolation (see 8.9.5.3, "Image Interpolation") is requested during stencil masking, the effect shall be to smooth the edges of the mask, not to interpolate the painted colour values. This effect can minimize the jaggy appearance of a low-resolution stencil mask.

8.9.6.3 Explicit Masking

In PDF 1.3, the **Mask** entry in an image dictionary may be an image mask, as described in sub-clause 8.9.6.2, "Stencil Masking", which serves as an *explicit mask* for the primary (base) image. The base image and the image mask need not have the same resolution (**Width** and **Height** values), but since all images shall be defined on the unit square in user space, their boundaries on the page will coincide; that is, they will overlay each other. The image mask indicates which places on the page shall be painted and which shall be masked out (left unchanged). Unmasked areas shall be painted with the corresponding portions of the base image; masked areas shall not be.

8.9.6.4 Colour Key Masking

In PDF 1.3, the **Mask** entry in an image dictionary may be an array specifying a range of colours to be masked out. Samples in the image that fall within this range shall not be painted, allowing the existing background to show through.

NOTE 1 The effect is similar to that of the video technique known as chroma-key.

For colour key masking, the value of the **Mask** entry shall be an array of 2 ¥ *n* integers, [min_1 max_1 … min_n max_n], where *n* is the number of colour components in the image’s colour space. Each integer shall be in the range 0 to $2^{\text{BitsPerComponent}}$ - 1, representing colour values *before* decoding with the **Decode** array. An image sample shall be masked (not painted) if all of its colour components before decoding, $c_1 \ldots c_n$, fall within the specified ranges (that is, if min_i £ c_i £ max_i for all 1 £ *i* £ *n*).

When colour key masking is specified, the use of a ***DCTDecode*** or lossy ***JPXDecode*** filter for the stream can produce unexpected results.

NOTE 2 ***DCTDecode*** is always a lossy filter while ***JPXDecode*** has a lossy filter option. The use of a lossy filter mean that the output is only an approximation of the original input data. Therefore, the use of this filter may lead to slight changes in the colour values of image samples, possibly causing samples that were intended to be masked to be unexpectedly painted instead, in colours slightly different from the mask colour.

8.9.7 Inline Images

As an alternative to the image XObjects described in 8.9.5, "Image Dictionaries", a sampled image may be specified in the form of an *inline image*. This type of image shall be defined directly within the content stream in which it will be painted rather than as a separate object. Because the inline format gives the reader less flexibility in managing the image data, it shall be used only for small images (4 KB or less).

An inline image object shall be delimited in the content stream by the operators **BI** (begin image), **ID** (image data), and **EI** (end image). These operators are summarized in Table 92. **BI** and **ID** shall bracket a series of key-value pairs specifying the characteristics of the image, such as its dimensions and colour space; the image data shall follow between the **ID** and **EI** operators. The format is thus analogous to that of a stream object such as an image XObject:

BI
…Key-value pairs…
ID
…Image data…
EI

Table 92 – Inline Image Operators

Operands	Operator	Description
—	**BI**	Begin an inline image object.
—	**ID**	Begin the image data for an inline image object.
—	**EI**	End an inline image object.

Inline image objects shall not be nested; that is, two **BI** operators shall not appear without an intervening **EI** to close the first object. Similarly, an **ID** operator shall only appear between a **BI** and its balancing **EI**. Unless the image uses **ASCIIHexDecode** or **ASCII85Decode** as one of its filters, the **ID** operator shall be followed by a single white-space character, and the next character shall be interpreted as the first byte of image data.

The key-value pairs appearing between the **BI** and **ID** operators are analogous to those in the dictionary portion of an image XObject (though the syntax is different). Table 93 shows the entries that are valid for an inline image, all of which shall have the same meanings as in a stream dictionary (see Table 5) or an image dictionary (see Table 89). Entries other than those listed shall be ignored; in particular, the **Type**, **Subtype**, and **Length** entries normally found in a stream or image dictionary are unnecessary. For convenience, the abbreviations shown in the table may be used in place of the fully spelled-out keys. Table 94 shows additional abbreviations that can be used for the names of colour spaces and filters.

These abbreviations are valid only in inline images; they shall not be used in image XObjects. **JBIG2Decode** and **JPXDecode** are not listed in Table 94 because those filters shall not be used with inline images.

Table 93 – Entries in an Inline Image Object

Full Name	Abbreviation
BitsPerComponent	**BPC**
ColorSpace	**CS**
Decode	**D**
DecodeParms	**DP**
Filter	**F**
Height	**H**
ImageMask	**IM**
Intent *(PDF 1.1)*	No abbreviation
Interpolate	**I** (uppercase I)
Width	**W**

Table 94 – Additional Abbreviations in an Inline Image Object

Full Name	Abbreviation
DeviceGray	**G**
DeviceRGB	**RGB**

Table 94 – Additional Abbreviations in an Inline Image Object (continued)

Full Name	Abbreviation
DeviceCMYK	**CMYK**
Indexed	**I** (uppercase I)
ASCIIHexDecode	**AHx**
ASCII85Decode	**A85**
LZWDecode	**LZW**
FlateDecode *(PDF 1.2)*	**Fl** (uppercase F, lowercase L)
RunLengthDecode	**RL**
CCITTFaxDecode	**CCF**
DCTDecode	**DCT**

The colour space specified by the **ColorSpace** (or **CS**) entry shall be one of the standard device colour spaces (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**). It shall not be a CIE-based colour space or a special colour space, with the exception of a limited form of **Indexed** colour space whose base colour space is a device space and whose colour table is specified by a byte string (see 8.6.6.3, "Indexed Colour Spaces"). Beginning with PDF 1.2, the value of the **ColorSpace** entry may also be the name of a colour space in the **ColorSpace** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). In this case, the name may designate any colour space that can be used with an image XObject.

NOTE 1 The names ***DeviceGray***, ***DeviceRGB***, and ***DeviceCMYK*** (as well as their abbreviations ***G***, ***RGB***, and ***CMYK***) always identify the corresponding colour spaces directly; they never refer to resources in the **ColorSpace** subdictionary.

The image data in an inline image may be encoded by using any of the standard PDF filters except JPXDecode and JBIG2Decode. The bytes between the **ID** and **EI** operators shall be treated the same as a stream object's data (see 7.3.8, "Stream Objects"), even though they do not follow the standard stream syntax.

NOTE 2 This is an exception to the usual rule that the data in a content stream shall be interpreted according to the standard PDF syntax for objects.

EXAMPLE This example shows an inline image 17 samples wide by 17 high with 8 bits per component in the **DeviceRGB** colour space. The image has been encoded using LZW and ASCII base-85 encoding. The **cm** operator is used to scale it to a width and height of 17 units in user space and position it at coordinates (298, 388). The **q** and **Q** operators encapsulate the **cm** operation to limit its effect to resizing the image.

```
q                                          % Save graphics state
17  0  0  17  298  388  cm                 % Scale and translate coordinate space

BI                                         % Begin inline image object
   /W  17                                  % Width in samples
   /H  17                                  % Height in samples
   /CS  /RGB                               % Colour space
   /BPC  8                                 % Bits per component
   /F  [/A85  /LZW]                        % Filters

ID                                         % Begin image data
J1/gKA>.]AN&J?]-<HW]aRVcg*bb.\eKAdVV%/PcZ
…Omitted data…
R.s(4KE3&d&7hb*7[%Ct2HCqC~>
EI                                         % End inline image object
Q                                          % Restore graphics state
```

8.10 Form XObjects

8.10.1 General

A *form XObject* is a PDF content stream that is a self-contained description of any sequence of graphics objects (including path objects, text objects, and sampled images). A form XObject may be painted multiple times—either on several pages or at several locations on the same page—and produces the same results each time, subject only to the graphics state at the time it is invoked. Not only is this shared definition economical to represent in the PDF file, but under suitable circumstances the conforming reader can optimize execution by caching the results of rendering the form XObject for repeated reuse.

NOTE 1 The term *form* also refers to a completely different kind of object, an *interactive form* (sometimes called an *AcroForm*), discussed in 12.7, "Interactive Forms". Whereas the form XObjects described in this sub-clause correspond to the notion of forms in the PostScript language, interactive forms are the PDF equivalent of the familiar paper instrument. Any unqualified use of the word *form* is understood to refer to an interactive form; the type of form described here is always referred to explicitly as a *form XObject*.

Form XObjects have various uses:

- As its name suggests, a form XObject may serve as the template for an entire page.

EXAMPLE A program that prints filled-in tax forms can first paint the fixed template as a form XObject and then paint the variable information on top of it.

- Any graphical element that is to be used repeatedly, such as a company logo or a standard component in the output from a computer-aided design system, may be defined as a form XObject.
- Certain document elements that are not part of a page’s contents, such as annotation appearances (see 12.5.5, "Appearance Streams"), shall be represented as form XObjects.
- A specialized type of form XObject, called a *group XObject (PDF 1.4)*, can be used to group graphical elements together as a unit for various purposes (see 8.10.3, "Group XObjects"). In particular, group XObjects shall be used to define transparency groups and soft masks for use in the transparent imaging model (see 11.6.5.2, "Soft-Mask Dictionaries" and 11.6.6, "Transparency Group XObjects").
- Another specialized type of form XObject, a *reference XObject (PDF 1.4)*, may be used to import content from one PDF document into another (see 8.10.4, "Reference XObjects").

A writer shall perform the following two specific operations in order to use a form XObject:

a) *Define the appearance of the form XObject*. A form XObject is a PDF content stream. The dictionary portion of the stream (called the *form dictionary*) shall contain descriptive information about the form XObject; the body of the stream shall describe the graphics objects that produce its appearance. The contents of the form dictionary are described in 8.10.2, "Form Dictionaries".

b) *Paint the form XObject*. The **Do** operator (see 8.8, "External Objects") shall paint a form XObject whose name is supplied as an operand. The name shall be defined in the **XObject** subdictionary of the current resource dictionary. Before invoking this operator, the content stream in which it appears should set appropriate parameters in the graphics state. In particular, it should alter the current transformation matrix to control the position, size, and orientation of the form XObject in user space.

Each form XObject is defined in its own coordinate system, called *form space*. The **BBox** entry in the form dictionary shall be expressed in form space, as shall be any coordinates used in the form XObject’s content stream, such as path coordinates. The **Matrix** entry in the form dictionary shall specify the mapping from form space to the current user space. Each time the form XObject is painted by the **Do** operator, this matrix shall be concatenated with the current transformation matrix to define the mapping from form space to device space.

NOTE 2 This differs from the **Matrix** entry in a pattern dictionary, which maps pattern space to the initial user space of the content stream in which the pattern is used.

When the **Do** operator is applied to a form XObject, a conforming reader shall perform the following tasks:

a) Saves the current graphics state, as if by invoking the **q** operator (see 8.4.4, "Graphics State Operators")

b) Concatenates the matrix from the form dictionary’s **Matrix** entry with the current transformation matrix (CTM)

c) Clips according to the form dictionary’s **BBox** entry

d) Paints the graphics objects specified in the form’s content stream

e) Restores the saved graphics state, as if by invoking the **Q** operator (see 8.4.4, "Graphics State Operators")

Except as described above, the initial graphics state for the form shall be inherited from the graphics state that is in effect at the time **Do** is invoked.

8.10.2 Form Dictionaries

Every form XObject shall have a *form type*, which determines the format and meaning of the entries in its form dictionary. This specification only defines one form type, Type 1. Form XObject dictionaries may contain the entries shown in Table 95, in addition to the usual entries common to all streams (see Table 5).

Table 95 – Additional Entries Specific to a Type 1 Form Dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **XObject** for a form XObject.
Subtype	name	*(Required)* The type of XObject that this dictionary describes; shall be **Form** for a form XObject.
FormType	integer	*(Optional)* A code identifying the type of form XObject that this dictionary describes. The only valid value is 1. Default value: 1.
BBox	rectangle	*(Required)* An array of four numbers in the form coordinate system (see above), giving the coordinates of the left, bottom, right, and top edges, respectively, of the form XObject’s bounding box. These boundaries shall be used to clip the form XObject and to determine its size for caching.
Matrix	array	*(Optional)* An array of six numbers specifying the *form matrix*, which maps form space into user space (see 8.3.4, "Transformation Matrices"). Default value: the identity matrix [1 0 0 1 0 0].
Resources	dictionary	*(Optional but strongly recommended; PDF 1.2)* A dictionary specifying any resources (such as fonts and images) required by the form XObject (see 7.8, "Content Streams and Resources"). In a PDF whose version is 1.1 and earlier, all named resources used in the form XObject shall be included in the resource dictionary of each page object on which the form XObject appears, regardless of whether they also appear in the resource dictionary of the form XObject. These resources should also be specified in the form XObject’s resource dictionary as well, to determine which resources are used inside the form XObject. If a resource is included in both dictionaries, it shall have the same name in both locations. In PDF 1.2 and later versions, form XObjects may be independent of the content streams in which they appear, and this is strongly recommended although not required. In an independent form XObject, the resource dictionary of the form XObject is required and shall contain all named resources used by the form XObject. These resources shall not be promoted to the outer content stream’s resource dictionary, although that stream’s resource dictionary refers to the form XObject.

Table 95 – Additional Entries Specific to a Type 1 Form Dictionary (continued)

Key	Type	Value
Group	dictionary	*(Optional; PDF 1.4)* A *group attributes dictionary* indicating that the contents of the form XObject shall be treated as a group and specifying the attributes of that group (see 8.10.3, "Group XObjects"). If a **Ref** entry (see below) is present, the group attributes shall also apply to the external page imported by that entry, which allows such an imported page to be treated as a group without further modification.
Ref	dictionary	*(Optional; PDF 1.4)* A reference dictionary identifying a page to be imported from another PDF file, and for which the form XObject serves as a proxy (see 8.10.4, "Reference XObjects").
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* containing metadata for the form XObject (see 14.3.2, "Metadata Streams").
PieceInfo	dictionary	*(Optional; PDF 1.3)* A page-piece dictionary associated with the form XObject (see 14.5, "Page-Piece Dictionaries").
LastModified	date	*(Required if **PieceInfo** is present; optional otherwise; PDF 1.3)* The date and time (see 7.9.4, "Dates") when the form XObject's contents were most recently modified. If a page-piece dictionary (**PieceInfo**) is present, the modification date shall be used to ascertain which of the application data dictionaries it contains correspond to the current content of the form (see 14.5, "Page-Piece Dictionaries").
StructParent	integer	*(Required if the form XObject is a structural content item; PDF 1.3)* The integer key of the form XObject's entry in the structural parent tree (see 14.7.4.4, "Finding Structure Elements from Content Items").
StructParents	integer	*(Required if the form XObject contains marked-content sequences that are structural content items; PDF 1.3)* The integer key of the form XObject's entry in the structural parent tree (see 14.7.4.4, "Finding Structure Elements from Content Items"). At most one of the entries **StructParent** or **StructParents** shall be present. A form XObject shall be either a *content* item in its entirety or a container for marked-content sequences that are *content* items, but not both.
OPI	dictionary	*(Optional; PDF 1.2)* An OPI version dictionary for the form XObject (see 14.11.7, "Open Prepress Interface (OPI)").
OC	dictionary	*(Optional; PDF 1.5)* An optional content group or optional content membership dictionary (see 8.11, "Optional Content") specifying the optional content properties for the form XObject. Before the form is processed, its visibility shall be determined based on this entry. If it is determined to be invisible, the entire form shall be skipped, as if there were no **Do** operator to invoke it.
Name	name	*(Required in PDF 1.0; optional otherwise)* The name by which this form XObject is referenced in the **XObject** subdictionary of the current resource dictionary (see 7.8.3, "Resource Dictionaries"). NOTE This entry is obsolescent and its use is no longer recommended.

EXAMPLE The following shows a simple form XObject that paints a filled square 1000 units on each side.

```
6 0 obj                                    % Form XObject
   << /Type /XObject
      /Subtype /Form
      /FormType 1
      /BBox [0 0 1000 1000]
      /Matrix [1 0 0 1 0 0]
      /Resources << /ProcSet [/PDF] >>
```

```
        /Length 58
    >>

stream
    0 0 m
    0 1000 l
    1000 1000 l
    1000 0 l
    f
endstream
endobj
```

8.10.3 Group XObjects

A *group XObject (PDF 1.4)* is a special type of form XObject that can be used to group graphical elements together as a unit for various purposes. It shall be distinguished by the presence of the optional **Group** entry in the form dictionary (see 8.10.2, "Form Dictionaries"). The value of this entry shall be a subsidiary *group attributes dictionary* describing the properties of the group.

As shown in Table 96, every group XObject shall have a *group subtype* (specified by the **S** entry in the group attributes dictionary) that determines the format and meaning of the dictionary's remaining entries. This specification only defines one subtype, a *transparency group XObject* (subtype **Transparency**) representing a transparency group for use in the transparent imaging model (see 11.4, "Transparency Groups"). The remaining contents of this type of dictionary are described in 11.6.6, "Transparency Group XObjects".

Table 96 – Entries Common to all Group Attributes Dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Group** for a group attributes dictionary.
S	name	*(Required)* The *group subtype*, which identifies the type of group whose attributes this dictionary describes and determines the format and meaning of the dictionary's remaining entries. The only group subtype defined is **Transparency**; see 11.6.6, "Transparency Group XObjects", for the remaining contents of this type of dictionary.

8.10.4 Reference XObjects

8.10.4.1 General

Reference XObjects (PDF 1.4) enable one PDF document to import content from another. The document in which the reference occurs is called the *containing document*; the one whose content is being imported is the *target document*. The target document may reside in a file external to the containing document or may be included within it as an embedded file stream (see 7.11.4, "Embedded File Streams").

The reference XObject in the containing document shall be a form XObject containing the **Ref** entry in its form dictionary, as described below. This form XObject shall serve as a *proxy* that shall be displayed or printed by a conforming reader in place of the imported content.

NOTE 3 The proxy might consist of a low-resolution image of the imported content, a piece of descriptive text referring to it, a gray box to be displayed in its place, or any other similar placeholder.

Conforming readers that do not recognize the **Ref** entry shall simply display or print the proxy as an ordinary form XObject. Those readers that do implement reference XObjects shall use the proxy in place of the imported content if the latter is unavailable. A conforming reader may also provide a user interface to allow editing and updating of imported content links.

The imported content shall consist of a single, complete PDF page in the target document. It shall be designated by a *reference dictionary*, which in turn shall be the value of the **Ref** entry in the reference XObject's form dictionary (see 8.10.2, "Form Dictionaries"). The presence of the **Ref** entry shall distinguish reference XObjects from other types of form XObjects. Table 97 shows the contents of the reference dictionary.

Table 97 – Entries in a Reference Dictionary

Key	Type	Value
F	file specification	*(Required)* The file containing the target document.
Page	integer or text string	*(Required)* A page index or page label (see 12.4.2, "Page Labels") identifying the page of the target document containing the content to be imported. This reference is a weak one and may be inadvertently invalidated if the referenced page is changed or replaced in the target document after the reference is created.
ID	array	*(Optional)* An array of two byte strings constituting a file identifier (see 14.4, "File Identifiers") for the file containing the target document. The use of this entry improves an reader's chances of finding the intended file and allows it to warn the user if the file has changed since the reference was created.

When the imported content replaces the proxy, it shall be transformed according to the proxy object's transformation matrix and clipped to the boundaries of its bounding box, as specified by the **Matrix** and **BBox** entries in the proxy's form dictionary (see 8.10.2, "Form Dictionaries"). The combination of the proxy object's matrix and bounding box thus implicitly defines the bounding box of the imported page. This bounding box typically coincides with the imported page's crop box or art box (see 14.11.2, "Page Boundaries"), but may not correspond to any of the defined page boundaries. If the proxy object's form dictionary contains a **Group** entry, the specified group attributes shall apply to the imported page as well, which allows the imported page to be treated as a group without further modification.

8.10.4.2 Printing Reference XObjects

When printing a page containing reference XObjects, an application may emit any of the following items, depending on the capabilities of the conforming reader, the user's preferences, and the nature of the print job:

- The imported content designated by the reference XObject
- The reference XObject as a proxy for the imported content
- An OPI proxy or substitute image taken from the reference XObject's OPI dictionary, if any (see 14.11.7, "Open Prepress Interface (OPI)")

The imported content or the reference XObject may also be emitted, by a conforming reader, in place of an OPI proxy when generating OPI comments in a PostScript output stream.

8.10.4.3 Special Considerations

Certain special considerations arise when reference XObjects interact with other PDF features:

- When the page imported by a reference XObject contains annotations (see 12.5, "Annotations"), all annotations that contain a printable, unhidden, visible appearance stream (12.5.5, "Appearance Streams") shall be included in the rendering of the imported page. If the proxy is a snapshot image of the imported page, it shall also include the annotation appearances. These appearances shall therefore be converted into part of the proxy's content stream, either as subsidiary form XObjects or by flattening them directly into the content stream.
- Logical structure information associated with a page (see 14.7, "Logical Structure") may be ignored when importing the page into another document with a reference XObject. In a target document with multiple

pages, structure elements occurring on the imported page are typically part of a larger structure pertaining to the document as a whole; such elements cannot meaningfully be incorporated into the structure of the containing document. In a one-page target document or one made up of independent, structurally unrelated pages, the logical structure for the imported page may be wholly self-contained; in this case, it may be possible to incorporate this structure information into that of the containing document. However, PDF provides no mechanism for the logical structure hierarchy of one document to refer indirectly to that of another.

8.11 Optional Content

8.11.1 General

Optional content (PDF 1.5) refers to sub-clauses of content in a PDF document that can be selectively viewed or hidden by document authors or consumers. This capability is useful in items such as CAD drawings, layered artwork, maps, and multi-language documents.

NOTE The following sub-clauses describe the PDF structures used to implement optional content:

8.11.2, "Optional Content Groups", describes the primary structures used to control the visibility of content.

8.11.3, "Making Graphical Content Optional", describes how individual pieces of content in a document may declare themselves as belonging to one or more optional content groups.

8.11.4, "Configuring Optional Content", describes how the states of optional content groups are set.

8.11.2 Optional Content Groups

8.11.2.1 General

An optional content group is a dictionary representing a collection of graphics that can be made visible or invisible dynamically by users of conforming readers. The graphics belonging to such a group may reside anywhere in the document: they need not be consecutive in drawing order, nor even belong to the same content stream. Table 98 shows the entries in an optional content group dictionary.

Table 98 – Entries in an Optional Content Group Dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **OCG** for an optional content group dictionary.
Name	text string	*(Required)* The name of the optional content group, suitable for presentation in a reader's user interface.
Intent	name or array	*(Optional)* A single intent name or an array containing any combination of names. PDF defines two names, *View* and *Design*, that may indicate the intended use of the graphics in the group. A conforming reader may choose to use only groups that have a specific intent and ignore others. Default value: *View*. See 8.11.2.3, "Intent" for more information.
Usage	dictionary	*(Optional)* A *usage dictionary* describing the nature of the content controlled by the group. It may be used by features that automatically control the state of the group based on outside factors. See 8.11.4.4, "Usage and Usage Application Dictionaries" for more information.

In its simplest form, each dictionary shall contain a **Type** entry and a **Name** for presentation in a user interface. It may also have an **Intent** entry that may describe its intended use (see 8.11.2.3, "Intent") and a **Usage** entry that shall describe the nature of its content (see 8.11.4.4, "Usage and Usage Application Dictionaries").

Individual content elements in a document may specify the optional content group or groups that affect their visibility (see 8.11.3, "Making Graphical Content Optional"). Any content whose visibility shall be affected by a given optional content group is said to belong to that group.

A group shall be assigned a state, which is either **ON** or **OFF**. States themselves are not part of the PDF document but may be set programmatically or through the reader's user interface to change the visibility of content. When a document is first opened by a conforming reader, the groups' states shall be initialized based on the document's default configuration dictionary (see 8.11.4.3, "Optional Content Configuration Dictionaries").

Content belonging to a group shall be visible when the group is **ON** and invisible when it is **OFF**. Content may belong to multiple groups, which may have conflicting states. These cases shall be described by the use of optional content membership dictionaries, described in the next sub-clause.

8.11.2.2 Optional Content Membership Dictionaries

As mentioned above, content may belong to a single optional content group and shall be visible when the group is **ON** and invisible when it is **OFF**. To express more complex visibility policies, content shall not declare itself to belong directly to an optional content group but rather to an *optional content membership dictionary*, whose entries are shown in Table 99.

NOTE 1 8.11.3, "Making Graphical Content Optional" describes how content declares its membership in a group or membership dictionary.

Table 99 – Entries in an Optional Content Membership Dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **OCMD** for an optional content membership dictionary.
OCGs	dictionary or array	*(Optional)* A dictionary or array of dictionaries specifying the optional content groups whose states shall determine the visibility of content controlled by this membership dictionary. Null values or references to deleted objects shall be ignored. If this entry is not present, is an empty array, or contains references only to null or deleted objects, the membership dictionary shall have no effect on the visibility of any content.
P	name	*(Optional)* A name specifying the *visibility policy* for content belonging to this membership dictionary. Valid values shall be: **AllOn** visible only if all of the entries in **OCGs** are **ON** **AnyOn** visible if any of the entries in **OCGs** are **ON** **AnyOff** visible if any of the entries in **OCGs** are **OFF** **AllOff** visible only if all of the entries in **OCGs** are **OFF** Default value: **AnyOn**
VE	array	*(Optional; PDF 1.6)* An array specifying a *visibility expression*, used to compute visibility of content based on a set of optional content groups; see discussion below.

An optional content membership dictionary may express its visibility policy in two ways:

- The **P** entry may specify a simple boolean expression indicating how the optional content groups specified by the **OCGs** entry determine the visibility of content controlled by the membership dictionary.
- PDF 1.6 introduced the **VE** entry, which is a visibility expression that may be used to specify an arbitrary boolean expression for computing the visibility of content from the states of optional content groups.

NOTE 2 Since the ***VE*** entry is more general, if it is present and supported by the conforming reader software, it should be used in preference to ***OCGs*** and ***P***. However, for compatibility purposes, conforming writers should use ***OCGs*** and ***P*** entries where possible. When the use of ***VE*** is necessary to express the intended behaviour, ***OCGs*** and ***P*** entries should also be provided to approximate the behaviour in non-conforming reader software.

A visibility expression is an array with the following characteristics:

- Its first element shall be a name representing a boolean operator (**And**, **Or**, or **Not**).
- Subsequent elements shall be either optional content groups or other visibility expressions.
- If the first element is **Not**, it shall have only one subsequent element. If the first element is **And** or **Or**, it shall have one or more subsequent elements.
- In evaluating a visibility expression, the **ON** state of an optional content group shall be equated to the boolean value **true**; **OFF** shall be equated to **false**.

Membership dictionaries are useful in cases such as these:

- Some content may choose to be *invisible* when a group is **ON** and *visible* when it is **OFF**. In this case, the content would belong to a membership dictionary whose **OCGs** entry consists of a single optional content group and whose **P** entry is **AnyOff** or **AllOff**.

NOTE 3 It is legal to have an ***OCGs*** entry consisting of a single group and a *P* entry that is ***AnyOn*** or ***AllOn***. However, in this case it is preferable to use an optional content group directly because it uses fewer objects.

- Some content may belong to more than one group and needs to specify its policy when the groups are in conflicting states. In this case, the content would belong to a membership dictionary whose **OCGs** entry consists of an array of optional content groups and whose **P** entry specifies the visibility policy, as illustrated in EXAMPLE 1 in this sub-clause. EXAMPLE 2 in this sub-clause shows the equivalent policy using visibility expressions.

EXAMPLE 1 This example shows content belonging to a membership dictionary whose **OCGs** entry consists of an array of optional content groups and whose **P** entry specifies the visibility policy.

```
<< /Type /OCMD                    % Content belonging to this optional content
                                  % membership dictionary is controlled by the states
   /OCGs [12 0 R 13 0 R 14 0 R]   % of three optional content groups.
   /P /AllOn                      % Content is visible only if the state of all three
>>                                % groups is ON; otherwise it's hidden.
```

EXAMPLE 2 This example shows a visibility expression equivalent to EXAMPLE 1 in this sub-clause

```
<< /Type /OCMD
   /VE [/And 12 0 R 13 0 R 14 0 R]   % Visibility expression equivalent to EXAMPLE 1.
>>
```

EXAMPLE 3 This example shows a more complicated visibility expression based on five optional content groups, represented by objects 1 through 5. It is equivalent to

"OCG 1" OR (NOT "OCG 2") OR ("OCG 3" AND "OCG 4" AND "OCG 5")

```
<< /Type /OCMD
   /VE [/Or                       % Visibility expression: OR
         1 0 R                    % OCG 1
         [/Not 2 0 R]             % NOT OCG 2
         [/And 3 0 R 4 0 R 5 0 R] % OCG 3 AND OCG 4 AND OCG 5
      ]
>>
```

8.11.2.3 Intent

PDF defines two intents: *Design*, which may be used to represent a document designer's structural organization of artwork, and *View*, which may be used for interactive use by document consumers. A conforming writer shall not use a value other than *Design* or *View*.

NOTE The **Intent** entry in Table 98 provides a way to distinguish between different intended uses of optional content. For example, many document design applications, such as CAD packages, offer layering features for collecting groups of graphics together and selectively hiding or viewing them for the convenience of the author. However, this layering may be different (at a finer granularity, for example) than would be useful to consumers of the document. Therefore, it is possible to specify different intents for optional content groups within a single document. A conforming reader may decide to use only groups that are of a specific intent.

Configuration dictionaries (see 8.11.4.3, "Optional Content Configuration Dictionaries") may also contain an **Intent** entry. If one or more of a group's intents is contained in the current configuration's set of intents, the group shall be used in determining visibility. If there is no match, the group shall have no effect on visibility.

If the configuration's ***Intent*** is an empty array, no groups shall be used in determining visibility; therefore, all content shall be considered visible.

8.11.3 Making Graphical Content Optional

8.11.3.1 General

Graphical content in a PDF file may be made optional by specifying membership in an optional content group or optional content membership dictionary. Two primary mechanisms exist for defining membership:

- Sections of content streams delimited by marked-content operators may be made optional, as described in 8.11.3.2, "Optional Content in Content Streams".
- Form and image XObjects and annotations may be made optional in their entirety by means of a dictionary entry, as described in 8.11.3.3, "Optional Content in XObjects and Annotations".

When a piece of optional content in a PDF file is determined that it shall be hidden, the following occurs:

- The content shall not be drawn.
- Graphics state operations, such as setting the colour, transformation matrix, and clipping, shall still be applied. In addition, graphics state side effects that arise from drawing operators shall be applied; in particular, the current text position shall be updated even for text wrapped in optional content. In other words, graphics state parameters that persist past the end of a marked-content section shall be the same whether the optional content is visible or not.

Hiding a section of optional content shall not change the colour of objects that do not belong to the same optional content group.

- This rule shall also apply to operators that set state that is not strictly graphics state; for example, **BX** and **EX**.
- Objects such as form XObjects and annotations that have been made optional may be skipped entirely, because their contents are encapsulated such that no changes to the graphics state (or other state) persist beyond the processing of their content stream.

Other features in conforming readers, such as searching and editing, may be affected by the ability to selectively show or hide content. A conforming reader may choose whether to use the document's current state of optional content groups (and, correspondingly, the document's visible graphics) or to supply their own states of optional content groups to control the graphics they process.

NOTE 4 Tools to select and move annotations should honour the current on-screen visibility of annotations when performing cursor tracking and mouse-click processing. A full text search engine, however, may need to process all content in a document, regardless of its current visibility on-screen. Export filters might choose the current on-screen visibility, the full content, or present the user with a selection of OCGs to control visibility.

NOTE 5 A non-conforming reader that does not support optional content, such as one that only supports PDF 1.4 functionality, will draw and process all content in a document.

8.11.3.2 Optional Content in Content Streams

Sections of content in a content stream (including a page's **Contents** stream, a form or pattern's content stream, glyph descriptions a Type 3 font as specified by its **CharProcs** entry, or an annotation's appearance) may be made optional by enclosing them between the marked-content operators **BDC** and **EMC** (see 14.6, "Marked Content") with a marked-content tag of OC. In addition, a **DP** marked-content operator may be placed in a page's content stream to force a reference to an optional content group or groups on the page, even when the page has no current content in that layer.

The property list associated with the marked content shall specify either an optional content group or optional content membership dictionary to which the content belongs. Because a group shall be an indirect object and a membership dictionary contains references to indirect objects, the property list shall be a named resource listed in the **Properties** subdictionary of the current resource dictionary (see 14.6.2, "Property Lists"), as shown in EXAMPLE 1 and EXAMPLE 2 in this sub-clause.

Although the marked-content tag shall be *OC*, other applications of marked content are not precluded from using *OC* as a tag. The marked content shall be considered to be for optional content only if the tag is *OC* and the dictionary operand is a valid optional content group or optional content membership dictionary.

NOTE 1 To avoid conflict with other features that used marked content (such as logical structure; see 14.7, "Logical Structure"), the following strategy is recommended:

Where content is to be tagged with optional content markers as well as other markers, the optional content markers should be nested inside the other marked content.

Where optional content and the other markers would overlap but there is not strict containment, the optional content should be broken up into two or more **BDC/EMC** sections, nesting the optional content sections inside the others as necessary. Breaking up optional content spans does not damage the nature of the visibility of the content, whereas the same guarantee cannot be made for all other uses of marked content.

NOTE 2 Any marked content tagged for optional content that is nested inside other marked content tagged for optional content is visible only if all the levels indicate visibility. In other words, if the settings that apply to the outer level indicate that the content should be hidden, the inner level is hidden regardless of its settings.

In the following example, the state of the Show Greeting optional content group directly controls the visibility of the text string “Hello” on the page. When the group is **ON**, the text shall be visible; when the group is **OFF**, the text shall be hidden.

EXAMPLE 1

```
% Within a content stream
...
/OC /oc1 BDC                               % Optional content follows
    BT
        /F1 1 Tf
        12 0 0 12 100 600 Tm
        (Hello) Tj
    ET
EMC                                        % End of optional content
...


<<                                         % In the resources dictionary
    /Properties << /oc1 5 0 R >>           % This dictionary maps the name oc1 to an
...                                        % optional content group (object 5)
>>

5 0 obj                                    % The OCG controlling the visibility
```

```
<<                                          % of the text.
    /Type /OCG
    /Name (Show Greeting)
>>
endobj
```

The example above shows one piece of content associated with one optional content group. There are other possibilities:

- More than one section of content may refer to the same group or membership dictionary, in which case the visibility of both sections is always the same.
- Equivalently, although less space-efficient, different sections may have separate membership dictionaries with the same **OCGs** and **P** entries. The sections shall have identical visibility behaviour.
- Two sections of content may belong to membership dictionaries that refer to the same group(s) but with different **P** settings. For example, if one section has no **P** entry, and the other has a **P** entry of **AllOff**, the visibility of the two sections of content shall be opposite. That is, the first section shall be visible when the second is hidden, and vice versa.

The following example demonstrates both the direct use of optional content groups and the indirect use of groups through a membership dictionary. The content (a black rectangle frame) is drawn if either of the images controlled by the groups named Image A or Image B is shown. If both groups are hidden, the rectangle frame shall be hidden.

EXAMPLE 2

```
% Within a content stream
...
/OC /OC2 BDC                % Draws a black rectangle frame
    0 g
    4 w
    100 100 412 592 re s
EMC

/OC /OC3 BDC                % Draws an image XObject
    q
    412 0 0 592 100 100 cm
    /Im3 Do
    Q
EMC
/OC /OC4 BDC                % Draws an image XObject
    q
    412 0 0 592 100 100 cm
    /Im4 Do
    Q
EMC
...

<<                          % The resource dictionary
    /Properties << /OC2 20 0 R /OC3 30 0 R /OC4 40 0 R >>
    /XObject << /Im3 50 0 R /Im4 /60 0 R >>
>>

20 0 obj
<<                          % Optional content membership dictionary
    /Type /OCMD
    /OCGs [30 0 R 40 0 R]
    /P /AnyOn
>>
endobj
30 0 obj                    % Optional content group “Image A”
<<
    /Type /OCG
    /Name (Image A)
```

```
>>
endobj
40 0 obj                        % Optional content group “Image B”
<<
    /Type /OCG
    /Name (Image B)
>>
endobj
```

8.11.3.3 Optional Content in XObjects and Annotations

In addition to marked content within content streams, form XObjects and image XObjects (see 8.8, "External Objects") and annotations (see 12.5, "Annotations") may contain an **OC** entry, which shall be an optional content group or an optional content membership dictionary.

A form or image XObject's visibility shall be determined by the state of the group or those of the groups referenced by the membership dictionary in conjunction with its **P** (or **VE**) entry, along with the current visibility state in the context in which the XObject is invoked (that is, whether objects are visible in the contents stream at the place where the **Do** operation occurred).

Annotations have various flags controlling on-screen and print visibility (see 12.5.3, "Annotation Flags"). If an annotation contains an **OC** entry, it shall be visible for screen or print only if the flags have the appropriate settings and the group or membership dictionary indicates it shall be visible.

8.11.4 Configuring Optional Content

8.11.4.1 General

A PDF document containing optional content may specify the default states for the optional content groups in the document and indicate which external factors shall be used to alter the states.

NOTE The following sub-clauses describe the PDF structures that are used to specify this information.

8.11.4.2, "Optional Content Properties Dictionary" describes the structure that lists all the optional content groups in the document and their possible configurations.

8.11.4.3, "Optional Content Configuration Dictionaries" describes the structures that specify initial state settings and other information about the groups in the document.

8.11.4.4, "Usage and Usage Application Dictionaries" and 8.11.4.5, "Determining the State of Optional Content Groups" describe how the states of groups can be affected based on external factors.

8.11.4.2 Optional Content Properties Dictionary

The optional **OCProperties** entry in the document catalog (see 7.7.2, "Document Catalog") shall contain, when present, the *optional content properties dictionary*, which contains a list of all the optional content groups in the document, as well as information about the default and alternate configurations for optional content. This dictionary shall be present if the file contains any optional content; if it is missing, a conforming reader shall ignore any optional content structures in the document.

This dictionary contains the following entries:

Table 100 – Entries in the Optional Content Properties Dictionary

Key	Type	Value
OCGs	array	*(Required)* An array of indirect references to all the optional content groups in the document (see 8.11.2, "Optional Content Groups"), in any order. Every optional content group shall be included in this array.

Table 100 – Entries in the Optional Content Properties Dictionary (continued)

Key	Type	Value
D	dictionary	*(Required)* The default viewing optional content configuration dictionary (see 8.11.4.3, "Optional Content Configuration Dictionaries").
Configs	array	(Optional) An array of alternate optional content configuration dictionaries (see 8.11.4.3, "Optional Content Configuration Dictionaries").

8.11.4.3 Optional Content Configuration Dictionaries

The **D** and **Configs** entries in Table 100 are *configuration dictionaries*, which represent different presentations of a document's optional content groups for use by conforming readers. The **D** configuration dictionary shall be used to specify the initial state of the optional content groups when a document is first opened. **Configs** lists other configurations that may be used under particular circumstances. The entries in a configuration dictionary are shown in Table 101.

Table 101 – Entries in an Optional Content Configuration Dictionary

Key	Type	Value
Name	text string	*(Optional)* A name for the configuration, suitable for presentation in a user interface.
Creator	text string	*(Optional)* Name of the application or feature that created this configuration dictionary.
BaseState	name	*(Optional)* Used to initialize the states of all the optional content groups in a document when this configuration is applied. The value of this entry shall be one of the following names: **ON** The states of all groups shall be turned **ON**. **OFF** The states of all groups shall be turned **OFF**. **Unchanged** The states of all groups shall be left unchanged. After this initialization, the contents of the **ON** and **OFF** arrays shall be processed, overriding the state of the groups included in the arrays. Default value: **ON**. If **BaseState** is present in the document's default configuration dictionary, its value shall be *ON*.
ON	array	*(Optional)* An array of optional content groups whose state shall be set to **ON** when this configuration is applied. If the **BaseState** entry is ***ON***, this entry is redundant.
OFF	array	*(Optional)* An array of optional content groups whose state shall be set to **OFF** when this configuration is applied. If the **BaseState** entry is ***OFF***, this entry is redundant.
Intent	name or array	*(Optional)* A single intent name or an array containing any combination of names. it shall be used to determine which optional content groups' states to consider and which to ignore in calculating the visibility of content (see 8.11.2.3, "Intent"). PDF defines two intent names, **View** and **Design**. In addition, the name **All** shall indicate the set of all intents, including those not yet defined. Default value: **View**. The value shall be **View** for the document's default configuration.

Table 101 – Entries in an Optional Content Configuration Dictionary (continued)

Key	Type	Value
AS	array	*(Optional)* An array of usage application dictionaries (see Table 103) specifying which usage dictionary categories (see Table 102) shall be consulted by conforming readers to automatically set the states of optional content groups based on external factors, such as the current system language or viewing magnification, and when they shall be applied.
Order	array	*(Optional)* An array specifying the order for presentation of optional content groups in a conforming reader's user interface. The array elements may include the following objects: Optional content group dictionaries, whose **Name** entry shall be displayed in the user interface by the conforming reader. Arrays of optional content groups which may be displayed by a conforming reader in a tree or outline structure. Each nested array may optionally have as its first element a text string to be used as a non-selectable label in a conforming reader's user interface. Text labels in nested arrays shall be used to present collections of related optional content groups, and not to communicate actual nesting of content inside multiple layers of groups (see EXAMPLE 1 in 8.11.4.3, "Optional Content Configuration Dictionaries"). To reflect actual nesting of groups in the content, such as for layers with sublayers, nested arrays of groups without a text label shall be used (see EXAMPLE 2 in 8.11.4.3, "Optional Content Configuration Dictionaries"). An empty array [] explicitly specifies that no groups shall be presented. In the default configuration dictionary, the default value shall be an empty array; in other configuration dictionaries, the default shall be the **Order** value from the default configuration dictionary. Any groups not listed in this array shall not be presented in any user interface that uses the configuration.
ListMode	name	*(Optional)* A name specifying which optional content groups in the **Order** array shall be displayed to the user. Valid values shall be: **AllPages** Display all groups in the **Order** array. **VisiblePages** Display only those groups in the **Order** array that are referenced by one or more visible pages. Default value: **AllPages**.
RBGroups	array	*(Optional)* An array consisting of one or more arrays, each of which represents a collection of optional content groups whose states shall be intended to follow a radio button paradigm. That is, the state of at most one optional content group in each array shall be **ON** at a time. If one group is turned **ON**, all others shall be turned **OFF**. However, turning a group from **ON** to **OFF** does not force any other group to be turned **ON**. An empty array [] explicitly indicates that no such collections exist. In the default configuration dictionary, the default value shall be an empty array; in other configuration dictionaries, the default is the **RBGroups** value from the default configuration dictionary.

Table 101 – Entries in an Optional Content Configuration Dictionary (continued)

Key	Type	Value
Locked	array	*(Optional; PDF 1.6)* An array of optional content groups that shall be locked when this configuration is applied. The state of a locked group cannot be changed through the user interface of a conforming reader. Conforming writers can use this entry to prevent the visibility of content that depends on these groups from being changed by users. Default value: an empty array. A conforming reader may allow the states of optional content groups from being changed by means other than the user interface, such as JavaScript or items in the **AS** entry of a configuration dictionary.

NOTE EXAMPLE 1 and EXAMPLE 2 in this sub-clause illustrate the use of the **Order** entry to control the display of groups in a user interface.

EXAMPLE 1 Given the following PDF objects:

```
1 0 obj <</Type /OCG /Name (Skin)>> endobj          % Optional content groups
2 0 obj <</Type /OCG /Name (Bones)>> endobj
3 0 obj <</Type /OCG /Name (Bark)>> endobj
4 0 obj <</Type /OCG /Name (Wood)>> endobj

5 0 obj                                              % Configuration dictionary
   << /Order [[(Frog Anatomy) 1 0 R 2 0 R] [(Tree Anatomy) 3 0 R 4 0 R] ] >>
```

A conforming reader should display the optional content groups as follows:

```
Frog Anatomy
   Skin
   Bones
Tree Anatomy
   Bark
   Wood
```

EXAMPLE 2 Given the following PDF objects:

```
                                    % Page contents
/OC /L1 BDC                         % Layer 1
   /OC /L1a BDC                     % Sublayer A of layer 1
      0 0 100 100 re f
   EMC
   /OC /L1b BDC                     % Sublayer B of layer 1
      0 100 100 100 re f
   EMC
EMC
...
<< /L1 1 0 R                        % Resource names
   /L1a 2 0 R
   /L1b 3 0 R
>>
...                                 %Optional content groups
1 0 obj <</Type /OCG /Name (Layer 1)>> endobj
2 0 obj <</Type /OCG /Name (Sublayer A)>> endobj
3 0 obj <</Type /OCG /Name (Sublayer B)>> endobj
...
4 0 obj                             % Configuration dictionary
   << /Order [1 0 R [2 0 R 3 0 R]] >>
```

A conforming reader should display the OCGs as follows:

```
Layer 1
   Sublayer A
   Sublayer B
```

The **AS** entry is an *auto state* array consisting of one or more *usage application dictionaries* that specify how conforming readers shall automatically set the state of optional content groups based on external factors, as discussed in the following sub-clause.

8.11.4.4 Usage and Usage Application Dictionaries

Optional content groups are typically constructed to control the visibility of graphic objects that are related in some way. Objects can be related in several ways; for example, a group may contain content in a particular language or content suitable for viewing at a particular magnification.

An optional content group's usage dictionary (the value of the **Usage** entry in an optional content group dictionary; see Table 98) shall contain information describing the nature of the content controlled by the group. This dictionary can contain any combination of the entries shown in Table 102.

Table 102 – Entries in an Optional Content Usage Dictionary

Key	Type	Value
CreatorInfo	dictionary	*(Optional)* A dictionary used by the creating application to store application-specific data associated with this optional content group. It shall contain two required entries: **Creator** A text string specifying the application that created the group. **Subtype** A name defining the type of content controlled by the group. Suggested values include but shall not be limited to **Artwork**, for graphic-design or publishing applications, and **Technical**, for technical designs such as building plans or schematics. Additional entries may be included to present information relevant to the creating application or related applications. Groups whose **Intent** entry contains ***Design*** typically include a **CreatorInfo** entry.
Language	dictionary	*(Optional)* A dictionary specifying the language of the content controlled by this optional content group. It may contain the following two entries: **Lang** *(required)* A text string that specifies a language and possibly a locale (see 14.9.2, "Natural Language Specification"). For example, es-MX represents Mexican Spanish. **Preferred** *(optional)* A name whose values shall be either **ON** or **OFF**. Default value: **OFF**. it shall be used by conforming readers when there is a partial match but no exact match between the system language and the language strings in all usage dictionaries. See 8.11.4.4, "Usage and Usage Application Dictionaries" for more information.
Export	dictionary	*(Optional)* A dictionary containing one entry, **ExportState**, a name whose value shall be either **ON** or **OFF**. This value shall indicate the recommended state for content in this group when the document (or part of it) is saved by a conforming reader to a format that does not support optional content (for example, a raster image format).
Zoom	dictionary	*(Optional)* A dictionary specifying a range of magnifications at which the content in this optional content group is best viewed. It shall contain one or both of the following entries: **min** The minimum recommended magnification factor at which the group shall be **ON**. Default value: 0. **max** The magnification factor below which the group shall be **ON**. Default value: infinity.

Table 102 – Entries in an Optional Content Usage Dictionary (continued)

Key	Type	Value
Print	dictionary	*(Optional)* A dictionary specifying that the content in this group is shall be used when printing. It may contain the following optional entries: **Subtype** A name object specifying the kind of content controlled by the group; for example, **Trapping**, **PrintersMarks** and **Watermark**. **PrintState** A name that shall be either **ON** or **OFF**, indicating that the group shall be set to that state when the document is printed from a conforming reader.
View	dictionary	*(Optional)* A dictionary that shall have a single entry, **ViewState**, a name that shall have a value of either **ON** or **OFF**, indicating that the group shall be set to that state when the document is opened in a conforming reader.
User	dictionary	*(Optional)* A dictionary specifying one or more users for whom this optional content group is primarily intended. Each dictionary shall have two required entries: **Type** A name object that shall be either **Ind** (individual), **Ttl** (title), or **Org** (organization). **Name** A text string or array of text strings representing the name(s) of the individual, position or organization.
PageElement	dictionary	*(Optional)* A dictionary declaring that the group contains a pagination artifact. It shall contain one entry, **Subtype**, whose value shall be a name that is either **HF** (header/footer), **FG** (foreground image or graphic), **BG** (background image or graphic), or **L** (logo).

While the data in the usage dictionary serves as information for a document user to examine, it may also be used by conforming readers to automatically manipulate the state of optional content groups based on external factors such as current system language settings or zoom level. Document authors may use *usage application dictionaries* to specify which entries in the usage dictionary shall be consulted to automatically set the state of optional content groups based on such factors. Usage application dictionaries shall be listed in the **AS** entry in an optional content configuration dictionary (see Table 101). If no **AS** entry is present, states shall not be automatically adjusted based on usage information.

A usage application dictionary specifies the rules for which usage entries shall be used by conforming readers to automatically manipulate the state of optional content groups, which groups shall be affected, and under which circumstances. Table 103 shows the entries in a usage application dictionary.

Usage application dictionaries shall only be used by interactive conforming readers, and shall not be used by applications that use PDF as final form output (see 8.11.4.5, "Determining the State of Optional Content Groups" for more information).

Table 103 – Entries in a Usage Application Dictionary

Key	Type	Value
Event	name	*(Required)* A name defining the situation in which this usage application dictionary should be used. Shall be one of **View**, **Print**, or **Export**.
OCGs	array	*(Optional)* An array listing the optional content groups that shall have their states automatically managed based on information in their usage dictionary (see 8.11.4.4, "Usage and Usage Application Dictionaries"). Default value: an empty array, indicating that no groups shall be affected.
Category	array	*(Required)* An array of names, each of which corresponds to a usage dictionary entry (see Table 102). When managing the states of the optional content groups in the **OCGs** array, each of the corresponding categories in the group's usage dictionary shall be considered.

The **Event** entry specifies whether the usage settings shall be applied during viewing, printing, or exporting the document. The **OCGs** entry specifies the set of optional content groups to which usage settings shall be applied. For each of the groups in **OCGs**, the entries in its usage dictionary (see Table 102) specified by **Category** shall be examined to yield a *recommended state* for the group. If all the entries yield a recommended state of **ON**, the group's state shall be set to **ON**; otherwise, its state shall be set to **OFF**.

The entries in the usage dictionary shall be used as follows:

- **View**: The state shall be the value of the **ViewState** entry. This entry allows a document to contain content that is relevant only when the document is viewed interactively, such as instructions for how to interact with the document.

- **Print**: The state shall be the value of the **PrintState** entry. If **PrintState** is not present, the state of the optional content group shall be left unchanged.

- **Export**: The state shall be the value of the **ExportState** entry.

- **Zoom**: If the current magnification level of the document is greater than or equal to **min** and less than **max**, the **ON** state shall be used; otherwise, **OFF** shall be used.

- **User**: The **Name** entry shall specify a name or names to match with the user's identification. The **Type** entry determines how the **Name** entry shall be interpreted (name, title, or organization). If there is an exact match, the **ON** state shall be used; otherwise **OFF** shall be used.

- **Language**: This category shall allow the selection of content based on the language and locale of the application. If an exact match to the language and locale is found among the **Lang** entries of the optional content groups in the usage application dictionary's **OCGs** list, all groups that have exact matches shall receive an **ON** recommendation. If no exact match is found, but a partial match is found (that is, the language matches but not the locale), all partially matching groups that have **Preferred** entries with a value of **ON** shall receive an **ON** recommendation. All other groups shall receive an **OFF** recommendation.

There shall be no restriction on multiple entries with the same value of ***Event***, in order to allow documents with incompatible usage application dictionaries to be combined into larger documents and have their behaviour preserved. If a given optional content group appears in more than one ***OCGs*** array, its state shall be ***ON*** only if all categories in all the usage application dictionaries it appears in shall have a state of ***ON***.

EXAMPLE This example shows the use of an auto state array with usage application dictionaries. The **AS** entry in the default configuration dictionary is an array of three usage application dictionaries, one for each of the **Event** values **View**, **Print**, and **Export**.

```
/OCProperties                          % OCProperties dictionary in document catalog
   << /OCGs [1 0 R 2 0 R 3 0 R 4 0 R]
      /D  << /BaseState /OFF           % The default configuration
            /ON [1 0 R]
            /AS [                      % Auto state array of usage application dictionaries
               << /Event /View /Category [/Zoom] /OCGs [1 0 R 2 0 R 3 0 R 4 0 R] >>
               << /Event /Print /Category [/Print] /OCGs [4 0 R] >>
               << /Event /Export /Category [/Export] /OCGs [3 0 R 4 0 R] >>
               ]
         >>
   >>
...

1 0 obj
   << /Type /OCG
      /Name (20000 foot view)
      /Usage << /Zoom << /max 1.0 >> >>
   >>
endobj
2 0 obj
   << /Type /OCG
```

```
        /Name (10000 foot view)
        /Usage << /Zoom << /min 1.0 /max 2.0 >> >>
>>
endobj
3 0 obj
    << /Type /OCG
        /Name (1000 foot view)
        /Usage << /Zoom << /min 2.0 /max 20.0 >>
                    /Export << /ExportState /OFF >> >>
        >>
endobj
4 0 obj
    << /Type /OCG
        /Name (Copyright notice)
        /Usage << /Print << /PrintState /ON >>
                    /Export << /ExportState /ON>> >>
    >>
endobj
```

In the example, the usage application dictionary with event type **View** specifies that all optional content groups shall have their states managed based on zoom level when viewing. Three groups (objects 1, 2, and 3) contain **Zoom** usage information. Object 4 has none; therefore, it shall not be affected by zoom level changes. Object 3 shall receive an **OFF** recommendation when exporting. When printing or exporting, object 4 shall receive an **ON** recommendation.

8.11.4.5 Determining the State of Optional Content Groups

This sub-clause summarizes the rules by which conforming readers make use of the configuration and usage application dictionaries to set the state of optional content groups. For purposes of this discussion, it is useful to distinguish the following types of conforming readers:

- Viewer applications which allow users to interact with the document in various ways.
- Design applications, which offer layering features for collecting groups of graphics together and selectively hiding or viewing them.

NOTE 1 The following rules are not meant to apply to design applications; they may manage their states in an entirely different manner if they choose.

- Aggregating applications, which import PDF files as graphics.
- Printing applications, which print PDF files.

When a document is first opened, its optional content groups shall be assigned a state based on the **D** (default) configuration dictionary in the **OCProperties** dictionary:

a) The value of **BaseState** shall be applied to all the groups.

b) The groups listed in either the **ON** or **OFF** array (depending on which one is opposite to **BaseState**) shall have their states adjusted.

This state shall be the state used by printing and aggregating application. Such applications shall not apply the changes based on usage application dictionaries described below. However, for more advanced functionality, they may provide user control for manipulating the individual states of optional content groups.

NOTE 2 Viewer applications may also provide users with an option to view documents in this state (that is, to disable the automatic adjustments discussed below). This option permits an accurate preview of the content as it will appear when placed into an aggregating application or sent to a stand-alone printing system.

The remaining discussion in this sub-clause applies only to viewer applications. Such applications shall examine the **AS** array for usage application dictionaries that have an **Event** of type **View**. For each one found, the groups listed in its **OCGs** array shall be adjusted as described in 8.11.4.4, "Usage and Usage Application Dictionaries".

Subsequently, the document is ready for interactive viewing by a user. Whenever there is a change to a factor that the usage application dictionaries with event type **View** depend on (such as zoom level), the corresponding dictionaries shall be reapplied.

The user may manipulate optional content group states manually or by triggering **SetOCGState** actions (see 12.6.4.12, "Set-OCG-State Actions") by, for example, clicking links or bookmarks. Manual changes shall override the states that were set automatically. The states of these groups remain overridden and shall not be readjusted based on usage application dictionaries with event type **View** as long as the document is open (or until the user reverts the document to its original state).

When a document is printed by a viewer application, usage application dictionaries with an event type **Print** shall be applied over the current states of optional content groups. These changes shall persist only for the duration of the print operation; then all groups shall revert to their prior states.

Similarly, when a document is exported to a format that does not support optional content, usage application dictionaries with an event type **Export** shall be applied over the current states of optional content groups. Changes shall persist only for the duration of the export operation; then all groups shall revert to their prior states.

NOTE 3 Although the event types ***Print*** and ***Export*** have identically named counterparts that are usage categories, the corresponding usage application dictionaries are permitted to specify that other categories may be applied.

9 Text

9.1 General

This clause describes the special facilities in PDF for dealing with text—specifically, for representing characters with glyphs from fonts. A glyph is a graphical shape and is subject to all graphical manipulations, such as coordinate transformation. Because of the importance of text in most page descriptions, PDF provides higher-level facilities to describe, select, and render glyphs conveniently and efficiently.

The first sub-clause is a general description of how glyphs from fonts are painted on the page. Subsequent sub-clauses cover these topics in detail:

- *Text state*. A subset of the graphics state parameters pertain to text, including parameters that select the font, scale the glyphs to an appropriate size, and accomplish other graphical effects.
- *Text objects and operators*. The text operators specify the glyphs to be painted, represented by string objects whose values shall be interpreted as sequences of character codes. A text object encloses a sequence of text operators and associated parameters.
- *Font data structures*. Font dictionaries and associated data structures provide information that a conforming reader needs to interpret the text and position the glyphs properly. The definitions of the glyphs themselves shall be contained in *font programs*, which may be embedded in the PDF file, built into a conforming reader, or obtained from an external font file.

9.2 Organization and Use of Fonts

9.2.1 General

A *character* is an abstract symbol, whereas a *glyph* is a specific graphical rendering of a character.

EXAMPLE 1 The glyphs A, **A**, and *A* are renderings of the abstract “A” character.

NOTE 1 Historically these two terms have often been used interchangeably in computer typography (as evidenced by the names chosen for some PDF dictionary keys and PostScript operators), but advances in this area have made the distinction more meaningful. Consequently, this standard distinguishes between characters and glyphs, though with some residual names that are inconsistent.

Glyphs are organized into fonts. A *font* defines glyphs for a particular character set.

EXAMPLE 2 The Helvetica and Times fonts define glyphs for a set of standard Latin characters.

A font for use with a conforming reader is prepared in the form of a program. Such a *font program* shall be written in a special-purpose language, such as the *Type 1*, *TrueType*, or *OpenType* font format, that is understood by a specialized font interpreter.

In PDF, the term *font* refers to a *font dictionary*, a PDF object that identifies the font program and contains additional information about it. There are several different font types, identified by the **Subtype** entry of the font dictionary.

For most font types, the font program shall be defined in a separate *font file*, which may be either embedded in a PDF stream object or obtained from an external source. The font program contains *glyph descriptions* that generate glyphs.

A content stream paints glyphs on the page by specifying a font dictionary and a string object that shall be interpreted as a sequence of one or more character codes identifying glyphs in the font. This operation is called *showing* the text string; the text strings drawn in this way are called *show strings*. The glyph description consists of a sequence of graphics operators that produce the specific shape for that character in this font. To render a glyph, the conforming reader executes the glyph description.

NOTE 2 Programmers who have experience with scan conversion of general shapes may be concerned about the amount of computation that this description seems to imply. However, this is only the abstract behaviour of glyph descriptions and font programs, not how they are implemented. In fact, an efficient implementation can be achieved through careful caching and reuse of previously rendered glyphs.

9.2.2 Basics of Showing Text

EXAMPLE 1 This example illustrates the most straightforward use of a font. The text ABC is placed 10 inches from the bottom of the page and 4 inches from the left edge, using 12-point Helvetica.

```
BT
   /F13 12 Tf
   288 720 Td
   (ABC) Tj
ET
```

The five lines of this example perform these steps:

a) Begin a text object.

b) Set the font and font size to use, installing them as parameters in the text state. In this case, the font resource identified by the name F13 specifies the font externally known as Helvetica.

c) Specify a starting position on the page, setting parameters in the text object.

d) Paint the glyphs for a string of characters at that position.

e) End the text object.

These paragraphs explain these operations in more detail.

To paint glyphs, a content stream shall first identify the font to be used. The **Tf** operator shall specify the name of a font resource—that is, an entry in the **Font** subdictionary of the current resource dictionary. The value of that entry shall be a font dictionary. The font dictionary shall identify the font's externally known name, such as Helvetica, and shall supply some additional information that the conforming reader needs to paint glyphs from that font. The font dictionary may provide the definition of the font program itself.

NOTE 1 The font resource name presented to the ***Tf*** operator is arbitrary, as are the names for all kinds of resources. It bears no relationship to an actual font name, such as Helvetica.

EXAMPLE 2 This Example illustrates an excerpt from the current page's resource dictionary, which defines the font dictionary that is referenced as F13 (see EXAMPLE 1 in this sub-clause).

```
/Resources
   << /Font << /F13 23 0 R >>
   >>

23 0 obj
   << /Type /Font
        /Subtype /Type1
        /BaseFont /Helvetica
   >>
endobj
```

A font defines the glyphs at one standard size. This standard is arranged so that the nominal height of tightly spaced lines of text is 1 unit. In the default user coordinate system, this means the standard glyph size is 1 unit in user space, or 1⁄72 inch. Starting with PDF 1.6, the size of this unit may be specified as greater than 1⁄72 inch by means of the **UserUnit** entry of the page dictionary; see Table 30. The standard-size font shall then be scaled to be usable. The scale factor is specified as the second operand of the **Tf** operator, thereby setting the *text font size* parameter in the graphics state. EXAMPLE 1 in this sub-clause establishes the Helvetica font with a 12-unit size in the graphics state.

Once the font has been selected and scaled, it may be used to paint glyphs. The **Td** operator shall adjust the translation components of the text matrix, as described in 9.4.2, "Text-Positioning Operators". When executed for the first time after **BT**, **Td** shall establish the text position in the current user coordinate system. This determines the position on the page at which to begin painting glyphs.

The **Tj** operator shall take a string operand and shall paint the corresponding glyphs, using the current font and other text-related parameters in the graphics state.

NOTE 2 The **Tj** operator treats each element of the string (an integer in the range 0 to 255) as a character code (see EXAMPLE 1 in this sub-clause).

Each byte shall select a glyph description in the font, and the glyph description shall be executed to paint that glyph on the page. This is the behaviour of **Tj** for simple fonts, such as ordinary Latin text fonts. Interpretation of the string as a sequence of character codes is more complex for composite fonts, described in 9.7, "Composite Fonts".

What these steps produce on the page is not a 12-*point* glyph, but rather a 12-*unit* glyph, where the unit size shall be that of the text space at the time the glyphs are rendered on the page. The actual size of the glyph shall be determined by the text matrix (T_m) in the text object, several text state parameters, and the current transformation matrix (CTM) in the graphics state; see 9.4.4, "Text Space Details".

EXAMPLE 3 If the text space is later scaled to make the unit size 1 centimeter, painting glyphs from the same 12-unit font generates results that are 12 centimeters high.

9.2.3 Achieving Special Graphical Effects

Normal uses of **Tj** and other glyph-painting operators cause black-filled glyphs to be painted. Other effects may be obtained by combining font operators with general graphics operators.

The colour used for painting glyphs shall be the current colour in the graphics state: either the nonstroking colour or the stroking colour (or both), depending on the text rendering mode (see 9.3.6, "Text Rendering Mode"). The default colour shall be black (in DeviceGray), but other colours may be obtained by executing an appropriate colour-setting operator or operators (see 8.6.8, "Colour Operators") before painting the glyphs.

EXAMPLE 1 This example uses text rendering mode 0 and the g operator to fill glyphs in 50 percent gray, as shown in Figure 36.

```
BT
   /F13  48  Tf
   20  40  Td
   0  Tr
   0.5  g
   (ABC)  Tj
ET
```

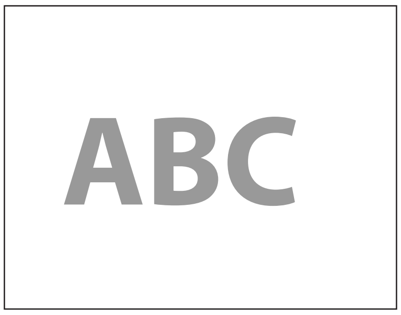


Figure 36 – Glyphs painted in 50% gray

Other graphical effects may be achieved by treating the glyph outline as a path instead of filling it. The *text rendering mode* parameter in the graphics state specifies whether glyph outlines shall be filled, stroked, used as a clipping boundary, or some combination of these effects. Only a subset of the possible rendering modes apply to Type 3 fonts.

EXAMPLE 2 This example treats glyph outlines as a path to be stroked. The Tr operator sets the text rendering mode to 1 (stroke). The w operator sets the line width to 2 units in user space. Given those graphics state parameters, the Tj operator strokes the glyph outlines with a line 2 points thick (see Figure 37).

```
BT
    /F13  48  Tf
    20  38  Td
    1  Tr
    2  w
    (ABC)  Tj
ET
```

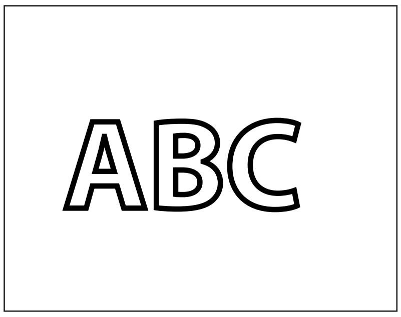


Figure 37 – Glyph outlines treated as a stroked path

EXAMPLE 3 This example illustrates how the glyphs' outlines may be used as a clipping boundary. The **Tr** operator sets the text rendering mode to 7 (clip), causing the subsequent **Tj** operator to impose the glyph outlines as the current clipping path. All subsequent painting operations mark the page only within this path, as illustrated in Figure 38. This state persists until an earlier clipping path is reinstated by the **Q** operator.

```
BT
    /F13  48  Tf
    20  38  Td
    7  Tr
    (ABC)  Tj
ET
…Graphics operators to draw a starburst…
```

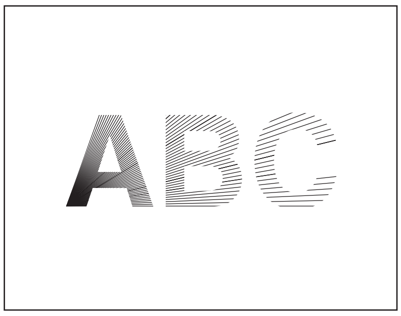


Figure 38 – Graphics clipped by a glyph path

9.2.4 Glyph Positioning and Metrics

A glyph's *width*—formally, its *horizontal displacement*—is the amount of space it occupies along the baseline of a line of text that is written horizontally. In other words, it is the distance the current text position shall move (by translating text space) when the glyph is painted.

NOTE 1 The width is distinct from the dimensions of the glyph outline.

In some fonts, the width is constant; it does not vary from glyph to glyph. Such fonts are called *fixed-pitch* or *monospaced*. They are used mainly for typewriter-style printing. However, most fonts used for high-quality typography associate a different width with each glyph. Such fonts are called *proportional* or *variable-pitch* fonts. In either case, the **Tj** operator shall position the consecutive glyphs of a string according to their widths.

The width information for each glyph shall be stored both in the font dictionary and in the font program itself. The two sets of widths shall be identical.

NOTE 2 Storing this information in the font dictionary, although redundant, enables a conforming reader to determine glyph positioning without having to look inside the font program.

NOTE 3 The operators for showing text are designed on the assumption that glyphs are ordinarily positioned according to their standard widths. However, means are provided to vary the positioning in certain limited ways. For example, the TJ operator enables the text position to be adjusted between any consecutive pair of glyphs corresponding to characters in a text string. There are graphics state parameters to adjust character and word spacing systematically.

In addition to width, a glyph has several other metrics that influence glyph positioning and painting. For most font types, this information is largely internal to the font program and is not specified explicitly in the PDF font dictionary. However, in a Type 3 font, all metrics are specified explicitly (see 9.6.5, "Type 3 Fonts").

The *glyph coordinate system* is the space in which an individual character's glyph is defined. All path coordinates and metrics shall be interpreted in glyph space. For all font types except Type 3, the units of glyph space are one-thousandth of a unit of text space; for a Type 3 font, the transformation from glyph space to text space shall be defined by a *font matrix* specified in an explicit **FontMatrix** entry in the font. Figure 39 shows a typical glyph outline and its metrics.

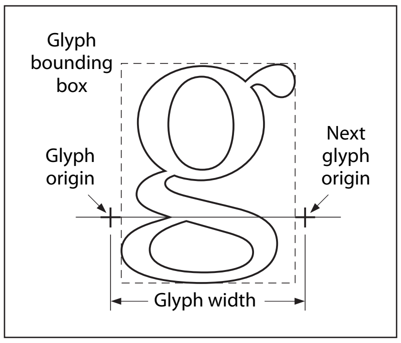


Figure 39 – Glyph metrics

The *glyph origin* is the point (0, 0) in the glyph coordinate system. **Tj** and other text-showing operators shall position the origin of the first glyph to be painted at the origin of text space.

EXAMPLE 1 This code adjusts the origin of text space to (40, 50) in the user coordinate system and then places the origin of the A glyph at that point:

```
BT
    40  50  Td
    (ABC)  Tj
ET
```

The *glyph displacement* is the distance from the glyph's origin to the point at which the origin of the *next* glyph should normally be placed when painting the consecutive glyphs of a line of text. This distance is a vector (called the *displacement vector*) in the glyph coordinate system; it has horizontal and vertical components.

NOTE 4 Most Western writing systems, including those based on the Latin alphabet, have a positive horizontal displacement and a zero vertical displacement. Some Asian writing systems have a nonzero vertical displacement. In all cases, the text-showing operators transform the displacement vector into text space and then translate text space by that amount.

The *glyph bounding box* shall be the smallest rectangle (oriented with the axes of the glyph coordinate system) that just encloses the entire glyph shape. The bounding box shall be expressed in terms of its left, bottom, right, and top coordinates relative to the glyph origin in the glyph coordinate system.

In some writing systems, text is frequently aligned in two different directions.

NOTE 5 It is common to write Japanese and Chinese glyphs either horizontally or vertically.

To handle this, a font may contain a second set of metrics for each glyph. Which set of metrics to use shall be selected according to a *writing mode*, where 0 shall specify horizontal writing and 1 shall specify vertical writing. This feature is available only for composite fonts, discussed in 9.7, "Composite Fonts".

When a glyph has two sets of metrics, each set shall specify a glyph origin and a displacement vector for that writing mode. In vertical writing, the glyph position shall be described by a *position vector* from the origin used for horizontal writing (origin 0) to the origin used for vertical writing (origin 1). Figure 40 illustrates the metrics for the two writing modes:

- The left diagram illustrates the glyph metrics associated with writing mode 0, horizontal writing. The coordinates *ll* and *ur* specify the bounding box of the glyph relative to origin 0. *w0* is the displacement vector that specifies how the text position shall be changed after the glyph is painted in writing mode 0; its vertical component shall be 0.

- The center diagram illustrates writing mode 1, vertical writing. *w1* shall be the displacement vector for writing mode 1; its horizontal component shall be 0.

- In the right diagram, *v* is a position vector defining the position of origin 1 relative to origin 0.

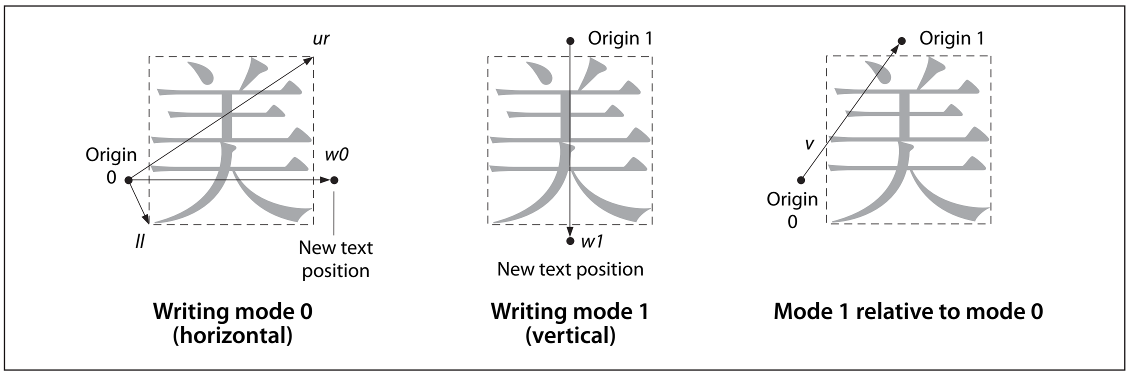

ur
Origin 0
w0
ll
New text position
Writing mode 0 (horizontal)
Origin 1
w1
New text position
Writing mode 1 (vertical)
Origin 1
v
Origin 0
Mode 1 relative to mode 0

Figure 40 – Metrics for horizontal and vertical writing modes

9.3 Text State Parameters and Operators

9.3.1 General

The *text state* comprises those graphics state parameters that only affect text. There are nine parameters in the text state (see Table 104).

Table 104 – Text state parameters

Parameter	Description
T_c	Character spacing
T_w	Word spacing
T_h	Horizontal scaling
T_l	Leading
T_f	Text font
T_{fs}	Text font size
T_{mode}	Text rendering mode
T_{rise}	Text rise
T_k	Text knockout

Except for the previously described T_f and T_{fs}, these parameters are discussed further in subsequent sub-clauses. (As described in 9.4, "Text Objects", three additional text-related parameters may occur only within a text object: T_m, the text matrix; T_{lm}, the text line matrix; and T_{rm}, the text rendering matrix.) The values of the text state parameters shall be consulted when text is positioned and shown (using the operators described in 9.4.2, "Text-Positioning Operators" and 9.4.3, "Text-Showing Operators"). In particular, the spacing and scaling parameters shall be used in a computation described in 9.4.4, "Text Space Details". The text state parameters may be set using the operators listed in Table 105.

The text knockout parameter, T_k, shall be set through the ***TK*** entry in a graphics state parameter dictionary by using the ***gs*** operator (see 8.4.5, "Graphics State Parameter Dictionaries"). There is no specific operator for setting this parameter.

The text state operators may appear outside text objects, and the values they set are retained across text objects in a single content stream. Like other graphics state parameters, these parameters shall be initialized to their default values at the beginning of each page.

Table 105 – Text state operators

Operands	Operator	Description
charSpace	**Tc**	Set the character spacing, T_c, to *charSpace*, which shall be a number expressed in unscaled text space units. Character spacing shall be used by the **Tj**, **TJ**, and **'** operators. Initial value: 0.
wordSpace	**Tw**	Set the word spacing, T_w, to *wordSpace*, which shall be a number expressed in unscaled text space units. Word spacing shall be used by the **Tj**, **TJ**, and **'** operators. Initial value: 0.
scale	**Tz**	Set the horizontal scaling, T_h, to (*scale* ÷ 100). *scale* shall be a number specifying the percentage of the normal width. Initial value: 100 (normal width).

Table 105 – Text state operators (continued)

Operands	Operator	Description
leading	**TL**	Set the text leading, T_l, to *leading*, which shall be a number expressed in unscaled text space units. Text leading shall be used only by the **T***, **'**, and **"** operators. Initial value: 0.
font size	**Tf**	Set the text font, T_f, to *font* and the text font size, T_{fs}, to *size*. *font* shall be the name of a font resource in the **Font** subdictionary of the current resource dictionary; *size* shall be a number representing a scale factor. There is no initial value for either *font* or *size*; they shall be specified explicitly by using **Tf** before any text is shown.
render	**Tr**	Set the text rendering mode, T_{mode}, to *render*, which shall be an integer. Initial value: 0.
rise	**Ts**	Set the text rise, T_{rise}, to *rise*, which shall be a number expressed in unscaled text space units. Initial value: 0.

Some of these parameters are expressed in *unscaled* text space units. This means that they shall be specified in a coordinate system that shall be defined by the text matrix, T_m but shall not be scaled by the font size parameter, T_{fs}.

9.3.2 Character Spacing

The character-spacing parameter, T_c, shall be a number specified in unscaled text space units (although it shall be subject to scaling by the T_h parameter if the writing mode is horizontal). When the glyph for each character in the string is rendered, T_c shall be *added* to the horizontal or vertical component of the glyph's displacement, depending on the writing mode. See 9.2.4, "Glyph Positioning and Metrics", for a discussion of glyph displacements. In the default coordinate system, horizontal coordinates increase from left to right and vertical coordinates from bottom to top. Therefore, for horizontal writing, a positive value of T_c has the effect of expanding the distance between glyphs (see Figure 41), whereas for vertical writing, a *negative* value of T_c has this effect.

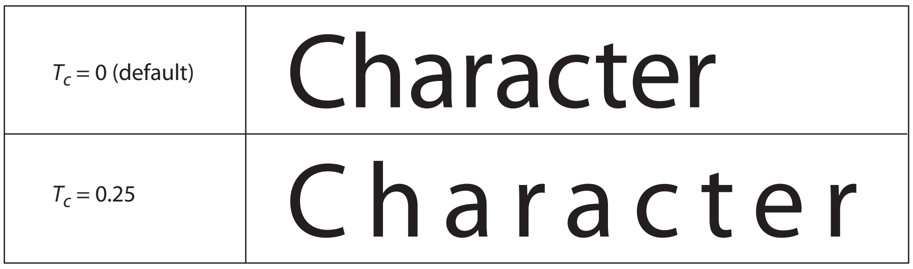


Figure 41 – Character spacing in horizontal writing

9.3.3 Word Spacing

Word spacing works the same way as character spacing but shall apply only to the ASCII SPACE character (20h). The word-spacing parameter, T_w, shall be added to the glyph's horizontal or vertical displacement (depending on the writing mode). For horizontal writing, a positive value for T_w has the effect of increasing the spacing between words. For vertical writing, a positive value for T_w *decreases* the spacing between words (and a negative value increases it), since vertical coordinates increase from bottom to top. Figure 42 illustrates the effect of word spacing in horizontal writing.

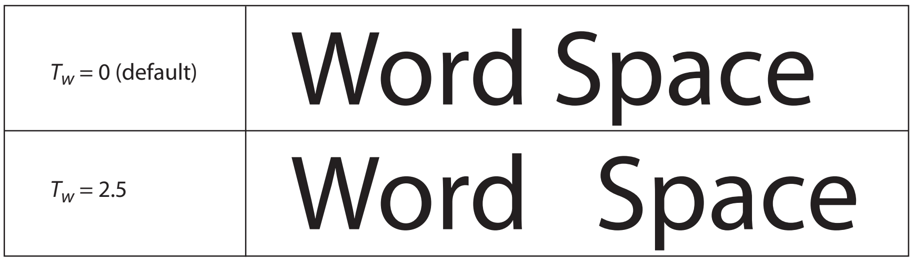


Figure 42 – Word spacing in horizontal writing

Word spacing shall be applied to every occurrence of the single-byte character code 32 in a string when using a simple font or a composite font that defines code 32 as a single-byte code. It shall not apply to occurrences of the byte value 32 in multiple-byte codes.

9.3.4 Horizontal Scaling

The horizontal scaling parameter, T_h, adjusts the width of glyphs by stretching or compressing them in the horizontal direction. Its value shall be specified as a percentage of the normal width of the glyphs, with 100 being the normal width. The scaling shall apply to the horizontal coordinate in text space, independently of the writing mode. It shall affect both the glyph's shape and its horizontal displacement (that is, its displacement vector). If the writing mode is horizontal, it shall also effect the spacing parameters T_c and T_w, as well as any positioning adjustments performed by the **TJ** operator. Figure 43 shows the effect of horizontal scaling.

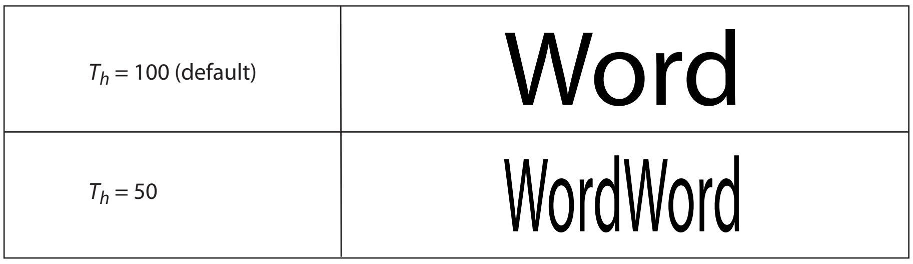


Figure 43 – Horizontal scaling

9.3.5 Leading

The leading parameter, T_l, shall be specified in unscaled text space units. It specifies the vertical distance between the baselines of adjacent lines of text, as shown in Figure 44.

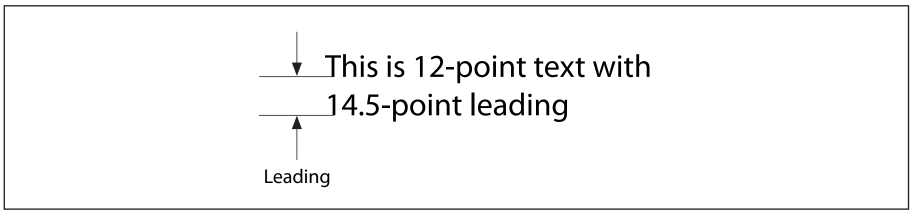


Figure 44 – Leading

The leading parameter shall be used by the **TD**, **T***, **'**, and **"** operators; see Table 108 for a precise description of its effects. This parameter shall apply to the vertical coordinate in text space, independently of the writing mode.

9.3.6 Text Rendering Mode

The text rendering mode, T_{mode}, determines whether showing text shall cause glyph outlines to be stroked, filled, used as a clipping boundary, or some combination of the three. Stroking, filling, and clipping shall have the same effects for a text object as they do for a path object (see 8.5.3, "Path-Painting Operators" and 8.5.4, "Clipping Path Operators"), although they are specified in an entirely different way. The graphics state parameters affecting those operations, such as line width, shall be interpreted in user space rather than in text space.

NOTE The text rendering modes are shown in Table 106. In the examples, a stroke colour of black and a fill colour of light gray are used. For the clipping modes (4 to 7), a series of lines has been drawn through the glyphs to show where the clipping occurs.

Only a value of 3 for text rendering mode shall have any effect on text displayed in a Type 3 font (see 9.6.5, "Type 3 Fonts").

If the text rendering mode calls for filling, the current nonstroking colour in the graphics state shall be used; if it calls for stroking, the current stroking colour shall be used. In modes that perform both filling and stroking, the effect shall be as if each glyph outline were filled and then stroked in separate operations. If any of the glyphs overlap, the result shall be equivalent to filling and stroking them one at a time, producing the appearance of stacked opaque glyphs, rather than first filling and then stroking them all at once. In the transparent imaging model, these combined filling and stroking modes shall be subject to further considerations; see 11.7.4.4, "Special Path-Painting Considerations".

The behaviour of the clipping modes requires further explanation. Glyph outlines shall begin accumulating if a **BT** operator is executed while the text rendering mode is set to a clipping mode or if it is set to a clipping mode within a text object. Glyphs shall accumulate until the text object is ended by an **ET** operator; the text rendering mode shall not be changed back to a nonclipping mode before that point.

Table 106 – Text rendering modes

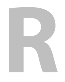
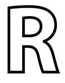
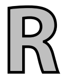
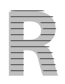
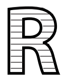
Mode	Example	Description
0		Fill text.
1		Stroke text.
2		Fill, then stroke text.
3		Neither fill nor stroke text (invisible).
4		Fill text and add to path for clipping (see 9.3.6, "Text Rendering Mode,").
5		Stroke text and add to path for clipping.

Table 106 – Text rendering modes (continued)

Mode	Example	Description
6		Fill, then stroke text and add to path for clipping.
7		Add text to path for clipping.

At the end of the text object, the accumulated glyph outlines, if any, shall be combined into a single path, treating the individual outlines as subpaths of that path and applying the nonzero winding number rule (see 8.5.3.3.2, "Nonzero Winding Number Rule"). The current clipping path in the graphics state shall be set to the intersection of this path with the previous clipping path. As is the case for path objects, this clipping shall occur *after* all filling and stroking operations for the text object have occurred. It remains in effect until a previous clipping path is restored by an invocation of the **Q** operator.

If no glyphs are shown or if the only glyphs shown have no outlines (for example, if they are ASCII SPACE characters (20h)), no clipping shall occur.

9.3.7 Text Rise

Text rise, T_{rise}, shall specify the distance, in unscaled text space units, to move the baseline up or down from its default location. Positive values of text rise shall move the baseline up. Figure 45 illustrates the effect of the text rise. Text rise shall apply to the vertical coordinate in text space, regardless of the writing mode.

NOTE Adjustments to the baseline are useful for drawing superscripts or subscripts. The default location of the baseline can be restored by setting the text rise to 0.

(This text is) Tj 5 Ts (superscripted) Tj	This text is $^{\text{superscripted}}$
(This text is) Tj –5 Ts (subscripted) Tj	This text is $_{\text{subscripted}}$
(This) Tj –5 Ts (text) Tj 5 Ts (moves) Tj 0 Ts (around) Tj	This $_{\text{text}}$ $^{\text{moves}}$ around

Figure 45 – Text rise

9.3.8 Text Knockout

The text knockout parameter, T_k *(PDF 1.4)*, shall be a boolean value that determines what text elements shall be considered elementary objects for purposes of colour compositing in the transparent imaging model. Unlike other text state parameters, there is no specific operator for setting this parameter; it may be set only through the **TK** entry in a graphics state parameter dictionary by using the **gs** operator (see 8.4.5, "Graphics State Parameter Dictionaries"). The text knockout parameter shall apply only to entire text objects; it shall not be set between the **BT** and **ET** operators delimiting a text object. Its initial value shall be **true**.

If the parameter is **false**, each glyph in a text object shall be treated as a separate elementary object; when glyphs overlap, they shall composite with one another.

If the parameter is **true**, all glyphs in the text object shall be treated together as a single elementary object; when glyphs overlap, later glyphs shall overwrite ("knock out") earlier ones in the area of overlap. This behaviour is equivalent to treating the entire text object as if it were a non-isolated knockout transparency group; see 11.4.6, "Knockout Groups". Transparency parameters shall be applied to the glyphs individually rather than to the implicit transparency group as a whole:

- Graphics state parameters, including transparency parameters, shall be inherited from the context in which the text object appears. They shall not be saved and restored. The transparency parameters shall not be reset at the beginning of the transparency group (as they are when a transparency group XObject is explicitly invoked). Changes made to graphics state parameters within the text object shall persist beyond the end of the text object.
- After the implicit transparency group for the text object has been completely evaluated, the group results shall be composited with the backdrop, using the **Normal** blend mode and alpha and soft mask values of 1.0.

9.4 Text Objects

9.4.1 General

A PDF *text object* consists of operators that may show text strings, move the text position, and set text state and certain other parameters. In addition, three parameters may be specified only within a text object and shall not persist from one text object to the next:

- T_m, the *text matrix*
- T_{lm}, the *text line matrix*
- T_{rm}, the *text rendering matrix*, which is actually just an intermediate result that combines the effects of text state parameters, the text matrix (T_m), and the current transformation matrix

A text object begins with the **BT** operator and ends with the **ET** operator, as shown in the Example, and described in Table 107.

EXAMPLE **BT**
…Zero or more text operators or other allowed operators…
ET

Table 107 – Text object operators

Operands	Operator	Description
—	**BT**	Begin a text object, initializing the text matrix, T_m, and the text line matrix, T_{lm}, to the identity matrix. Text objects shall not be nested; a second **BT** shall not appear before an **ET**.
—	**ET**	End a text object, discarding the text matrix.

These specific categories of text-related operators may appear in a text object:

- *Text state operators*, described in 9.3, "Text State Parameters and Operators"
- *Text-positioning operators*, described in 9.4.2, "Text-Positioning Operators"
- *Text-showing operators*, described in 9.4.3, "Text-Showing Operators"

The latter two sub-clauses also provide further details about these text object parameters. The other operators that may appear in a text object are those related to the general graphics state, colour, and marked content, as shown in Figure 9.

If a content stream does not contain any text, the ***Text*** procedure set may be omitted (see 14.2, "Procedure Sets"). In those circumstances, no text operators (including operators that merely set the text state) shall be present in the content stream, since those operators are defined in the same procedure set.

NOTE Although text objects cannot be statically nested, text might be shown using a Type 3 font whose glyph descriptions include any graphics objects, including another text object. Likewise, the current colour might be a tiling pattern whose pattern cell includes a text object.

9.4.2 Text-Positioning Operators

Text space is the coordinate system in which text is shown. It shall be defined by the text matrix, T_m, and the text state parameters T_{fs}, T_h, and T_{rise}, which together shall determine the transformation from text space to user space. Specifically, the origin of the first glyph shown by a text-showing operator shall be placed at the origin of text space. If text space has been translated, scaled, or rotated, then the position, size, or orientation of the glyph in user space shall be correspondingly altered.

The text-positioning operators shall only appear within text objects.

Table 108 – Text-positioning operators

Operands	Operator	Description
t_x t_y	**Td**	Move to the start of the next line, offset from the start of the current line by (t_x, t_y). t_x and t_y shall denote numbers expressed in unscaled text space units. More precisely, this operator shall perform these assignments: $$T_m = T_{lm} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \times T_{lm}$$
t_x t_y	**TD**	Move to the start of the next line, offset from the start of the current line by (t_x, t_y). As a side effect, this operator shall set the leading parameter in the text state. This operator shall have the same effect as this code: $-t_y$ TL t_x t_y Td

Table 108 – Text-positioning operators (continued)

Operands	Operator	Description
a b c d e f	**Tm**	Set the text matrix, T_m, and the text line matrix, T_{lm}: $$T_m = T_{lm} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$ The operands shall all be numbers, and the initial value for T_m and T_{lm} shall be the identity matrix, [1 0 0 1 0 0]. Although the operands specify a matrix, they shall be passed to **Tm** as six separate numbers, not as an array. The matrix specified by the operands shall not be concatenated onto the current text matrix, but shall replace it.
—	**T***	Move to the start of the next line. This operator has the same effect as the code 0 -T_l Td where T_l denotes the current leading parameter in the text state. The negative of T_l is used here because T_l is the text leading expressed as a positive number. Going to the next line entails decreasing the y coordinate.

At the beginning of a text object, T_m shall be the identity matrix; therefore, the origin of text space shall be initially the same as that of user space. The *text-positioning operators*, described in Table 108, alter T_m and thereby control the placement of glyphs that are subsequently painted. Also, the *text-showing operators*, described in Table 109, update T_m (by altering its *e* and *f* translation components) to take into account the horizontal or vertical displacement of each glyph painted as well as any character or word-spacing parameters in the text state.

Additionally, within a text object, a conforming reader shall keep track of a text line matrix, T_{lm}, which captures the value of T_m at the beginning of a line of text. The text-positioning and text-showing operators shall read and set T_{lm} on specific occasions mentioned in Tables 108 and 109.

NOTE This can be used to compactly represent evenly spaced lines of text.

9.4.3 Text-Showing Operators

The *text-showing operators* (Table 109) shall show text on the page, repositioning text space as they do so. All of the operators shall interpret the text string and apply the text state parameters as described in Table 109.

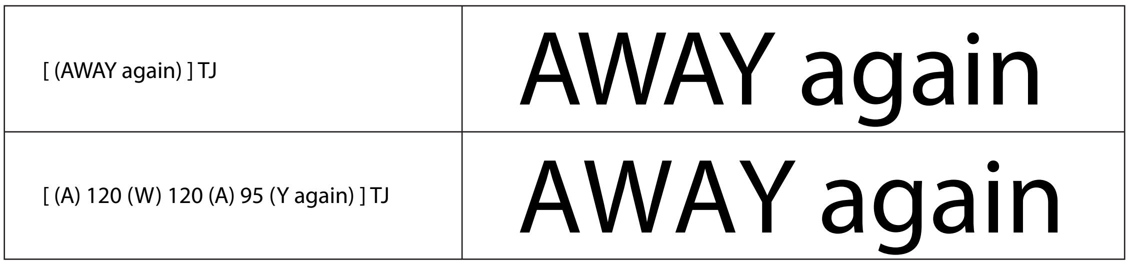
The text-showing operators shall only appear within text objects.

Table 109 – Text-showing operators

Operands	Operator	Description
string	**Tj**	Show a text string.
string	**'**	Move to the next line and show a text string. This operator shall have the same effect as the code T* *string* Tj

Table 109 – Text-showing operators (continued)

Operands	Operator	Description
a_w a_c *string*	**"**	Move to the next line and show a text string, using a_w as the word spacing and a_c as the character spacing (setting the corresponding parameters in the text state). a_w and a_c shall be numbers expressed in unscaled text space units. This operator shall have the same effect as this code: a_w Tw a_c Tc *string* '
array	**TJ**	Show one or more text strings, allowing individual glyph positioning. Each element of *array* shall be either a string or a number. If the element is a string, this operator shall show the string. If it is a number, the operator shall adjust the text position by that amount; that is, it shall translate the text matrix, T_m. The number shall be expressed in thousandths of a unit of text space (see 9.4.4, "Text Space Details"). This amount shall be *subtracted* from the current horizontal or vertical coordinate, depending on the writing mode. In the default coordinate system, a positive adjustment has the effect of moving the next glyph painted either to the left or down by the given amount. Figure 46 shows an example of the effect of passing offsets to **TJ**.



```
[ (AWAY again) ] TJ
[ (A) 120 (W) 120 (A) 95 (Y again) ] TJ
```

Figure 46 – Operation of the TJ operator in horizontal writing

A string operand of a text-showing operator shall be interpreted as a sequence of character codes identifying the glyphs to be painted.

With a simple font, each byte of the string shall be treated as a separate character code. The character code shall then be looked up in the font's encoding to select the glyph, as described in 9.6.6, "Character Encoding".

With a composite font *(PDF 1.2)*, multiple-byte codes may be used to select glyphs. In this instance, one or more consecutive bytes of the string shall be treated as a single character code. The code lengths and the mappings from codes to glyphs are defined in a data structure called a *CMap*, described in 9.7, "Composite Fonts".

The strings shall conform to the syntax for string objects. When a string is written by enclosing the data in parentheses, bytes whose values are equal to those of the ASCII characters LEFT PARENTHESIS (28h), RIGHT PARENTHESIS (29h), and REVERSE SOLIDUS (5Ch) (backslash) shall be preceded by a REVERSE SOLIDUS) character. All other byte values between 0 and 255 may be used in a string object. These rules apply to each individual byte in a string object, whether the string is interpreted by the text-showing operators as single-byte or multiple-byte character codes.

Strings presented to the text-showing operators may be of any length—even a single character code per string—and may be placed on the page in any order. The grouping of glyphs into strings has no significance for the display of text. Showing multiple glyphs with one invocation of a text-showing operator such as **Tj** shall produce the same results as showing them with a separate invocation for each glyph.

NOTE 6 The performance of text searching (and other text extraction operations) is significantly better if the text strings are as long as possible and are shown in natural reading order.

NOTE 7 In some cases, the text that is extracted can vary depending on the grouping of glyphs into strings. See, for example, 14.8.2.3.3, "Reverse-Order Show Strings".

9.4.4 Text Space Details

As stated in 9.4.2, "Text-Positioning Operators", text shall be shown in *text space*, defined by the combination of the text matrix, T_m, and the text state parameters T_{fs}, T_h, and T_{rise}. This determines how text coordinates are transformed into user space. Both the glyph's shape and its displacement (horizontal or vertical) shall be interpreted in text space.

NOTE 1 Glyphs are actually defined in glyph space, whose definition varies according to the font type as discussed in 9.2.4, "Glyph Positioning and Metrics". Glyph coordinates are first transformed from glyph space to text space before being subjected to the transformations described in Note 2.

NOTE 2 Conceptually, the entire transformation from text space to device space may be represented by a *text rendering matrix*, *T*rm:

$$T_{rm} = \begin{bmatrix} T_{fs} \times T_h & 0 & 0 \\ 0 & T_{fs} & 0 \\ 0 & T_{rise} & 1 \end{bmatrix} \times T_m \times CTM$$

*T*rm is a temporary matrix; conceptually, it is recomputed before each glyph is painted during a text-showing operation.

After the glyph is painted, the text matrix shall be updated according to the glyph displacement and any spacing parameters that apply. First, a combined displacement shall be computed, denoted by t_x in horizontal writing mode or t_y in vertical writing mode (the variable corresponding to the other writing mode shall be set to 0):

$$t_x = \left(\left(w0 - \frac{T_j}{1000} \right) \times T_{fs} + T_c + T_w \right) \times T_h$$

$$t_y = \left(w1 - \frac{T_j}{1000} \right) \times T_{fs} + T_c + T_w$$

where

w0 and *w1* denote the glyph's horizontal and vertical displacements

T_j denotes a number in a **TJ** array, if any, which specifies a position adjustment

T_{fs} and T_h denote the current text font size and horizontal scaling parameters in the graphics state

T_c and T_w denote the current character- and word-spacing parameters in the graphics state, if applicable

The text matrix shall then be then updated as follows:

$$T_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \times T_m$$

9.5 Introduction to Font Data Structures

A font shall be represented in PDF as a dictionary specifying the type of font, its PostScript name, its encoding, and information that can be used to provide a substitute when the font program is not available. Optionally, the font program may be embedded as a stream object in the PDF file.

The font types are distinguished by the **Subtype** entry in the font dictionary. Table 110 lists the font types defined in PDF. Type 0 fonts are called *composite fonts*; other types of fonts are called *simple fonts*. In addition to fonts, PDF supports two classes of font-related objects, called *CIDFonts* and *CMaps*, described in 9.7.2, "CID-Keyed Fonts Overview". CIDFonts are listed in Table 110 because, like fonts, they are collections of glyphs; however, a CIDFont shall not be used directly but only as a component of a Type 0 font.

Table 110 – Font types

Type	Subtype Value	Description
Type 0	**Type0**	*(PDF 1.2)* A *composite* font—a font composed of glyphs from a descendant CIDFont (see 9.7, "Composite Fonts")
Type 1	**Type1**	A font that defines glyph shapes using Type 1 font technology (see 9.6.2, "Type 1 Fonts").
	MMType1	A *multiple master* font—an extension of the Type 1 font that allows the generation of a wide variety of typeface styles from a single font (see 9.6.2.3, "Multiple Master Fonts")
Type 3	**Type3**	A font that defines glyphs with streams of PDF graphics operators (see 9.6.5, "Type 3 Fonts")
TrueType	**TrueType**	A font based on the TrueType font format (see 9.6.3, "TrueType Fonts")
CIDFont	**CIDFontType0**	*(PDF 1.2)* A CIDFont whose glyph descriptions are based on Type 1 font technology (see 9.7.4, "CIDFonts")
	CIDFontType2	*(PDF 1.2)* A CIDFont whose glyph descriptions are based on TrueType font technology (see 9.7.4, "CIDFonts")

For all font types, the term *font dictionary* refers to a PDF dictionary containing information about the font; likewise, a *CIDFont dictionary* contains information about a CIDFont. Except for Type 3, this dictionary is distinct from the *font program* that defines the font’s glyphs. That font program may be embedded in the PDF file as a stream object or be obtained from some external source.

NOTE 1 This terminology differs from that used in the PostScript language. In PostScript, a font dictionary is a PostScript data structure that is created as a direct result of interpreting a font program. In PDF, a font program is always treated as if it were a separate file, even when its content is embedded in the PDF file. The font program is interpreted by a specialized font interpreter when necessary; its contents never materialize as PDF objects.

NOTE 2 Most font programs (and related programs, such as CIDFonts and CMaps) conform to external specifications, such as the *Adobe Type 1 Font Format*. This standard does not include those specifications. See the Bibliography for more information about the specifications mentioned in this clause.

NOTE 3 The most predictable and dependable results are produced when all font programs used to show text are embedded in the PDF file. The following sub-clauses describe precisely how to do so. If a PDF file refers to font programs that are not embedded, the results depend on the availability of fonts in the conforming reader’s environment. The following sub-clauses specify some conventions for referring to external font programs. However, some details of font naming, font substitution, and glyph selection are implementation-dependent and may vary among different conforming readers, writers and operating system environments.

9.6 Simple Fonts

9.6.1 General

There are several types of simple fonts, all of which have these properties:

- Glyphs in the font shall be selected by single-byte character codes obtained from a string that is shown by the text-showing operators. Logically, these codes index into a table of 256 glyphs; the mapping from codes to glyphs is called the font’s *encoding*. Under some circumstances, the encoding may be altered by means described in 9.6.6, "Character Encoding".

- Each glyph shall have a single set of metrics, including a horizontal displacement or width, as described in 9.2.4, "Glyph Positioning and Metrics"; that is, simple fonts support only horizontal writing mode.

- Except for Type 0 fonts, Type 3 fonts in non-Tagged PDF documents, and certain standard Type 1 fonts, every font dictionary shall contain a subsidiary dictionary, the *font descriptor*, containing font-wide metrics and other attributes of the font; see 9.8, "Font Descriptors". Among those attributes is an optional *font file* stream containing the font program.

9.6.2 Type 1 Fonts

9.6.2.1 General

A Type 1 font program is a stylized PostScript program that describes glyph shapes. It uses a compact encoding for the glyph descriptions, and it includes hint information that enables high-quality rendering even at small sizes and low resolutions.

NOTE 1 Details on this format are provided in a separate specification, *Adobe Type 1 Font Format*. An alternative, more compact but functionally equivalent representation of a Type 1 font program is documented in Adobe Technical Note #5176, *The Compact Font Format Specification*.

NOTE 2 Although a Type 1 font program uses PostScript language syntax, using it does not require a full PostScript interpreter; a specialized Type 1 font interpreter suffices.

A Type 1 font dictionary may contain the entries listed in Table 111. Some entries are optional for the standard 14 fonts listed under 9.6.2.2, "Standard Type 1 Fonts (Standard 14 Fonts)", but are required otherwise.

Table 111 – Entries in a Type 1 font dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Font** for a font dictionary.
Subtype	name	*(Required)* The type of font; shall be **Type1** for a Type 1 font.
Name	name	*(Required in PDF 1.0; optional otherwise)* The name by which this font is referenced in the **Font** subdictionary of the current resource dictionary. This entry is obsolete and should not be used.
BaseFont	name	*(Required)* The PostScript name of the font. For Type 1 fonts, this is always the value of the **FontName** entry in the font program; for more information, see Section 5.2 of the *PostScript Language Reference*, Third Edition. The PostScript name of the font may be used to find the font program in the conforming reader or its environment. It is also the name that is used when printing to a PostScript output device.

Table 111 – Entries in a Type 1 font dictionary (continued)

Key	Type	Value
FirstChar	integer	*(Required except for the standard 14 fonts)* The first character code defined in the font’s **Widths** array. Beginning with PDF 1.5, the special treatment given to the standard 14 fonts is deprecated. Conforming writers should represent all fonts using a complete font descriptor. For backwards capability, conforming readers shall still provide the special treatment identified for the standard 14 fonts.
LastChar	integer	*(Required except for the standard 14 fonts)* The last character code defined in the font’s **Widths** array. Beginning with PDF 1.5, the special treatment given to the standard 14 fonts is deprecated. Conforming writers should represent all fonts using a complete font descriptor. For backwards capability, conforming readers shall still provide the special treatment identified for the standard 14 fonts.
Widths	array	*(Required except for the standard 14 fonts; indirect reference preferred)* An array of (**LastChar** − **FirstChar** + 1) widths, each element being the glyph width for the character code that equals **FirstChar** plus the array index. For character codes outside the range **FirstChar** to **LastChar**, the value of **MissingWidth** from the **FontDescriptor** entry for this font shall be used. The glyph widths shall be measured in units in which 1000 units correspond to 1 unit in text space. These widths shall be consistent with the actual widths given in the font program. For more information on glyph widths and other glyph metrics, see 9.2.4, "Glyph Positioning and Metrics". Beginning with PDF 1.5, the special treatment given to the standard 14 fonts is deprecated. Conforming writers should represent all fonts using a complete font descriptor. For backwards capability, conforming readers shall still provide the special treatment identified for the standard 14 fonts.
FontDescriptor	dictionary	*(Required except for the standard 14 fonts;* shall *be an indirect reference)* A font descriptor describing the font’s metrics other than its glyph widths (see 9.8, "Font Descriptors""\). For the standard 14 fonts, the entries ***FirstChar***, ***LastChar***, ***Widths***, and ***FontDescriptor*** shall either all be present or all be absent. Ordinarily, these dictionary keys may be absent; specifying them enables a standard font to be overridden; see 9.6.2.2, "Standard Type 1 Fonts (Standard 14 Fonts)". Beginning with PDF 1.5, the special treatment given to the standard 14 fonts is deprecated. Conforming writers should represent all fonts using a complete font descriptor. For backwards capability, conforming readers shall still provide the special treatment identified for the standard 14 fonts.
Encoding	name or dictionary	*(Optional)* A specification of the font’s character encoding if different from its built-in encoding. The value of **Encoding** shall be either the name of a predefined encoding (**MacRomanEncoding**, **MacExpertEncoding**, or **WinAnsiEncoding**, as described in Annex D) or an encoding dictionary that shall specify differences from the font’s built-in encoding or from a specified predefined encoding (see 9.6.6, "Character Encoding").
ToUnicode	stream	*(Optional; PDF 1.2)* A stream containing a CMap file that maps character codes to Unicode values (see 9.10, "Extraction of Text Content").

EXAMPLE This example shows the font dictionary for the Adobe Garamond® Semibold font. The font has an encoding dictionary (object 25), although neither the encoding dictionary nor the font descriptor (object 7) is shown in the example.

```
14 0 obj
    << /Type /Font
        /Subtype /Type1
        /BaseFont /AGaramond−Semibold
        /FirstChar 0
        /LastChar 255
        /Widths 21 0 R
        /FontDescriptor 7 0 R
        /Encoding 25 0 R
    >>
endobj


21 0 obj
    [ 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
      255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
      255 280 438 510 510 868 834 248 320 320 420 510 255 320 255 347
      510 510 510 510 510 510 510 510 510 510 255 255 510 510 510 330
      781 627 627 694 784 580 533 743 812 354 354 684 560 921 780 792
      588 792 656 504 682 744 650 968 648 590 638 320 329 320 510 500
      380 420 510 400 513 409 301 464 522 268 259 484 258 798 533 492
      516 503 349 346 321 520 434 684 439 448 390 320 255 320 510 255
      627 627 694 580 780 792 744 420 420 420 420 420 420 402 409 409
      409 409 268 268 268 268 533 492 492 492 492 492 520 520 520 520
      486 400 510 510 506 398 520 555 800 800 1044 360 380 549 846 792
      713 510 549 549 510 522 494 713 823 549 274 354 387 768 615 496
      330 280 510 549 510 549 612 421 421 1000 255 627 627 792 1016 730
      500 1000 438 438 248 248 510 494 448 590 100 510 256 256 539 539
      486 255 248 438 1174 627 580 627 580 580 354 354 354 354 792 792
      790 792 744 744 744 268 380 380 380 380 380 380 380 380 380 380
    ]
endobj
```

9.6.2.2 Standard Type 1 Fonts (Standard 14 Fonts)

The PostScript names of 14 Type 1 fonts, known as the *standard 14 fonts*, are as follows: Times-Roman, Helvetica, Courier, Symbol, Times-Bold, Helvetica-Bold, Courier-Bold, ZapfDingbats, Times-Italic, Helvetica-Oblique, Courier-Oblique, Times-BoldItalic, Helvetica-BoldOblique, Courier-BoldOblique

These fonts, or their font metrics and suitable substitution fonts, shall be available to the conforming reader.

NOTE The character sets and encodings for these fonts are listed in Annex D. The font metrics files for the standard 14 fonts are available from the ASN Web site (see the Bibliography). For more information on font metrics, see Adobe Technical Note #5004, *Adobe Font Metrics File Format Specification*.

9.6.2.3 Multiple Master Fonts

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 program. This is accomplished through the presence of various design dimensions in the font.

EXAMPLE 1 Examples of design dimensions are weight (light to extra-bold) and width (condensed to expanded).

Coordinates along these design dimensions (such as the degree of boldness) are specified by numbers. A particular choice of numbers selects an *instance* of the multiple master font. PDFs can contain multiple master instances.

NOTE Adobe Technical Note #5015, *Type 1 Font Format Supplement*, describes multiple master fonts in detail.

The font dictionary for a multiple master font instance may contain the same entries as a Type 1 font dictionary (see Table 111), with these differences:

- The value of **Subtype** shall be **MMType1**.
- If the PostScript name of the instance contains SPACEs (20h), the SPACEs shall be replaced by LOW LINEs (underscores) (5Fh) in the value of **BaseFont**. For instance, as illustrated in this example, the name “MinionMM 366 465 11 ” (which ends with a SPACE character) becomes /MinionMM_366_465_11_.

EXAMPLE 2

```
7  0  obj
   <<  /Type  /Font
       /Subtype  /MMType1
       /BaseFont  /MinionMM_366_465_11_
       /FirstChar  32
       /LastChar  255
       /Widths  19 0 R
       /FontDescriptor  6 0 R
       /Encoding  5 0 R
   >>
endobj

19  0  obj
   [  187  235  317  430  427  717  607  168  326  326  421  619  219  317  219  282  427
      …Omitted data…
      569  0  569  607  607  607  239  400  400  400  400  253  400  400  400  400  400
   ]
endobj
```

This example illustrates a convention for including the numeric values of the design coordinates as part of the instance’s **BaseFont** name. This convention is commonly used for accessing multiple master font instances from an external source in the conforming reader’s environment; it is documented in Adobe Technical Note #5088, *Font Naming Issues*. However, this convention is not prescribed as part of the PDF specification.

If the font program for a multiple master font instance is embedded in the PDF file, it shall be an ordinary Type 1 font program, not a multiple master font program. This font program is called a *snapshot* of the multiple master font instance that incorporates the chosen values of the design coordinates.

9.6.3 TrueType Fonts

A TrueType font dictionary may contain the same entries as a Type 1 font dictionary (see Table 111), with these differences:

- The value of **Subtype** shall be **TrueType**.
- The value of **Encoding** is subject to limitations that are described in 9.6.6, "Character Encoding".
- The value of **BaseFont** is derived differently.

The PostScript name for the value of **BaseFont** may be determined in one of two ways:

- If the TrueType font program's “name” table contains a PostScript name, it shall be used.
- In the absence of such an entry in the “name” table, a PostScript name shall be derived from the name by which the font is known in the host operating system. On a Windows system, the name shall be based on the lfFaceName field in a LOGFONT structure; in the Mac OS, it shall be based on the name of the FOND resource. If the name contains any SPACEs, the SPACEs shall be removed.

NOTE 1 The *TrueType* font format was developed by Apple Computer, Inc., and has been adopted as a standard font format for the Microsoft Windows operating system. Specifications for the TrueType font file format are available in Apple's *TrueType Reference Manual* and Microsoft's *TrueType 1.0 Font Files Technical Specification* (see Bibliography).

NOTE 2 A TrueType font program may be embedded directly in a PDF file as a stream object.

NOTE 3 The Type 42 font format that is defined for PostScript does not apply to PDF.

NOTE 4 For CJK (Chinese, Japanese, and Korean) fonts, the host font system's font name is often encoded in the host operating system's script. For instance, a Japanese font may have a name that is written in Japanese using some (unidentified) Japanese encoding. Thus, TrueType font names may contain multiple-byte character codes, each of which requires multiple characters to represent in a PDF name object (using the # notation to quote special characters as needed).

9.6.4 Font Subsets

PDF documents may include subsets of Type 1 and TrueType fonts. The font and font descriptor that describe a font subset are slightly different from those of ordinary fonts. These differences allow a conforming reader to recognize font subsets and to merge documents containing different subsets of the same font. (For more information on font descriptors, see 9.8, "Font Descriptors".)

For a font subset, the PostScript name of the font—the value of the font's **BaseFont** entry and the font descriptor's **FontName** entry— shall begin with a *tag* followed by a plus sign (+). The tag shall consist of exactly six uppercase letters; the choice of letters is arbitrary, but different subsets in the same PDF file shall have different tags.

EXAMPLE EOODIA+Poetica is the name of a subset of Poetica®, a Type 1 font.

9.6.5 Type 3 Fonts

Type 3 fonts differ from the other fonts supported by PDF. A Type 3 font dictionary defines the font; font dictionaries for other fonts simply contain information *about* the font and refer to a separate font program for the actual glyph descriptions. In Type 3 fonts, glyphs shall be defined by streams of PDF graphics operators. These streams shall be associated with glyph names. A separate encoding entry shall map character codes to the appropriate glyph names for the glyphs.

NOTE 1 Type 3 fonts are more flexible than Type 1 fonts because the glyph descriptions may contain arbitrary PDF graphics operators. However, Type 3 fonts have no hinting mechanism for improving output at small sizes or low resolutions.

A Type 3 font dictionary may contain the entries listed in Table 112.

Table 112 – Entries in a Type 3 font dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Font** for a font dictionary.
Subtype	name	*(Required)* The type of font; shall be **Type3** for a Type 3 font.
Name	name	*(Required in PDF 1.0; optional otherwise)* See Table 111.

Table 112 – Entries in a Type 3 font dictionary (continued)

Key	Type	Value
FontBBox	rectangle	*(Required)* A rectangle (see 7.9.5, "Rectangles") expressed in the glyph coordinate system, specifying the *font bounding box*. This is the smallest rectangle enclosing the shape that would result if all of the glyphs of the font were placed with their origins coincident and then filled. If all four elements of the rectangle are zero, a conforming reader shall make no assumptions about glyph sizes based on the font bounding box. If any element is nonzero, the font bounding box shall be accurate. If any glyph's marks fall outside this bounding box, incorrect behavior may result.
FontMatrix	array	*(Required)* An array of six numbers specifying the *font matrix*, mapping glyph space to text space (see 9.2.4, "Glyph Positioning and Metrics"). NOTE A common practice is to define glyphs in terms of a 1000-unit glyph coordinate system, in which case the font matrix is [0.001 0 0 0.001 0 0].
CharProcs	dictionary	*(Required)* A dictionary in which each key shall be a glyph name and the value associated with that key shall be a content stream that constructs and paints the glyph for that character. The stream shall include as its first operator either **d0** or **d1**, followed by operators describing one or more graphics objects, which may include path, text, or image objects. See below for more details about Type 3 glyph descriptions.
Encoding	name or dictionary	*(Required)* An encoding dictionary whose **Differences** array shall specify the complete character encoding for this font (see 9.6.6, "Character Encoding").
FirstChar	integer	*(Required)* The first character code defined in the font's **Widths** array.
LastChar	integer	*(Required)* The last character code defined in the font's **Widths** array.
Widths	array	*(Required; should be an indirect reference)* An array of (**LastChar** − **FirstChar** + 1) widths, each element being the glyph width for the character code that equals **FirstChar** plus the array index. For character codes outside the range **FirstChar** to **LastChar**, the width shall be 0. These widths shall be interpreted in glyph space as specified by **FontMatrix** (unlike the widths of a Type 1 font, which are in thousandths of a unit of text space). If ***FontMatrix*** specifies a rotation, only the *horizontal* component of the transformed width shall be used. That is, the resulting displacement shall be horizontal in text space, as is the case for all simple fonts.
FontDescriptor	dictionary	*(Required in Tagged PDF documents; shall be an indirect reference)* A font descriptor describing the font's default metrics other than its glyph widths (see 9.8, "Font Descriptors").
Resources	dictionary	*(Optional but should be used; PDF 1.2)* A list of the named resources, such as fonts and images, required by the glyph descriptions in this font (see 7.8.3, "Resource Dictionaries"). If any glyph descriptions refer to named resources but this dictionary is absent, the names shall be looked up in the resource dictionary of the page on which the font is used.
ToUnicode	stream	*(Optional; PDF 1.2)* A stream containing a CMap file that maps character codes to Unicode values (see 9.10, "Extraction of Text Content").

For each character code shown by a text-showing operator that uses a Type 3 font, the conforming reader shall:

a) Look up the character code in the font’s **Encoding** entry, as described in 9.6.6, "Character Encoding," to obtain a glyph name.

b) Look up the glyph name in the font’s **CharProcs** dictionary to obtain a stream object containing a glyph description. If the name is not present as a key in **CharProcs**, no glyph shall be painted.

c) Invoke the glyph description. The graphics state shall be saved before this invocation and shall be restored afterward; therefore, any changes the glyph description makes to the graphics state do not persist after it finishes.

When the glyph description begins execution, the current transformation matrix (CTM) shall be the concatenation of the font matrix (**FontMatrix** in the current font dictionary) and the text space that was in effect at the time the text-showing operator was invoked (see 9.4.4, "Text Space Details"). This means that shapes described in the glyph coordinate system are transformed into the user coordinate system and appear in the appropriate size and orientation on the page. The glyph description shall describe the glyph in terms of absolute coordinates in the glyph coordinate system, placing the glyph origin at (0, 0) in this space. It shall make no assumptions about the initial text position.

Aside from the CTM, the graphics state shall be inherited from the environment of the text-showing operator that caused the glyph description to be invoked. To ensure predictable results, the glyph description shall initialize any graphics state parameters on which it depends. In particular, if it invokes the **S** (stroke) operator, it shall explicitly set the line width, line join, line cap, and dash pattern to appropriate values.

NOTE 2 Normally, it is unnecessary and undesirable to initialize the current colour parameter because the text-showing operators are designed to paint glyphs with the current colour.

The glyph description shall execute one of the operators described in Table 113 to pass width and bounding box information to the font machinery. This shall precede the execution of any path construction or path-painting operators describing the glyph.

NOTE 3 Type 3 fonts in PDF are very similar to those in PostScript. Some of the information provided in Type 3 font dictionaries and glyph descriptions, while seemingly redundant or unnecessary, is nevertheless required for correct results when a conforming reader prints to a PostScript output device. This applies particularly to the operands of the ***d0*** and ***d1*** operators, are the equivalent of PostScript's ***setcharwidth*** and ***setcachedevice***. For further explanation, see Section 5.7 of the PostScript Language Reference, Third Edition.

Table 113 – Type 3 font operators

Operands	**Operator**	**Description**
w_x w_y	**d0**	Set width information for the glyph and declare that the glyph description specifies both its shape and its colour. NOTE This operator name ends in the digit **0**. w_x denotes the horizontal displacement in the glyph coordinate system; it shall be consistent with the corresponding width in the font’s **Widths** array. w_y shall be 0 (see 9.2.4, "Glyph Positioning and Metrics"). This operator shall only be permitted in a content stream appearing in a Type 3 font’s **CharProcs** dictionary. It is typically used only if the glyph description executes operators to set the colour explicitly.

Table 113 – Type 3 font operators (continued)

Operands	Operator	Description
w_x w_y ll_x ll_y ur_x ur_y	**d1**	Set width and bounding box information for the glyph and declare that the glyph description specifies only shape, not colour. NOTE This operator name ends in the digit **1**. w_x denotes the horizontal displacement in the glyph coordinate system; it shall be consistent with the corresponding width in the font's **Widths** array. w_y shall be 0 (see 9.2.4, "Glyph Positioning and Metrics"). ll_x and ll_y denote the coordinates of the lower-left corner, and ur_x and ur_y denote the upper-right corner, of the glyph bounding box. The glyph bounding box is the smallest rectangle, oriented with the axes of the glyph coordinate system, that completely encloses all marks placed on the page as a result of executing the glyph's description. The declared bounding box shall be correct—in other words, sufficiently large to enclose the entire glyph. If any marks fall outside this bounding box, the result is unpredictable. A glyph description that begins with the **d1** operator should not execute any operators that set the colour (or other colour-related parameters) in the graphics state; any use of such operators shall be ignored. The glyph description is executed solely to determine the glyph's shape. Its colour shall be determined by the graphics state in effect each time this glyph is painted by a text-showing operator. For the same reason, the glyph description shall not include an image; however, an image mask is acceptable, since it merely defines a region of the page to be painted with the current colour. This operator shall be used only in a content stream appearing in a Type 3 font's **CharProcs** dictionary.

EXAMPLE This example shows the definition of a Type 3 font with only two glyphs—a filled square and a filled triangle, selected by the character codes a and b. Figure 47 shows the result of showing the string (ababab) using this font.

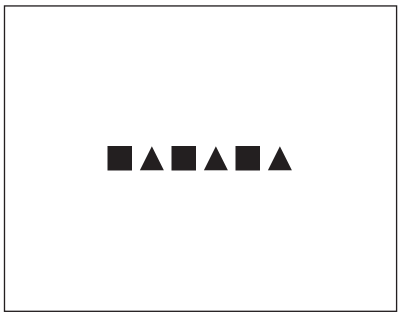


Figure 47 – Output from the example in 9.6.5, "Type 3 Fonts"

```
4  0  obj
    <<  /Type  /Font
         /Subtype  /Type3
         /FontBBox  [0  0  750  750]
         /FontMatrix  [0.001  0  0  0.001  0  0]
         /CharProcs  10 0 R
         /Encoding  9 0 R
         /FirstChar  97
         /LastChar  98
         /Widths  [1000  1000]
    >>
endobj

9  0  obj
    <<  /Type  /Encoding
         /Differences  [97  /square  /triangle]
    >>
endobj

10  0  obj
    <<  /square  11 0 R
         /triangle  12 0 R
    >>
endobj

11  0  obj
    <<  /Length  39  >>
stream
    1000  0  0  0  750  750  d1
    0  0  750  750  re
    f
endstream
endobj

12  0  obj
    <<  /Length  48  >>
stream
    1000  0  0  0  750  750  d1
    0  0  m
    375  750  l
    750  0  l
    f
endstream
endobj
```

9.6.6 Character Encoding

9.6.6.1 General

A font’s *encoding* is the association between character codes (obtained from text strings that are shown) and glyph descriptions. This sub-clause describes the character encoding scheme used with simple PDF fonts. Composite fonts (Type 0) use a different character mapping algorithm, as discussed in 9.7, "Composite Fonts".

Except for Type 3 fonts, every font program shall have a built-in encoding. Under certain circumstances, a PDF font dictionary may change the encoding used with the font program to match the requirements of the conforming writer generating the text being shown.

NOTE This flexibility in character encoding is valuable for two reasons:

It permits showing text that is encoded according to any of the various existing conventions. For example, the Microsoft Windows and Apple Mac OS operating systems use different standard encodings for Latin text, and many conforming writers use their own special-purpose encodings.

It permits conforming writers to specify how characters selected from a large character set are to be encoded.

Some character sets consist of more than 256 characters, including ligatures, accented characters, and other symbols required for high-quality typography or non-Latin writing systems. Different encodings may select different subsets of the same character set.

One commonly used font encoding for Latin-text font programs is often referred to as **StandardEncoding** or sometimes as the Adobe standard encoding. The name **StandardEncoding** shall have no special meaning in PDF, but this encoding does play a role as a default encoding (as shown in Table 114). The regular encodings used for Latin-text fonts on Mac OS and Windows systems shall be named **MacRomanEncoding** and **WinAnsiEncoding**, respectively. An encoding named **MacExpertEncoding** may be used with “expert” fonts that contain additional characters useful for sophisticated typography. Complete details of these encodings and of the characters present in typical fonts are provided in Annex D.

In PDF, a font is classified as either *nonsymbolic* or *symbolic* according to whether all of its characters are members of the standard Latin character set; see D.2, “Latin Character Set and Encodings”. This shall be indicated by flags in the font descriptor; see 9.8.2, "Font Descriptor Flags". Symbolic fonts contain other character sets, to which the encodings mentioned previously ordinarily do not apply. Such font programs have built-in encodings that are usually unique to each font. The standard 14 fonts include two symbolic fonts, Symbol and ZapfDingbats, whose encodings and character sets are documented in Annex D.

A font program’s built-in encoding may be overridden by including an **Encoding** entry in the PDF font dictionary. The possible encoding modifications depend on the font type. The value of the **Encoding** entry shall be either a named encoding (the name of one of the predefined encodings **MacRomanEncoding**, **MacExpertEncoding**, or **WinAnsiEncoding**) or an *encoding dictionary.* An encoding dictionary contains the entries listed in Table 114.

Table 114 – Entries in an encoding dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Encoding** for an encoding dictionary.
BaseEncoding	name	*(Optional)* The *base encoding*—that is, the encoding from which the **Differences** entry (if present) describes differences— shall be the name of one of the predefined encodings **MacRomanEncoding**, **MacExpertEncoding**, or **WinAnsiEncoding** (see Annex D). If this entry is absent, the **Differences** entry shall describe differences from an implicit base encoding. For a font program that is embedded in the PDF file, the implicit base encoding shall be the font program’s built-in encoding, as described in 9.6.6, "Character Encoding" and further elaborated in the sub-clauses on specific font types. Otherwise, for a nonsymbolic font, it shall be **StandardEncoding**, and for a symbolic font, it shall be the font’s built-in encoding.
Differences	array	*(Optional; should not be used with TrueType fonts)* An array describing the differences from the encoding specified by **BaseEncoding** or, if **BaseEncoding** is absent, from an implicit base encoding. The **Differences** array is described in subsequent sub-clauses.

The value of the **Differences** entry shall be an array of character codes and character names organized as follows:

$code_1$ $name_{1,1}$ $name_{1,2}$ …

code$_2$ *name*$_{2,1}$ *name*$_{2,2}$ …
…
code$_n$ *name*$_{n,1}$ *name*$_{n,2}$ …

Each code shall be the first index in a sequence of character codes to be changed. The first character name after the code becomes the name corresponding to that code. Subsequent names replace consecutive code indices until the next code appears in the array or the array ends. These sequences may be specified in any order but shall not overlap.

EXAMPLE In the encoding dictionary in this example, the name quotesingle (') is associated with character code 39, Adieresis (Ä) with code 128, Aring (Å) with 129, and trademark (™) with 170.

```
25 0 obj
   << /Type /Encoding
       /Differences
         [     39 /quotesingle
               96 /grave

              128 /Adieresis /Aring /Ccedilla /Eacute /Ntilde /Odieresis /Udieresis
                  /aacute /agrave /acircumflex /adieresis /atilde /aring /ccedilla
                  /eacute /egrave /ecircumflex /edieresis /iacute /igrave /icircumflex
                  /idieresis /ntilde /oacute /ograve /ocircumflex /odieresis /otilde
                  /uacute /ugrave /ucircumflex /udieresis /dagger /degree /cent
                  /sterling /section /bullet /paragraph /germandbls /registered
                  /copyright /trademark /acute /dieresis

              174 /AE /Oslash
              177 /plusminus
              180 /yen /mu
              187 /ordfeminine /ordmasculine
              190 /ae /oslash /questiondown /exclamdown /logicalnot
              196 /florin
              199 /guillemotleft /guillemotright /ellipsis
              203 /Agrave /Atilde /Otilde /OE /oe /endash /emdash /quotedblleft
                  /quotedblright /quoteleft /quoteright /divide

              216 /ydieresis /Ydieresis /fraction /currency /guilsinglleft /guilsinglright
                  /fi /fl /daggerdbl /periodcentered /quotesinglbase /quotedblbase
                  /perthousand /Acircumflex /Ecircumflex /Aacute /Edieresis /Egrave
                  /Iacute /Icircumflex /Idieresis /Igrave /Oacute /Ocircumflex

              241 /Ograve /Uacute /Ucircumflex /Ugrave /dotlessi /circumflex /tilde
                  /macron /breve /dotaccent /ring /cedilla /hungarumlaut /ogonek
                  /caron
         ]
   >>
endobj
```

9.6.6.2 Encodings for Type 1 Fonts

A Type 1 font program's glyph descriptions are keyed by glyph *names*, not by character *codes*. Glyph names are ordinary PDF name objects. Descriptions of Latin alphabetic characters are normally associated with names consisting of single letters, such as **A** or **a**. Other characters are associated with names composed of words, such as three, ampersand, or parenleft. A Type 1 font's built-in encoding shall be defined by an **Encoding** array that is part of the font program, not to be confused with the **Encoding** entry in the PDF font dictionary.

An **Encoding** entry may override a Type 1 font's mapping from character codes to character names. The **Differences** array may map a code to the name of any glyph description that exists in the font program, regardless of whether that glyph is referenced by the font's built-in encoding or by the encoding specified in the **BaseEncoding** entry.

All Type 1 font programs shall contain an actual glyph named .notdef. The effect produced by showing the .notdef glyph is at the discretion of the font designer. If an encoding maps to a character name that does not exist in the Type 1 font program, the .notdef glyph shall be substituted.

9.6.6.3 Encodings for Type 3 Fonts

A Type 3 font, like a Type 1 font, contains glyph descriptions that are keyed by glyph names; in this case, they appear as explicit keys in the font's **CharProcs** dictionary. A Type 3 font's mapping from character codes to glyph names shall be entirely defined by its **Encoding** entry, which is required in this case.

9.6.6.4 Encodings for TrueType Fonts

A TrueType font program's built-in encoding maps directly from character codes to glyph descriptions by means of an internal data structure called a "cmap" (not to be confused with the CMap described in 9.7.5, "CMaps"). This sub-clause describes how the PDF font dictionary's **Encoding** entry shall be used in conjunction with a "cmap" to map from a character code in a string to a glyph description in a TrueType font program.

A "cmap" table may contain one or more subtables that represent multiple encodings intended for use on different platforms (such as Mac OS and Windows). Each subtable shall be identified by the two numbers, such as (3, 1), that represent a combination of a *platform ID* and a *platform-specific encoding ID*, respectively.

Glyph names are not required in TrueType fonts, although some font programs have an optional "post" table listing glyph names for the glyphs. If the conforming reader needs to select glyph descriptions by name, it translates from glyph names to codes in one of the encodings given in the font program's "cmap" table. When there is no character code in the "cmap" that corresponds to a glyph name, the "post" table shall be used to select a glyph description directly from the glyph name.

Because some aspects of TrueType glyph selection are dependent on the conforming reader or the operating system, PDF files that use TrueType fonts should follow certain guidelines to ensure predictable behaviour across all conforming readers:

- The font program should be embedded.
- A nonsymbolic font should specify **MacRomanEncoding** or **WinAnsiEncoding** as the value of its **Encoding** entry, with no **Differences** array.
- A font that is used to display glyphs that do not use **MacRomanEncoding** or **WinAnsiEncoding** should not specify an **Encoding** entry. The font descriptor's Symbolic flag (see Table 123) should be set, and its font program's "cmap" table should contain a (1, 0) subtable. It may also contain a (3, 0) subtable; if present, this subtable should map from character codes in the range 0xF000 to 0xF0FF by prepending the single-byte codes in the (1, 0) subtable with 0xF0 and mapping to the corresponding glyph descriptions.

NOTE 1 Some popular TrueType font programs contain incorrect encoding information. Implementations of TrueType font interpreters have evolved heuristics for dealing with such problems; those heuristics are not described here. For maximum portability, only well-formed TrueType font programs should be used in PDF files. Therefore, a TrueType font program in a PDF file may need to be modified to conform to these guidelines.

The following paragraphs describe the treatment of TrueType font encodings beginning with PDF 1.3.

If the font has a named **Encoding** entry of either **MacRomanEncoding** or **WinAnsiEncoding**, or if the font descriptor's Nonsymbolic flag (see Table 123) is set, the conforming reader shall create a table that maps from character codes to glyph names:

- If the **Encoding** entry is one of the names **MacRomanEncoding** or **WinAnsiEncoding**, the table shall be initialized with the mappings described in Annex D.

- If the **Encoding** entry is a dictionary, the table shall be initialized with the entries from the dictionary's **BaseEncoding** entry (see Table 114). Any entries in the **Differences** array shall be used to update the table. Finally, any undefined entries in the table shall be filled using **StandardEncoding**.

If a (3, 1) "cmap" subtable (Microsoft Unicode) is present:

- A character code shall be first mapped to a glyph name using the table described above.
- The glyph name shall then be mapped to a Unicode value by consulting the *Adobe Glyph List* (see the Bibliography).
- Finally, the Unicode value shall be mapped to a glyph description according to the (3, 1) subtable.

If no (3, 1) subtable is present but a (1, 0) subtable (Macintosh Roman) is present:

- A character code shall be first mapped to a glyph name using the table described above.
- The glyph name shall then be mapped back to a character code according to the standard Roman encoding used on Mac OS.
- Finally, the code shall be mapped to a glyph description according to the (1, 0) subtable.

In any of these cases, if the glyph name cannot be mapped as specified, the glyph name shall be looked up in the font program's "post" table (if one is present) and the associated glyph description shall be used.

The standard Roman encoding that is used on Mac OS is the same as the ***MacRomanEncoding*** described in Annex D, with the addition of 15 entries and the replacement of the currency glyph with the Euro glyph, as shown in Table 115.

Table 115 – Differences between MacRomanEncoding and Mac OS Roman encoding

Name	**Code (Octal)**	**Code (DEcimal)**
notequal	255	173
infinity	260	176
lessequal	262	178
greaterequal	263	179
partialdiff	266	182
summation	267	183
product	270	184
pi	271	185
integral	272	186
Omega	275	189
radical	303	195
approxequal	305	197
Delta	306	198
lozenge	327	215
Euro	333	219
apple	360	240

When the font has no **Encoding** entry, or the font descriptor's Symbolic flag is set (in which case the **Encoding** entry is ignored), this shall occur:

- If the font contains a (3, 0) subtable, the range of character codes shall be one of these: 0x0000 - 0x00FF, 0xF000 - 0xF0FF, 0xF100 - 0xF1FF, or 0xF200 - 0xF2FF. Depending on the range of codes, each byte from the string shall be prepended with the high byte of the range, to form a two-byte character, which shall be used to select the associated glyph description from the subtable.

- Otherwise, if the font contains a (1, 0) subtable, single bytes from the string shall be used to look up the associated glyph descriptions from the subtable.

If a character cannot be mapped in any of the ways described previously, a conforming reader may supply a mapping of its choosing.

9.7 Composite Fonts

9.7.1 General

A *composite font,* also called a Type 0 font, is one whose glyphs are obtained from a fontlike object called a *CIDFont*. A composite font shall be represented by a font dictionary whose **Subtype** value is **Type0**. The Type 0 font is known as the *root font*, and its associated CIDFont is called its *descendant.*

NOTE 1 Composite fonts in PDF are analogous to composite fonts in PostScript but with some limitations. In particular, PDF requires that the character encoding be defined by a CMap, which is only one of several encoding methods available in PostScript. Also, PostScript allows a Type 0 font to have multiple descendants, which might also be Type 0 fonts. PDF supports only a single descendant, which shall be a CIDFont.

When the current font is composite, the text-showing operators shall behave differently than with simple fonts. For simple fonts, each byte of a string to be shown selects one glyph, whereas for composite fonts, a sequence of one or more bytes are decoded to select a glyph from the descendant CIDFont.

NOTE 2 This facility supports the use of very large character sets, such as those for the Chinese, Japanese, and Korean languages. It also simplifies the organization of fonts that have complex encoding requirements.

This sub-clause first introduces the architecture of *CID-keyed fonts*, which are the only kind of composite font supported in PDF. Then it describes the *CIDFont* and *CMap* dictionaries, which are the PDF objects that represent the correspondingly named components of a CID-keyed font. Finally, it describes the Type 0 font dictionary, which combines a CIDFont and a CMap to produce a font whose glyphs may be accessed by means of variable-length character codes in a string to be shown.

9.7.2 CID-Keyed Fonts Overview

CID-keyed fonts provide a convenient and efficient method for defining multiple-byte character encodings and fonts with a large number of glyphs. These capabilities provide great flexibility for representing text in writing systems for languages with large character sets, such as Chinese, Japanese, and Korean (CJK).

The *CID-keyed font* architecture specifies the external representation of certain font programs, called *CMap* and *CIDFont* files, along with some conventions for combining and using those files. As mentioned earlier, PDF does not support the entire CID-keyed font architecture, which is independent of PDF; CID-keyed fonts may be used in other environments.

NOTE For complete documentation on the architecture and the file formats, see Adobe Technical Notes #5092, *CID-Keyed Font Technology Overview*, and #5014, *Adobe CMap and CIDFont Files Specification*. This sub-clause describes only the PDF objects that represent these font programs.

The term *CID-keyed font* reflects the fact that *CID* (character identifier) numbers are used to index and access the glyph descriptions in the font. This method is more efficient for large fonts than the method of accessing by character name, as is used for some simple fonts. CIDs range from 0 to a maximum value that may be subject to an implementation limit (see Table C.1).

A *character collection* is an ordered set of glyphs. The order of the glyphs in the character collection shall determine the CID number for each glyph. Each CID-keyed font shall explicitly reference the character collection on which its CID numbers are based; see 9.7.3, "CIDSystemInfo Dictionaries".

A *CMap* (character map) file shall specify the correspondence between character codes and the CID numbers used to identify glyphs. It is equivalent to the concept of an encoding in simple fonts. Whereas a simple font allows a maximum of 256 glyphs to be encoded and accessible at one time, a CMap can describe a mapping from multiple-byte codes to thousands of glyphs in a large CID-keyed font.

EXAMPLE A CMap can describe Shift-JIS, one of several widely used encodings for Japanese.

A CMap file may reference an entire character collection or a subset of a character collection. The CMap file's mapping yields a *font number* (which in PDF shall be 0) and a *character selector* (which in PDF shall be a CID). Furthermore, a CMap file may incorporate another CMap file by reference, without having to duplicate it. These features enable character collections to be combined or supplemented and make all the constituent characters accessible to text-showing operations through a single encoding.

A *CIDFont* contains the glyph descriptions for a character collection. The glyph descriptions themselves are typically in a format similar to those used in simple fonts, such as Type 1. However, they are identified by CIDs rather than by names, and they are organized differently.

In PDF, the data from a CMap file and CIDFont shall be represented by PDF objects as described in 9.7.4, "CIDFonts" and 9.7.5, "CMaps". The CMap file and CIDFont programs themselves may be either referenced by name or embedded as stream objects in the PDF file.

A CID-keyed font, then, shall be the combination of a CMap with a CIDFont containing glyph descriptions. It shall be represented as a Type 0 font. It contains an **Encoding** entry whose value shall be a CMap dictionary, and its **DescendantFonts** entry shall reference the CIDFont dictionary with which the CMap has been combined.

9.7.3 CIDSystemInfo Dictionaries

CIDFont and CMap dictionaries shall contain a **CIDSystemInfo** entry specifying the character collection assumed by the CIDFont associated with the CMap—that is, the interpretation of the CID numbers used by the CIDFont. A character collection shall be uniquely identified by the **Registry**, **Ordering**, and **Supplement** entries in the **CIDSystemInfo** dictionary, as described in Table 116. In order to be compatible, the Registry and Ordering values must be the same.

The **CIDSystemInfo** entry in a CIDFont is a dictionary that shall specify the CIDFont's character collection. The CIDFont need not contain glyph descriptions for all the CIDs in a collection; it may contain a subset. The **CIDSystemInfo** entry in a CMap file shall be either a single dictionary or an array of dictionaries, depending on whether it associates codes with a single character collection or with multiple character collections; see 9.7.5, "CMaps".

For proper behaviour, the **CIDSystemInfo** entry of a CMap shall be compatible with that of the CIDFont or CIDFonts with which it is used.

Table 116 – Entries in a CIDSystemInfo dictionary

Key	Type	Value
Registry	ASCII string	*(Required)* A string identifying the issuer of the character collection. For information about assigning a registry identifier, contact the Adobe Solutions Network or consult the ASN Web site (see the Bibliography).

Table 116 – Entries in a CIDSystemInfo dictionary (continued)

Key	Type	Value
Ordering	ASCII string	*(Required)* A string that uniquely names the character collection within the specified registry.
Supplement	integer	*(Required)* The *supplement number* of the character collection. An original character collection has a supplement number of 0. Whenever additional CIDs are assigned in a character collection, the supplement number shall be increased. Supplements shall not alter the ordering of existing CIDs in the character collection. This value shall not be used in determining compatibility between character collections.

9.7.4 CIDFonts

9.7.4.1 General

A CIDFont program contains glyph descriptions that are accessed using a CID as the character selector. There are two types of CIDFonts:

- A Type 0 CIDFont contains glyph descriptions based on CFF

NOTE The term “Type 0” when applied to a CIDFont has a different meaning than for a “Type 0 font”.

- A Type 2 CIDFont contains glyph descriptions based on the TrueType font format

A CIDFont dictionary is a PDF object that contains information about a CIDFont program. Although its **Type** value is **Font**, a CIDFont is not actually a font. It does not have an **Encoding** entry, it may not be listed in the **Font** subdictionary of a resource dictionary, and it may not be used as the operand of the **Tf** operator. It shall be used only as a descendant of a Type 0 font. The CMap in the Type 0 font shall be what defines the encoding that maps character codes to CIDs in the CIDFont. Table 117 lists the entries in a CIDFont dictionary.

Table 117 – Entries in a CIDFont dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Font** for a CIDFont dictionary.
Subtype	name	*(Required)* The type of CIDFont shall be **CIDFontType0** or **CIDFontType2**.
BaseFont	name	*(Required)* The PostScript name of the CIDFont. For Type 0 CIDFonts, this shall be the value of the **CIDFontName** entry in the CIDFont program. For Type 2 CIDFonts, it shall be derived the same way as for a simple TrueType font; see 9.6.3, "TrueType Fonts". In either case, the name may have a subset prefix if appropriate; see 9.6.4, "Font Subsets".
CIDSystemInfo	dictionary	*(Required)* A dictionary containing entries that define the character collection of the CIDFont. See Table 116.
FontDescriptor	dictionary	*(Required; shall be an indirect reference)* A font descriptor describing the CIDFont’s default metrics other than its glyph widths (see 9.8, "Font Descriptors").
DW	integer	*(Optional)* The default width for glyphs in the CIDFont (see 9.7.4.3, "Glyph Metrics in CIDFonts"). Default value: 1000 (defined in user units).

Table 117 – Entries in a CIDFont dictionary (continued)

Key	Type	Value
W	array	*(Optional)* A description of the widths for the glyphs in the CIDFont. NOTE The array's elements have a variable format that can specify individual widths for consecutive CIDs or one width for a range of CIDs (see 9.7.4.3, "Glyph Metrics in CIDFonts"). Default value: none (the **DW** value shall be used for all glyphs).
DW2	array	*(Optional; applies only to CIDFonts used for vertical writing)* An array of two numbers specifying the default metrics for vertical writing (see 9.7.4.3, "Glyph Metrics in CIDFonts"). Default value: [880 −1000].
W2	array	*(Optional; applies only to CIDFonts used for vertical writing)* A description of the metrics for vertical writing for the glyphs in the CIDFont (see 9.7.4.3, "Glyph Metrics in CIDFonts"). Default value: none (the **DW2** value shall be used for all glyphs).
CIDToGIDMap	stream or name	*(Optional; Type 2 CIDFonts only)* A specification of the mapping from CIDs to glyph indices. If the value is a stream, the bytes in the stream shall contain the mapping from CIDs to glyph indices: the glyph index for a particular CID value *c* shall be a 2-byte value stored in bytes $2 \times c$ and $2 \times c + 1$, where the first byte shall be the high-order byte. If the value of **CIDToGIDMap** is a name, it shall be **Identity**, indicating that the mapping between CIDs and glyph indices is the identity mapping. Default value: **Identity**. This entry may appear only in a Type 2 CIDFont whose associated TrueType font program is embedded in the PDF file.

9.7.4.2 Glyph Selection in CIDFonts

Type 0 and Type 2 CIDFonts handle the mapping from CIDs to glyph descriptions in somewhat different ways.

For Type 0, the CIDFont program contains glyph descriptions that are identified by CIDs. The CIDFont program identifies the character collection by a **CIDSystemInfo** dictionary, which should be copied into the PDF CIDFont dictionary. CIDs shall be interpreted uniformly in all CIDFont programs supporting a given character collection, whether the program is embedded in the PDF file or obtained from an external source.

When the CIDFont contains an embedded font program that is represented in the Compact Font Format (CFF), the **FontFile3** entry in the font descriptor (see Table 126) may be **CIDFontType0C** or **OpenType**. There are two cases, depending on the contents of the font program:

- The "CFF" font program has a Top DICT that uses CIDFont operators: The CIDs shall be used to determine the GID value for the glyph procedure using the charset table in the CFF program. The GID value shall then be used to look up the glyph procedure using the CharStrings INDEX table.

NOTE Although in many fonts the CID value and GID value are the same, the CID and GID values may differ.

- The "CFF" font program has a Top DICT that does not use CIDFont operators: The CIDs shall be used directly as GID values, and the glyph procedure shall be retrieved using the CharStrings INDEX.

For Type 2, the CIDFont program is actually a TrueType font program, which has no native notion of CIDs. In a TrueType font program, glyph descriptions are identified by *glyph index* values. Glyph indices are internal to the font and are not defined consistently from one font to another. Instead, a TrueType font program contains a "cmap" table that provides mappings directly from character codes to glyph indices for one or more predefined encodings.

TrueType font programs are integrated with the CID-keyed font architecture in one of two ways, depending on whether the font program is embedded in the PDF file:

- If the TrueType font program is embedded, the Type 2 CIDFont dictionary shall contain a **CIDToGIDMap** entry that maps CIDs to the glyph indices for the appropriate glyph descriptions in that font program.
- If the TrueType font program is not embedded but is referenced by name, the Type 2 CIDFont dictionary shall *not* contain a **CIDToGIDMap** entry, since it is not meaningful to refer to glyph indices in an external font program. In this case, CIDs shall not participate in glyph selection, and only predefined CMaps may be used with this CIDFont (see 9.7.5, "CMaps"). The conforming reader shall select glyphs by translating characters from the encoding specified by the predefined CMap to one of the encodings in the TrueType font's “cmap” table. The means by which this is accomplished are implementation-dependent.

Even though the CIDs are not used to select glyphs in a Type 2 CIDFont, they shall always be used to determine the glyph metrics, as described in the next sub-clause.

Every CIDFont shall contain a glyph description for CID 0, which is analogous to the .notdef character name in simple fonts (see 9.7.6.3, "Handling Undefined Characters").

9.7.4.3 Glyph Metrics in CIDFonts

As discussed in 9.2.4, "Glyph Positioning and Metrics", the *width* of a glyph refers to the horizontal displacement between the origin of the glyph and the origin of the next glyph when writing in horizontal mode. In this mode, the vertical displacement between origins shall be 0. Widths for a CIDFont are defined using the **DW** and **W** entries in the CIDFont dictionary. These widths shall be consistent with the actual widths given in the CIDFont program.

The **W** array allows the definition of widths for individual CIDs. The elements of the array shall be organized in groups of two or three, where each group shall be in one of these two formats:

$c\ [w_1\ w_2\ \dots\ w_n]$

$c_{\text{first}}\ c_{\text{last}}\ w$

In the first format, *c* shall be an integer specifying a starting CID value; it shall be followed by an array of *n* numbers that shall specify the widths for *n* consecutive CIDs, starting with *c*. The second format shall define the same width, *w*, for all CIDs in the range *c*first to *c*last.

EXAMPLE 1 In this example, the glyphs having CIDs 120, 121, and 122 are 400, 325, and 500 units wide, respectively. CIDs in the range 7080 through 8032 all have a width of 1000 units.

W entry example:

```
/W [ 120 [400 325 500]
       7080 8032 1000
    ]
```

Glyphs from a CIDFont may be shown in vertical writing mode. This is selected by the **WMode** entry in the associated CMap dictionary; see 9.7.5, "CMaps". To be used in this way, the CIDFont shall define the vertical displacement for each glyph and the position vector that relates the horizontal and vertical writing origins.

The default position vector and vertical displacement vector shall be specified by the **DW2** entry in the CIDFont dictionary. **DW2** shall be an array of two values: the vertical component of the position vector *v* and the vertical component of the displacement vector *w1* (see Figure 40). The horizontal component of the position vector shall be half the glyph width, and that of the displacement vector shall be 0.

EXAMPLE 2 If the **DW2** entry is

```
/DW2 [880 −1000]
```

then a glyph's position vector and vertical displacement vector are

$$\begin{aligned} \mathrm{v} &= (w0 \div 2, 880) \\ w1 &= (0, -1000) \end{aligned}$$

where *w0* is the width (horizontal displacement) for the same glyph.

NOTE A negative value for the vertical component places the origin of the next glyph *below* the current glyph because vertical coordinates in a standard coordinate system increase from bottom to top.

The **W2** array shall define vertical metrics for individual CIDs. The elements of the array shall be organized in groups of two or five, where each group shall be in one of these two formats:

$c\ [w1_{1y}\ v_{1x}\ v_{1y}\ \ w1_{2y}\ v_{2x}\ v_{2y}\ \ \ldots]$

$c_{\text{first}}\ c_{\text{last}}\ \ w1_{1y}\ v_{1x}\ v_{1y}$

In the first format, *c* is a starting CID and shall be followed by an array containing numbers interpreted in groups of three. Each group shall consist of the vertical component of the vertical displacement vector *w*1 (whose horizontal component shall be 0) followed by the horizontal and vertical components for the position vector *v*. Successive groups shall define the vertical metrics for consecutive CIDs starting with *c*. The second format defines a range of CIDs from *c*first to *c*last, that shall be followed by three numbers that define the vertical metrics for all CIDs in this range.

EXAMPLE 3 This **W2** entry defines the vertical displacement vector for the glyph with CID 120 as (0, −1000) and the position vector as (250, 772). It also defines the displacement vector for CIDs in the range 7080 through 8032 as (0, −1000) and the position vector as (500, 900).

```
/W2 [ 120 [−1000 250 772]
    7080 8032 −1000 500 900
  ]
```

9.7.5 CMaps

9.7.5.1 General

A CMap shall specify the mapping from character codes to character selectors. In PDF, the character selectors shall be CIDs in a CIDFont (as mentioned earlier, PostScript CMaps can use names or codes as well). A CMap serves a function analogous to the **Encoding** dictionary for a simple font. The CMap shall not refer directly to a specific CIDFont; instead, it shall be combined with it as part of a CID-keyed font, represented in PDF as a Type 0 font dictionary (see 9.7.6, "Type 0 Font Dictionaries"). Within the CMap, the character mappings shall refer to the associated CIDFont by *font number*, which in PDF shall be 0.

PDF also uses a special type of CMap to map character codes to Unicode values (see 9.10.3, "ToUnicode CMaps").

A CMap shall specify the writing mode—horizontal or vertical—for any CIDFont with which the CMap is combined. The writing mode determines which metrics shall be used when glyphs are painted from that font.

NOTE Writing mode is specified as part of the CMap because, in some cases, different shapes are used when writing horizontally and vertically. In such cases, the horizontal and vertical variants of a CMap specify different CIDs for a given character code.

A CMap shall be specified in one of two ways:

- As a name object identifying a predefined CMap, whose value shall be one of the predefined CMap names defined in Table 118.
- As a stream object whose contents shall be a CMap file.

9.7.5.2 Predefined CMaps

Several of the CMaps define mappings from Unicode encodings to character collections. Unicode values appearing in a text string shall be represented in big-endian order (high-order byte first). CMap names containing *"UCS2"* use UCS-2 encoding; names containing "*UTF16"* use UTF-16BE (big-endian) encoding.

NOTE 1 Table 118 lists the names of the predefined CMaps. These CMaps map character codes to CIDs in a single descendant CIDFont. CMaps whose names end in H specify horizontal writing mode; those ending in V specify vertical writing mode.

Table 118 – Predefined CJK CMap names

Name	**Description**
Chinese (Simplified)	
GB-EUC-H	Microsoft Code Page 936 (lfCharSet 0x86), GB 2312-80 character set, EUC-CN encoding
GB-EUC-V	Vertical version of GB-EUC-H
GBpc-EUC-H	Mac OS, GB 2312-80 character set, EUC-CN encoding, Script Manager code 19
GBpc-EUC-V	Vertical version of GBpc-EUC-H
GBK-EUC-H	Microsoft Code Page 936 (lfCharSet 0x86), GBK character set, GBK encoding
GBK-EUC-V	Vertical version of GBK-EUC-H
GBKp-EUC-H	Same as GBK-EUC-H but replaces half-width Latin characters with proportional forms and maps character code 0x24 to a dollar sign ($) instead of a yuan symbol (¥)
GBKp-EUC-V	Vertical version of GBKp-EUC-H
GBK2K-H	GB 18030-2000 character set, mixed 1-, 2-, and 4-byte encoding
GBK2K-V	Vertical version of GBK2K-H
UniGB-UCS2-H	Unicode (UCS-2) encoding for the Adobe-GB1 character collection
UniGB-UCS2-V	Vertical version of UniGB-UCS2-H
UniGB-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-GB1 character collection; contains mappings for all characters in the GB18030-2000 character set
UniGB-UTF16-V	Vertical version of UniGB-UTF16-H
Chinese (Traditional)	
B5pc-H	Mac OS, Big Five character set, Big Five encoding, Script Manager code 2
B5pc-V	Vertical version of B5pc-H
HKscs-B5-H	Hong Kong SCS, an extension to the Big Five character set and encoding
HKscs-B5-V	Vertical version of HKscs-B5-H
ETen-B5-H	Microsoft Code Page 950 (lfCharSet 0x88), Big Five character set with ETen extensions
ETen-B5-V	Vertical version of ETen-B5-H
ETenms-B5-H	Same as ETen-B5-H but replaces half-width Latin characters with proportional forms

Table 118 – Predefined CJK CMap names (continued)

Name	Description
ETenms-B5-V	Vertical version of ETenms-B5-H
CNS-EUC-H	CNS 11643-1992 character set, EUC-TW encoding
CNS-EUC-V	Vertical version of CNS-EUC-H
UniCNS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-CNS1 character collection
UniCNS-UCS2-V	Vertical version of UniCNS-UCS2-H
UniCNS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-CNS1 character collection; contains mappings for all the characters in the HKSCS-2001 character set and contains both 2- and 4-byte character codes
UniCNS-UTF16-V	Vertical version of UniCNS-UTF16-H
Japanese	
83pv-RKSJ-H	Mac OS, JIS X 0208 character set with KanjiTalk6 extensions, Shift-JIS encoding, Script Manager code 1
90ms-RKSJ-H	Microsoft Code Page 932 (lfCharSet 0x80), JIS X 0208 character set with NEC and IBM® extensions
90ms-RKSJ-V	Vertical version of 90ms-RKSJ-H
90msp-RKSJ-H	Same as 90ms-RKSJ-H but replaces half-width Latin characters with proportional forms
90msp-RKSJ-V	Vertical version of 90msp-RKSJ-H
90pv-RKSJ-H	Mac OS, JIS X 0208 character set with KanjiTalk7 extensions, Shift-JIS encoding, Script Manager code 1
Add-RKSJ-H	JIS X 0208 character set with Fujitsu FMR extensions, Shift-JIS encoding
Add-RKSJ-V	Vertical version of Add-RKSJ-H
EUC-H	JIS X 0208 character set, EUC-JP encoding
EUC-V	Vertical version of EUC-H
Ext-RKSJ-H	JIS C 6226 (JIS78) character set with NEC extensions, Shift-JIS encoding
Ext-RKSJ-V	Vertical version of Ext-RKSJ-H
H	JIS X 0208 character set, ISO-2022-JP encoding
V	Vertical version of H
UniJIS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-Japan1 character collection
UniJIS-UCS2-V	Vertical version of UniJIS-UCS2-H
UniJIS-UCS2-HW-H	Same as UniJIS-UCS2-H but replaces proportional Latin characters with half-width forms
UniJIS-UCS2-HW-V	Vertical version of UniJIS-UCS2-HW-H
UniJIS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-Japan1 character collection; contains mappings for all characters in the JIS X 0213:1000 character set
UniJIS-UTF16-V	Vertical version of UniJIS-UTF16-H
Korean	

Table 118 – Predefined CJK CMap names (continued)

Name	Description
KSC-EUC-H	KS X 1001:1992 character set, EUC-KR encoding
KSC-EUC-V	Vertical version of KSC-EUC-H
KSCms-UHC-H	Microsoft Code Page 949 (lfCharSet 0x81), KS X 1001:1992 character set plus 8822 additional hangul, Unified Hangul Code (UHC) encoding
KSCms-UHC-V	Vertical version of KSCms–UHC-H
KSCms-UHC-HW-H	Same as KSCms-UHC-H but replaces proportional Latin characters with half-width forms
KSCms-UHC-HW-V	Vertical version of KSCms-UHC-HW-H
KSCpc-EUC-H	Mac OS, KS X 1001:1992 character set with Mac OS KH extensions, Script Manager Code 3
UniKS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-Korea1 character collection
UniKS-UCS2-V	Vertical version of UniKS-UCS2-H
UniKS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-Korea1 character collection
UniKS-UTF16-V	Vertical version of UniKS-UTF16-H
Generic	
Identity-H	The horizontal identity mapping for 2-byte CIDs; may be used with CIDFonts using any **Registry**, **Ordering**, and **Supplement** values. It maps 2-byte character codes ranging from 0 to 65,535 to the same 2-byte CID value, interpreted high-order byte first.
Identity-V	Vertical version of Identity-H. The mapping is the same as for Identity-H.

NOTE 2 The Identity-H and Identity-V CMaps may be used to refer to glyphs directly by their CIDs when showing a text string.

When the current font is a Type 0 font whose Encoding entry is Identity-H or Identity-V, the string to be shown shall contain pairs of bytes representing CIDs, high-order byte first. When the current font is a CIDFont, the string to be shown shall contain pairs of bytes representing CIDs, high-order byte first. When the current font is a Type 2 CIDFont in which the CIDToGIDMap entry is Identity and if the TrueType font is embedded in the PDF file, the 2-byte CID values shall be identical glyph indices for the glyph descriptions in the TrueType font program.

NOTE 3 Table 119 lists the character collections referenced by the predefined CMaps for the different versions of PDF. A dash (—) indicates that the CMap is not predefined in that PDF version.

Table 119 – Character collections for predefined CMaps, by PDF version

CMAP	PDF 1.2	PDF 1.3	PDF 1.4	PDF 1.5
Chinese (Simplified)				
GB-EUC-H/V	Adobe-GB1-0	Adobe-GB1-0	Adobe-GB1-0	Adobe-GB1-0
GBpc-EUC-H	Adobe-GB1-0	Adobe-GB1-0	Adobe-GB1-0	Adobe-GB1-0
GBpc-EUC-V	—	Adobe-GB1-0	Adobe-GB1-0	Adobe-GB1-0
GBK-EUC-H/V	—	Adobe-GB1-2	Adobe-GB1-2	Adobe-GB1-2
GBKp-EUC-H/V	—	—	Adobe-GB1-2	Adobe-GB1-2

Table 119 – Character collections for predefined CMaps, by PDF version (continued)

CMAP	**PDF 1.2**	**PDF 1.3**	**PDF 1.4**	**PDF 1.5**
GBK2K-H/V	—	—	Adobe-GB1-4	Adobe-GB1-4
UniGB-UCS2-H/V	—	Adobe-GB1-2	Adobe-GB1-4	Adobe-GB1-4
UniGB-UTF16-H/V	—	—	—	Adobe-GB1-4
Chinese (Traditional)				
B5pc-H/V	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0
HKscs-B5-H/V	—	—	Adobe-CNS1-3	Adobe-CNS1-3
ETen-B5-H/V	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0
ETenms-B5-H/V	—	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0
CNS-EUC-H/V	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0	Adobe-CNS1-0
UniCNS-UCS2-H/V	—	Adobe-CNS1-0	Adobe-CNS1-3	Adobe-CNS1-3
UniCNS-UTF16-H/V	—	—	—	Adobe-CNS1-4
Japanese				
83pv-RKSJ-H	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1
90ms-RKSJ-H/V	Adobe-Japan1-2	Adobe-Japan1-2	Adobe-Japan1-2	Adobe-Japan1-2
90msp-RKSJ-H/V	—	Adobe-Japan1-2	Adobe-Japan1-2	Adobe-Japan1-2
90pv-RKSJ-H	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1
Add-RKSJ-H/V	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1
EUC-H/V	—	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1
Ext-RKSJ-H/V	Adobe-Japan1-2	Adobe-Japan1-2	Adobe-Japan1-2	Adobe-Japan1-2
H/V	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1	Adobe-Japan1-1
UniJIS-UCS2-H/V	—	Adobe-Japan1-2	Adobe-Japan1-4	Adobe-Japan1-4
UniJIS-UCS2-HW-H/V	—	Adobe-Japan1-2	Adobe-Japan1-4	Adobe-Japan1-4
UniJIS-UTF16-H/V	—	—	—	Adobe-Japan1-5
Korean				
KSC-EUC-H/V	Adobe-Korea1-0	Adobe-Korea1-0	Adobe-Korea1-0	Adobe-Korea1-0
KSCms-UHC-H/V	Adobe-Korea1-1	Adobe-Korea1-1	Adobe-Korea1-1	Adobe-Korea1-1
KSCms-UHC-HW-H/V	—	Adobe-Korea1-1	Adobe-Korea1-1	Adobe-Korea1-1
KSCpc-EUC-H	Adobe-Korea1-0	Adobe-Korea1-0	Adobe-Korea1-0	Adobe-Korea1-0
UniKS-UCS2-H/V	—	Adobe-Korea1-1	Adobe-Korea1-1	Adobe-Korea1-1
UniKS-UTF16-H/V	—	—	—	Adobe-Korea1-2
Generic				
Identity-H/V	Adobe-Identity-0	Adobe-Identity-0	Adobe-Identity-0	Adobe-Identity-0

A conforming reader shall support all of the character collections listed in Table 119. As noted in 9.7.3, "CIDSystemInfo Dictionaries", a character collection is identified by registry, ordering, and supplement number, and supplements are cumulative; that is, a higher-numbered supplement includes the CIDs contained in lower-numbered supplements, as well as some additional CIDs. Consequently, text encoded according to the predefined CMaps for a given PDF version shall be valid when interpreted by a conforming reader supporting the same or a later PDF version. When interpreted by a conforming reader supporting an earlier PDF version, such text causes an error if a CMap is encountered that is not predefined for that PDF version. If character codes are encountered that were added in a higher-numbered supplement than the one corresponding to the supported PDF version, no characters are displayed for those codes; see 9.7.6.3, "Handling Undefined Characters".

The Identity-H and Identity-V CMaps shall not be used with a non-embedded font. Only standardized character sets may be used.

NOTE 4 If a conforming writer producing a PDF file encounters text to be included that uses CIDs from a higher-numbered supplement than the one corresponding to the PDF version being generated, the application should embed the CMap for the higher-numbered supplement rather than refer to the predefined CMap.

The CMap programs that define the predefined CMaps are available through the ASN Web site.

9.7.5.3 Embedded CMap Files

For character encodings that are not predefined, the PDF file shall contain a stream that defines the CMap. In addition to the standard entries for streams (listed in Table 5), the CMap stream dictionary contains the entries listed in Table 120. The data in the stream defines the mapping from character codes to a font number and a character selector. The data shall follow the syntax defined in Adobe Technical Note #5014, Adobe CMap and CIDFont Files Specification (see bibliography).

Table 120 – Additional entries in a CMap stream dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **CMap** for a CMap dictionary.
CMapName	name	*(Required)* The name of the CMap. It shall be the same as the value of **CMapName** in the CMap file.
CIDSystemInfo	dictionary	*(Required)* A dictionary (see 9.7.3, "CIDSystemInfo Dictionaries") containing entries that define the character collection for the CIDFont or CIDFonts associated with the CMap. The value of this entry shall be the same as the value of **CIDSystemInfo** in the CMap file. (However, it does not need to match the values of **CIDSystemInfo** for the Identity-H or Identity-V CMaps.)
WMode	integer	*(Optional)* A code that specifies the writing mode for any CIDFont with which this CMap is combined. The value shall be 0 for horizontal or 1 for vertical. Default value: 0. The value of this entry shall be the same as the value of **WMode** in the CMap file.
UseCMap	name or stream	*(Optional)* The name of a predefined CMap, or a stream containing a CMap. If this entry is present, the referencing CMap shall specify only the character mappings that differ from the referenced CMap.

9.7.5.4 CMap Example and Operator Summary

Embedded CMap files shall conform to the format documented in Adobe Technical Note #5014, subject to these additional constraints:

a) If the embedded CMap file contains a usecmap reference, the CMap indicated there shall also be identified by the UseCMap entry in the CMap stream dictionary.

b) The usefont operator, if present, shall specify a font number of 0.

c) The beginbfchar and endbfchar shall not appear in a CMap that is used as the Encoding entry of a Type 0 font; however, they may appear in the definition of a ToUnicode CMap.

d) A notdef mapping, defined using beginnotdefchar, endnotdefchar, beginnotdefrange, and endnotdefrange shall be used if the normal mapping produces a CID for which no glyph is present in the associated CIDFont.

e) The beginrearrangedfont, endrearrangedfont, beginusematrix, and endusematrix operators shall not be used.

EXAMPLE This example shows a sample CMap for a Japanese Shift-JIS encoding. Character codes in this encoding can be either 1 or 2 bytes in length. This CMap could be used with a CIDFont that uses the same CID ordering as specified in the **CIDSystemInfo** entry. Note that several of the entries in the stream dictionary are also replicated in the stream data.

```
22 0 obj
   << /Type /CMap
        /CMapName /90ms-RKSJ-H
        /CIDSystemInfo << /Registry (Adobe)
                           /Ordering (Japan1)
                           /Supplement 2
                        >>
        /WMode 0
        /Length 23 0 R
   >>

stream
%!PS-Adobe-3.0 Resource-CMap
%%DocumentNeededResources: ProcSet (CIDInit)
%%IncludeResource: ProcSet (CIDInit)
%%BeginResource: CMap (90ms-RKSJ-H)
%%Title: (90ms-RKSJ-H Adobe Japan1 2)
%%Version: 10.001
%%Copyright: Copyright 1990-2001 Adobe Systems Inc.
%%Copyright: All Rights Reserved.
%%EndComments

/CIDInit /ProcSet findresource begin
12 dict begin
begincmap
/CIDSystemInfo
3 dict dup begin
/Registry (Adobe) def
/Ordering (Japan1) def
/Supplement 2 def
end def

/CMapName /90ms-RKSJ-H def
/CMapVersion 10.001 def
/CMapType 1 def
/UIDOffset 950 def
/XUID [1 10 25343] def
/WMode 0 def

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

```
<E040>  <FCFC>
endcodespacerange

1 beginnotdefrange
<00>    <1F>    231
endnotdefrange

100 begincidrange
<20>    <7D>    231
<7E>    <7E>    631
<8140>  <817E>  633
<8180>  <81AC>  696
<81B8>  <81BF>  741
<81C8>  <81CE>  749
…Additional ranges…
<FB40>  <FB7E>  8518
<FB80>  <FBFC>  8581
<FC40>  <FC4B>  8706
endcidrange
endcmap
CMapName currentdict /CMap defineresource pop
end
end
%%EndResource
%%EOF
endstream
endobj
```

9.7.6 Type 0 Font Dictionaries

9.7.6.1 General

A Type 0 font dictionary contains the entries listed in Table 121.

Table 121 – Entries in a Type 0 font dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Font** for a font dictionary.
Subtype	name	*(Required)* The type of font; shall be **Type0** for a Type 0 font.
BaseFont	name	*(Required)* The name of the font. If the descendant is a Type 0 CIDFont, this name should be the concatenation of the CIDFont's **BaseFont** name, a hyphen, and the CMap name given in the **Encoding** entry (or the **CMapName** entry in the CMap). If the descendant is a Type 2 CIDFont, this name should be the same as the CIDFont's **BaseFont** name. NOTE In principle, this is an arbitrary name, since there is no font program associated directly with a Type 0 font dictionary. The conventions described here ensure maximum compatibility with existing readers.
Encoding	name or stream	*(Required)* The name of a predefined CMap, or a stream containing a CMap that maps character codes to font numbers and CIDs. If the descendant is a Type 2 CIDFont whose associated TrueType font program is not embedded in the PDF file, the **Encoding** entry shall be a predefined CMap name (see 9.7.4.2, "Glyph Selection in CIDFonts").

Table 121 – Entries in a Type 0 font dictionary (continued)

Key	Type	Value
DescendantFonts	array	*(Required)* A one-element array specifying the CIDFont dictionary that is the descendant of this Type 0 font.
ToUnicode	stream	*(Optional)* A stream containing a CMap file that maps character codes to Unicode values (see 9.10, "Extraction of Text Content").

EXAMPLE This code sample shows a Type 0 font.

```
14 0 obj
   << /Type /Font
        /Subtype /Type0
        /BaseFont /HeiseiMin-W5-90ms-RKSJ-H
        /Encoding /90ms-RKSJ-H
        /DescendantFonts [15 0 R]
   >>
endobj
```

9.7.6.2 CMap Mapping

The **Encoding** entry of a Type 0 font dictionary specifies a CMap that specifies how text-showing operators (such as **Tj**) shall interpret the bytes in the string to be shown when the current font is the Type 0 font. This sub-clause describes how the characters in the string shall be decoded and mapped into character selectors, which in PDF are always CIDs.

The codespace ranges in the CMap (delimited by **begincodespacerange** and **endcodespacerange**) specify how many bytes are extracted from the string for each successive character code. A codespace range shall be specified by a pair of codes of some particular length giving the lower and upper bounds of that range. A code shall be considered to match the range if it is the same length as the bounding codes and the value of each of its bytes lies between the corresponding bytes of the lower and upper bounds. The code length shall not be greater than 4.

A sequence of one or more bytes shall be extracted from the string and matched against the codespace ranges in the CMap. That is, the first byte shall be matched against 1-byte codespace ranges; if no match is found, a second byte shall be extracted, and the 2-byte code shall be matched against 2-byte codespace ranges. This process continues for successively longer codes until a match is found or all codespace ranges have been tested. There will be at most one match because codespace ranges shall not overlap.

The code extracted from the string shall be looked up in the character code mappings for codes of that length. (These are the mappings defined by **beginbfchar**, **endbfchar**, **begincidchar**, **endcidchar**, and corresponding operators for ranges.) Failing that, it shall be looked up in the notdef mappings, as described in the next sub-clause.

The results of the CMap mapping algorithm are a font number and a character selector. The font number shall be used as an index into the Type 0 font's **DescendantFonts** array to select a CIDFont. In PDF, the font number shall be 0 and the character selector shall be a CID; this is the only case described here. The CID shall then be used to select a glyph in the CIDFont. If the CIDFont contains no glyph for that CID, the *notdef mappings* shall be consulted, as described in 9.7.6.3, "Handling Undefined Characters".

9.7.6.3 Handling Undefined Characters

A CMap mapping operation can fail to select a glyph for a variety of reasons. This sub-clause describes those reasons and what happens when they occur.

If a code maps to a CID for which no such glyph exists in the descendant CIDFont, the *notdef mappings* in the CMap shall be consulted to obtain a substitute character selector. These mappings are delimited by the operators **beginnotdefchar**, **endnotdefchar**, **beginnotdefrange**, and **endnotdefrange** within an embedded CMap file. They shall always map to a CID. If a matching notdef mapping is found, the CID selects a glyph in the associated descendant, which shall be a CIDFont. If no glyph exists for that CID, the glyph for CID 0 (which shall be present) shall be substituted.

NOTE 5 The *notdef mappings are similar to* the .notdef character mechanism in simple fonts.

If the CMap does not contain either a character mapping or a notdef mapping for the code, descendant 0 shall be selected and the glyph for CID 0 shall be substituted from the associated CIDFont.

If the code is invalid—that is, the bytes extracted from the string to be shown do not match any codespace range in the CMap—a substitute glyph is chosen as just described. The character mapping algorithm shall be reset to its original position in the string, and a modified mapping algorithm chooses the best partially matching codespace range:

a) If the first byte extracted from the string to be shown does not match the first byte of any codespace range, the range having the shortest codes shall be chosen.

b) Otherwise (that is, if there is a partial match), for each additional byte extracted, the code accumulated so far shall be matched against the beginnings of all longer codespace ranges until the longest such partial match has been found. If multiple codespace ranges have partial matches of the same length, the one having the shortest codes shall be chosen.

The length of the codes in the chosen codespace range determines the total number of bytes to consume from the string for the current mapping operation.

9.8 Font Descriptors

9.8.1 General

A *font descriptor* specifies metrics and other attributes of a simple font or a CIDFont as a whole, as distinct from the metrics of individual glyphs. These font metrics provide information that enables a conforming reader to synthesize a substitute font or select a similar font when the font program is unavailable. The font descriptor may also be used to embed the font program in the PDF file.

Font descriptors shall not be used with Type 0 fonts. Beginning with PDF 1.5, font descriptors may be used with Type 3 fonts.

A font descriptor is a dictionary whose entries specify various font attributes. The entries common to all font descriptors—for both simple fonts and CIDFonts—are listed in Table 122. Additional entries in the font descriptor for a CIDFont are described in 9.8.3, "Font Descriptors for CIDFonts". All integer values shall be units in glyph space. The conversion from glyph space to text space is described in 9.2.4, "Glyph Positioning and Metrics".

Table 122 – Entries common to all font descriptors

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **FontDescriptor** for a font descriptor.
FontName	name	*(Required)* The PostScript name of the font. This name shall be the same as the value of **BaseFont** in the font or CIDFont dictionary that refers to this font descriptor.
FontFamily	byte string	*(Optional; PDF 1.5; should be used for Type 3 fonts in Tagged PDF documents)* A byte string specifying the preferred font family name. EXAMPLE 1 For the font Times Bold Italic, the **FontFamily** is Times.

Table 122 – Entries common to all font descriptors (continued)

Key	Type	Value
FontStretch	name	*(Optional; PDF 1.5; should be used for Type 3 fonts in Tagged PDF documents)* The font stretch value. It shall be one of these names (ordered from narrowest to widest): UltraCondensed, ExtraCondensed, Condensed, SemiCondensed, Normal, SemiExpanded, Expanded, ExtraExpanded or UltraExpanded. The specific interpretation of these values varies from font to font. EXAMPLE 2 ***Condensed*** in one font may appear most similar to ***Normal*** in another.
FontWeight	number	*(Optional; PDF 1.5; should be used for Type 3 fonts in Tagged PDF documents)* The weight (thickness) component of the fully-qualified font name or font specifier. The possible values shall be 100, 200, 300, 400, 500, 600, 700, 800, or 900, where each number indicates a weight that is at least as dark as its predecessor. A value of 400 shall indicate a normal weight; 700 shall indicate bold. The specific interpretation of these values varies from font to font. EXAMPLE 3 300 in one font may appear most similar to 500 in another.
Flags	integer	*(Required)* A collection of flags defining various characteristics of the font (see 9.8.2, "Font Descriptor Flags").
FontBBox	rectangle	*(Required, except for Type 3 fonts)* A rectangle (see 7.9.5, "Rectangles"), expressed in the glyph coordinate system, that shall specify the *font bounding box*. This should be the smallest rectangle enclosing the shape that would result if all of the glyphs of the font were placed with their origins coincident and then filled.
ItalicAngle	number	*(Required)* The angle, expressed in degrees counterclockwise from the vertical, of the dominant vertical strokes of the font. EXAMPLE 4 The 9-o'clock position is 90 degrees, and the 3-o'clock position is –90 degrees. The value shall be negative for fonts that slope to the right, as almost all italic fonts do.
Ascent	number	*(Required, except for Type 3 fonts)* The maximum height above the baseline reached by glyphs in this font. The height of glyphs for accented characters shall be excluded.
Descent	number	*(Required, except for Type 3 fonts)* The maximum depth below the baseline reached by glyphs in this font. The value shall be a negative number.
Leading	number	*(Optional)* The spacing between baselines of consecutive lines of text. Default value: 0.
CapHeight	number	*(Required for fonts that have Latin characters, except for Type 3 fonts)* The vertical coordinate of the top of flat capital letters, measured from the baseline.
XHeight	number	*(Optional)* The font's *x height*: the vertical coordinate of the top of flat nonascending lowercase letters (like the letter *x*), measured from the baseline, in fonts that have Latin characters. Default value: 0.
StemV	number	*(Required, except for Type 3 fonts)* The thickness, measured horizontally, of the dominant vertical stems of glyphs in the font.
StemH	number	*(Optional)* The thickness, measured vertically, of the dominant horizontal stems of glyphs in the font. Default value: 0.

Table 122 – Entries common to all font descriptors (continued)

Key	Type	Value
AvgWidth	number	*(Optional)* The average width of glyphs in the font. Default value: 0.
MaxWidth	number	*(Optional)* The maximum width of glyphs in the font. Default value: 0.
MissingWidth	number	*(Optional)* The width to use for character codes whose widths are not specified in a font dictionary's **Widths** array. This shall have a predictable effect only if all such codes map to glyphs whose actual widths are the same as the value of the **MissingWidth** entry. Default value: 0.
FontFile	stream	*(Optional)* A stream containing a Type 1 font program (see 9.9, "Embedded Font Programs").
FontFile2	stream	*(Optional; PDF 1.1)* A stream containing a TrueType font program (see 9.9, "Embedded Font Programs").
FontFile3	stream	*(Optional; PDF 1.2)* A stream containing a font program whose format is specified by the **Subtype** entry in the stream dictionary (see Table 126).
CharSet	ASCII string or byte string	*(Optional; meaningful only in Type 1 fonts; PDF 1.1)* A string listing the character names defined in a font subset. The names in this string shall be in PDF syntax—that is, each name preceded by a slash (/). The names may appear in any order. The name .notdef shall be omitted; it shall exist in the font subset. If this entry is absent, the only indication of a font subset shall be the subset tag in the **FontName** entry (see 9.6.4, "Font Subsets").

At most, only one of the FontFile, FontFile2, and FontFile3 entries shall be present.

9.8.2 Font Descriptor Flags

The value of the **Flags** entry in a font descriptor shall be an unsigned 32-bit integer containing flags specifying various characteristics of the font. Bit positions within the flag word are numbered from 1 (low-order) to 32 (high-order). Table 123 shows the meanings of the flags; all undefined flag bits are reserved and shall be set to 0 by conforming writers. Figure 48 shows examples of fonts with these characteristics.

Table 123 – Font flags

Bit position	Name	Meaning
1	FixedPitch	All glyphs have the same width (as opposed to proportional or variable-pitch fonts, which have different widths).
2	Serif	Glyphs have serifs, which are short strokes drawn at an angle on the top and bottom of glyph stems. (*Sans serif* fonts do not have serifs.)
3	Symbolic	Font contains glyphs outside the Adobe standard Latin character set. This flag and the Nonsymbolic flag shall not both be set or both be clear.
4	Script	Glyphs resemble cursive handwriting.
6	Nonsymbolic	Font uses the Adobe standard Latin character set or a subset of it.
7	Italic	Glyphs have dominant vertical strokes that are slanted.
17	AllCap	Font contains no lowercase letters; typically used for display purposes, such as for titles or headlines.

Table 123 – Font flags (continued)

Bit position	Name	Meaning
18	SmallCap	Font contains both uppercase and lowercase letters. The uppercase letters are similar to those in the regular version of the same typeface family. The glyphs for the lowercase letters have the same shapes as the corresponding uppercase letters, but they are sized and their proportions adjusted so that they have the same size and stroke weight as lowercase glyphs in the same typeface family.
19	ForceBold	See description after Note 1 in this sub-clause.

The Nonsymbolic flag (bit 6 in the **Flags** entry) indicates that the font's character set is the Adobe standard Latin character set (or a subset of it) and that it uses the standard names for those glyphs. This character set is shown in D.2, "Latin Character Set and Encodings". If the font contains any glyphs outside this set, the Symbolic flag shall be set and the Nonsymbolic flag shall be clear. In other words, any font whose character set is not a subset of the Adobe standard character set shall be considered to be symbolic. This influences the font's implicit base encoding and may affect a conforming reader's font substitution strategies.

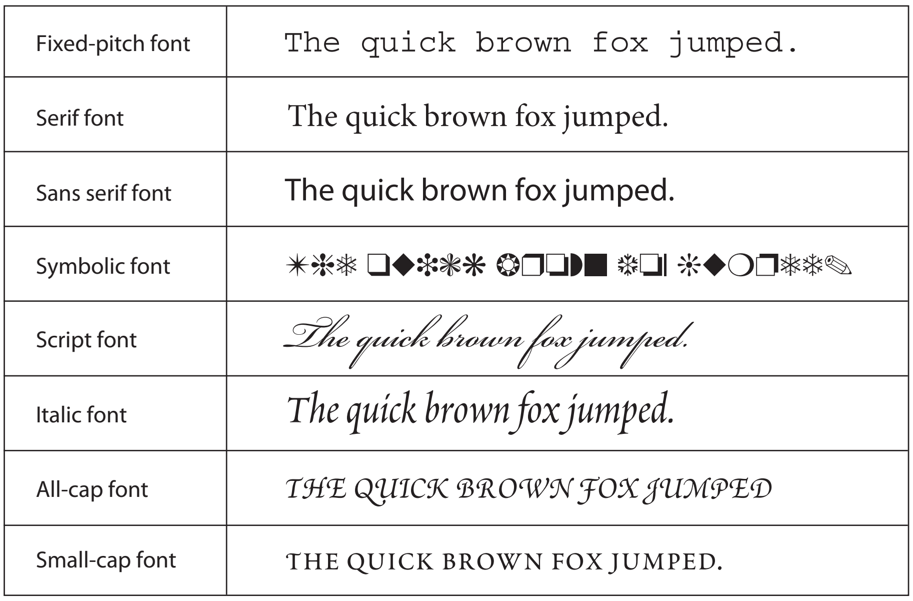

Fixed-pitch font
The quick brown fox jumped.
Serif font
The quick brown fox jumped.
Sans serif font
The quick brown fox jumped.
Symbolic font
Script font
The quick brown fox jumped.
Italic font
The quick brown fox jumped.
All-cap font
THE QUICK BROWN FOX JUMPED
Small-cap font
THE QUICK BROWN FOX JUMPED.

Figure 48 – Characteristics represented in the Flags entry of a font descriptor

NOTE 1 This classification of nonsymbolic and symbolic fonts is peculiar to PDF. A font may contain additional characters that are used in Latin writing systems but are outside the Adobe standard Latin character set; PDF considers such a font to be symbolic. The use of two flags to represent a single binary choice is a historical accident.

The ForceBold flag (bit 19) shall determine whether bold glyphs shall be painted with extra pixels even at very small text sizes by a conforming reader. If the ForceBold flag is set, features of bold glyphs may be thickened at small text sizes.

NOTE 2 Typically, when glyphs are painted at small sizes on very low-resolution devices such as display screens, features of bold glyphs may appear only 1 pixel wide. Because this is the minimum feature width on a pixel-based device, ordinary (nonbold) glyphs also appear with 1-pixel-wide features and therefore cannot be distinguished from bold glyphs.

EXAMPLE This code sample illustrates a font descriptor whose Flags entry has the Serif, Nonsymbolic, and ForceBold flags (bits 2, 6, and 19) set.

```
7 0 obj
   << /Type /FontDescriptor
       /FontName /AGaramond-Semibold
       /Flags 262178                          % Bits 2, 6, and 19
       /FontBBox [−177 −269 1123 866]
       /MissingWidth 255
       /StemV 105
       /StemH 45
       /CapHeight 660
       /XHeight 394
       /Ascent 720
       /Descent −270
       /Leading 83
       /MaxWidth 1212
       /AvgWidth 478
       /ItalicAngle 0
   >>
endobj
```

9.8.3 Font Descriptors for CIDFonts

9.8.3.1 General

In addition to the entries in Table 122, the **FontDescriptor** dictionaries of CIDFonts may contain the entries listed in Table 124.

Table 124 – Additional font descriptor entries for CIDFonts

Key	Type	Value
Style	dictionary	*(Optional)* A dictionary containing entries that describe the style of the glyphs in the font (see 9.8.3.2, "Style").
Lang	name	*(Optional; PDF 1.5)* A name specifying the language of the font, which may be used for encodings where the language is not implied by the encoding itself. The value shall be one of the codes defined by Internet RFC 3066, *Tags for the Identification of Languages* or (*PDF 1.0*) 2-character language codes defined by ISO 639 (see the Bibliography). If this entry is absent, the language shall be considered to be unknown.
FD	dictionary	*(Optional)* A dictionary whose keys identify a class of glyphs in a CIDFont. Each value shall be a dictionary containing entries that shall override the corresponding values in the main font descriptor dictionary for that class of glyphs (see 9.8.3.3, "FD").
CIDSet	stream	*(Optional)* A stream identifying which CIDs are present in the CIDFont file. If this entry is present, the CIDFont shall contain only a subset of the glyphs in the character collection defined by the **CIDSystemInfo** dictionary. If it is absent, the only indication of a CIDFont subset shall be the subset tag in the **FontName** entry (see 9.6.4, "Font Subsets"). The stream's data shall be organized as a table of bits indexed by CID. The bits shall be stored in bytes with the high-order bit first. Each bit shall correspond to a CID. The most significant bit of the first byte shall correspond to CID 0, the next bit to CID 1, and so on.

9.8.3.2 Style

The **Style** dictionary contains entries that define style attributes and values for the CIDFont. Only the **Panose** entry is defined. The value of **Panose** shall be a 12-byte string consisting of these elements:

- The font family class and subclass ID bytes, given in the sFamilyClass field of the "OS/2" table in a TrueType font. This field is documented in Microsoft's *TrueType 1.0 Font Files Technical Specification*.
- Ten bytes for the PANOSE classification number for the font. The PANOSE classification system is documented in Hewlett-Packard Company's *PANOSE Classification Metrics Guide*.

See the Bibliography for more information about these documents.

EXAMPLE This is an example of a **Style** entry in the font descriptor:

```
/Style << /Panose <01 05 02 02 03 00 00 00 00 00 00 00> >>
```

9.8.3.3 FD

A CIDFont may be made up of different classes of glyphs, each class requiring different sets of the font-wide attributes that appear in font descriptors.

EXAMPLE 1 Latin glyphs, for example, may require different attributes than kanji glyphs.

The font descriptor shall define a set of default attributes that apply to all glyphs in the CIDFont. The **FD** entry in the font descriptor shall contain exceptions to these defaults.

The key for each entry in an **FD** dictionary shall be the name of a class of glyphs—that is, a particular subset of the CIDFont's character collection. The entry's value shall be a font descriptor whose contents shall override the font-wide attributes for that class only. This font descriptor shall contain entries for metric information only; it shall not include **FontFile**, **FontFile2**, **FontFile3**, or any of the entries listed in Table 122.

The **FD** dictionary should contain at least the metrics for the proportional Latin glyphs. With the information for these glyphs, a more accurate substitution font can be created.

The names of the glyph classes depend on the character collection, as identified by the **Registry**, **Ordering**, and **Supplement** entries in the **CIDSystemInfo** dictionary. Table 125 lists the valid keys for the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, Adobe-Japan2, and Adobe-Korea1 character collections.

Table 125 – Glyph classes in CJK fonts

Character Collection	Class	Glyphs in Class
Adobe-GB1	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	Dingbats	Special symbols
	Generic	Typeface-independent glyphs, such as line-drawing
	Hanzi	Full-width hanzi (Chinese) glyphs
	HRoman	Half-width Latin glyphs
	HRomanRot	Same as **HRoman** but rotated for use in vertical writing
	Kana	Japanese kana (katakana and hiragana) glyphs
	Proportional	Proportional Latin glyphs
	ProportionalRot	Same as **Proportional** but rotated for use in vertical writing
Adobe-CNS1	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	Dingbats	Special symbols
	Generic	Typeface-independent glyphs, such as line-drawing
	Hanzi	Full-width hanzi (Chinese) glyphs
	HRoman	Half-width Latin glyphs
	HRomanRot	Same as **HRoman** but rotated for use in vertical writing
	Kana	Japanese kana (katakana and hiragana) glyphs
	Proportional	Proportional Latin glyphs
	ProportionalRot	Same as **Proportional** but rotated for use in vertical writing
Adobe-Japan1	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	AlphaNum	Numeric glyphs
	Dingbats	Special symbols
	DingbatsRot	Same as **Dingbats** but rotated for use in vertical writing
	Generic	Typeface-independent glyphs, such as line-drawing
	GenericRot	Same as **Generic** but rotated for use in vertical writing
	HKana	Half-width kana (katakana and hiragana) glyphs
	HKanaRot	Same as **HKana** but rotated for use in vertical writing
	HRoman	Half-width Latin glyphs
	HRomanRot	Same as **HRoman** but rotated for use in vertical writing
	Kana	Full-width kana (katakana and hiragana) glyphs
	Kanji	Full-width kanji (Chinese) glyphs
	Proportional	Proportional Latin glyphs
	ProportionalRot	Same as **Proportional** but rotated for use in vertical writing
	Ruby	Glyphs used for setting ruby (small glyphs that serve to annotate other glyphs with meanings or readings)
Adobe-Japan2	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	Dingbats	Special symbols
	HojoKanji	Full-width kanji glyphs

Table 125 – Glyph classes in CJK fonts (continued)

Character Collection	Class	Glyphs in Class
Adobe-Korea1	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	Dingbats	Special symbols
	Generic	Typeface-independent glyphs, such as line-drawing
	Hangul	Hangul and jamo glyphs
	Hanja	Full-width hanja (Chinese) glyphs
	HRoman	Half-width Latin glyphs
	HRomanRot	Same as **HRoman** but rotated for use in vertical writing
	Kana	Japanese kana (katakana and hiragana) glyphs
	Proportional	Proportional Latin glyphs
	ProportionalRot	Same as **Proportional** but rotated for use in vertical writing

EXAMPLE 2 This example illustrates an **FD** dictionary containing two entries.

```
/FD   <<   /Proportional  25 0 R
             /HKana  26 0 R
      >>

25  0  obj
    <<  /Type  /FontDescriptor
         /FontName  /HeiseiMin-W3-Proportional
         /Flags  2
         /AvgWidth  478
         /MaxWidth  1212
         /MissingWidth  250
         /StemV  105
         /StemH  45
         /CapHeight  660
         /XHeight  394
         /Ascent  720
         /Descent  –270
         /Leading  83
    >>
endobj

26  0  obj
    <<  /Type  /FontDescriptor
         /FontName  /HeiseiMin-W3-HKana
         /Flags  3
         /AvgWidth  500
         /MaxWidth  500
         /MissingWidth  500
         /StemV  50
         /StemH  75
         /Ascent  720
         /Descent  0
         /Leading  83
    >>
endobj
```

9.9 Embedded Font Programs

A font program may be embedded in a PDF file as data contained in a PDF stream object.

NOTE 1 Such a stream object is also called a *font file* by analogy with font programs that are available from sources external to the conforming writer.

Font programs are subject to copyright, and the copyright owner may impose conditions under which a font program may be used. These permissions are recorded either in the font program or as part of a separate license. One of the conditions may be that the font program cannot be embedded, in which case it should not be incorporated into a PDF file. A font program may allow embedding for the sole purpose of viewing and printing the document but not for creating new or modified text that uses the font (in either the same document or other documents). The latter operation would require the user performing the operation to have a licensed copy of the font program, not a copy extracted from the PDF file. In the absence of explicit information to the contrary, embedded font programs shall be used only to view and print the document and not for any other purposes.

Table 126 summarizes the ways in which font programs shall be embedded in a PDF file, depending on the representation of the font program. The key shall be the name used in the font descriptor to refer to the font file stream; the subtype shall be the value of the **Subtype** key, if present, in the font file stream dictionary. Further details of specific font program representations are given below.

Table 126 – Embedded font organization for various font types

Key	**Subtype**	**Description**
FontFile	—	Type 1 font program, in the original (noncompact) format described in *Adobe Type 1 Font Format*. This entry may appear in the font descriptor for a **Type1** or **MMType1** font dictionary.
FontFile2	—	*(PDF 1.1)* TrueType font program, as described in the *TrueType Reference Manual*. This entry may appear in the font descriptor for a **TrueType** font dictionary or (*PDF 1.3*) for a **CIDFontType2** CIDFont dictionary.
FontFile3	**Type1C**	*(PDF 1.2)* Type 1–equivalent font program represented in the Compact Font Format (CFF), as described in Adobe Technical Note #5176, *The Compact Font Format Specification*. This entry may appear in the font descriptor for a **Type1** or **MMType1** font dictionary.
FontFile3	**CIDFontType0C**	*(PDF 1.3)* Type 0 CIDFont program represented in the Compact Font Format (CFF), as described in Adobe Technical Note #5176, *The Compact Font Format Specification*. This entry may appear in the font descriptor for a **CIDFontType0** CIDFont dictionary.

Table 126 – Embedded font organization for various font types (continued)

Key	Subtype	Description
FontFile3	**OpenType**	(*PDF 1.6*) OpenType® font program, as described in the *OpenType Specification v.1.4* (see the Bibliography). OpenType is an extension of TrueType that allows inclusion of font programs that use the Compact Font Format (CFF). A **FontFile3** entry with an **OpenType** subtype may appear in the font descriptor for these types of font dictionaries: • A **TrueType** font dictionary or a **CIDFontType2** CIDFont dictionary, if the embedded font program contains a “glyf” table. In addition to the “glyf” table, the font program must include these tables: “head”, “hhea”, “hmtx”, “loca”, and “maxp”. The “cvt ” (notice the trailing SPACE), “fpgm”, and “prep” tables must also be included if they are required by the font instructions. • A **CIDFontType0** CIDFont dictionary, if the embedded font program contains a “CFF ” table (notice the trailing SPACE) with a Top DICT that uses CIDFont operators (this is equivalent to subtype **CIDFontType0C**). In addition to the “CFF ” table, the font program must include the “cmap” table. • A **Type1** font dictionary or **CIDFontType0** CIDFont dictionary, if the embedded font program contains a “CFF ” table without CIDFont operators. In addition to the “CFF ” table, the font program must include the “cmap” table. The *OpenType Specification* describes a set of required tables; however, not all tables are required in the font file, as described for each type of font dictionary that can include this entry. NOTE The absence of some optional tables (such as those used for advanced line layout) may prevent editing of text containing the font.

The stream dictionary for a font file shall contain the normal entries for a stream, such as **Length** and **Filter** (listed in Table 5), plus the additional entries listed in Table 127.

Table 127 – Additional entries in an embedded font stream dictionary

Key	Type	Value
Length1	integer	*(Required for Type 1 and TrueType fonts)* The length in bytes of the clear-text portion of the Type 1 font program, or the entire TrueType font program, after it has been decoded using the filters specified by the stream’s **Filter** entry, if any.
Length2	integer	*(Required for Type 1 fonts)* The length in bytes of the encrypted portion of the Type 1 font program after it has been decoded using the filters specified by the stream’s **Filter** entry.
Length3	integer	*(Required for Type 1 fonts)* The length in bytes of the fixed-content portion of the Type 1 font program after it has been decoded using the filters specified by the stream’s **Filter** entry. If **Length3** is 0, it indicates that the 512 zeros and **cleartomark** have not been included in the **FontFile** font program and shall be added by the conforming reader.
Subtype	name	*(Required if referenced from **FontFile3**; PDF 1.2)* A name specifying the format of the embedded font program. The name shall be **Type1C** for Type 1 compact fonts, **CIDFontType0C** for Type 0 compact CIDFonts, or **OpenType** for OpenType fonts.
Metadata	stream	*(Optional; PDF 1.4)* A *metadata stream* containing metadata for the embedded font program (see 14.3.2, "Metadata Streams").

NOTE 2 A standard Type 1 font program, as described in the *Adobe Type 1 Font Format* specification, consists of three parts: a clear-text portion (written using PostScript syntax), an encrypted portion, and a fixed-content portion. The fixed-content portion contains 512 ASCII zeros followed by a **cleartomark** operator, and perhaps followed

by additional data. Although the encrypted portion of a standard Type 1 font may be in binary or ASCII hexadecimal format, PDF supports only the binary format. However, the entire font program may be encoded using any filters.

EXAMPLE This code shows the structure of an embedded standard Type 1 font.

```
12  0  obj
    <<  /Filter  /ASCII85Decode
         /Length  41116
         /Length1  2526
         /Length2  32393
         /Length3  570
    >>
stream
,p>`rDKJj'E+LaU0eP.@+AH9dBOu$hFD55nC
…Omitted data…
JJQ&Nt')<=^p&mGf(%:%h1%9c//K(/*o=.C>UXkbVGTrr~>
endstream
endobj
```

As noted in Table 126, a Type 1–equivalent font program or a Type 0 CIDFont program may be represented in the Compact Font Format (CFF). The **Length1**, **Length2**, and **Length3** entries are not needed in that case and shall not be present. Although CFF enables multiple font or CIDFont programs to be bundled together in a single file, an embedded CFF font file in PDF shall consist of exactly one font or CIDFont (as appropriate for the associated font dictionary).

According to the Adobe Type 1 Font Format specification, a Type 1 font program may contain a ***PaintType*** entry specifying whether the glyphs' outlines are to be filled or stroked. For fonts embedded in a PDF file, this entry shall be ignored; the decision whether to fill or stroke glyph outlines is entirely determined by the PDF text rendering mode parameter (see 9.3.6, "Text Rendering Mode"). This shall also applies to Type 1 compact fonts and Type 0 compact CIDFonts.

A TrueType font program may be used as part of either a font or a CIDFont. Although the basic font file format is the same in both cases, there are different requirements for what information shall be present in the font program. These TrueType tables shall always be present if present in the original TrueType font program: "head", "hhea", "loca", "maxp", "cvt", "prep", "glyf", "hmtx", and "fpgm". If used with a simple font dictionary, the font program shall additionally contain a cmap table defining one or more encodings, as discussed in 9.6.6.4, "Encodings for TrueType Fonts". If used with a CIDFont dictionary, the cmap table is not needed and shall not be present, since the mapping from character codes to glyph descriptions is provided separately.

The "vhea" and "vmtx" tables that specify vertical metrics shall never be used by a conforming reader. The only way to specify vertical metrics in PDF shall be by means of the ***DW2*** and ***W2*** entries in a CIDFont dictionary.

NOTE 3 Beginning with PDF 1.6, font programs may be embedded using the OpenType format, which is an extension of the TrueType format that allows inclusion of font programs using the Compact Font Format (CFF). It also allows inclusion of data to describe glyph substitutions, kerning, and baseline adjustments. In addition to rendering glyphs, conforming readers may use the data in OpenType fonts to do advanced line layout, automatically substitute ligatures, provide selections of alternate glyphs to users, and handle complex writing scripts.

The process of finding glyph descriptions in OpenType fonts by a conforming reader shall be the following:

- For Type 1 fonts using "CFF" tables, the process shall be as described in 9.6.6.2, "Encodings for Type 1 Fonts".

- For TrueType fonts using "glyf" tables, the process shall be as described in 9.6.6.4, "Encodings for TrueType Fonts". Since this process sometimes produces ambiguous results, conforming writers, instead of using a simple font, shall use a Type 0 font with an Identity-H encoding and use the glyph indices as character codes, as described following Table 118.

- For **CIDFontType0** fonts using “CFF” tables, the process shall be as described in the discussion of embedded Type 0 CIDFonts in 9.7.4.2, "Glyph Selection in CIDFonts".
- For **CIDFontType2** fonts using “glyf” tables, the process shall be as described in the discussion of embedded Type 2 CIDFonts in 9.7.4.2, "Glyph Selection in CIDFonts".

As discussed in 9.6.4, "Font Subsets", an embedded font program may contain only the subset of glyphs that are used in the PDF document. This may be indicated by the presence of a **CharSet** or **CIDSet** entry in the font descriptor that refers to the font file.

9.10 Extraction of Text Content

9.10.1 General

The preceding sub-clauses describe all the facilities for showing text and causing glyphs to be painted on the page. In addition to displaying text, conforming readers sometimes need to determine the information content of text—that is, its meaning according to some standard character identification as opposed to its rendered appearance. This need arises during operations such as searching, indexing, and exporting of text to other file formats.

The Unicode standard defines a system for numbering all of the common characters used in a large number of languages. It is a suitable scheme for representing the information content of text, but not its appearance, since Unicode values identify characters, not glyphs. For information about Unicode, see the *Unicode Standard* by the Unicode Consortium (see the Bibliography).

When extracting character content, a conforming reader can easily convert text to Unicode values if a font’s characters are identified according to a standard character set that is known to the conforming reader. This character identification can occur if either the font uses a standard named encoding or the characters in the font are identified by standard character names or CIDs in a well-known collection. 9.10.2, "Mapping Character Codes to Unicode Values", describes in detail the overall algorithm for mapping character codes to Unicode values.

If a font is not defined in one of these ways, the glyphs can still be shown, but the characters cannot be converted to Unicode values without additional information:

- This information can be provided as an optional **ToUnicode** entry in the font dictionary *(PDF 1.2*; see 9.10.3, "ToUnicode CMaps"*)*, whose value shall be a stream object containing a special kind of CMap file that maps character codes to Unicode values.
- An **ActualText** entry for a structure element or marked-content sequence (see 14.9.4, "Replacement Text") may be used to specify the text content directly.

9.10.2 Mapping Character Codes to Unicode Values

A conforming reader can use these methods, in the priority given, to map a character code to a Unicode value. Tagged PDF documents, in particular, shall provide at least one of these methods (see 14.8.2.4.2, "Unicode Mapping in Tagged PDF"):

- If the font dictionary contains a **ToUnicode** CMap (see 9.10.3, "ToUnicode CMaps"), use that CMap to convert the character code to Unicode.
- If the font is a simple font that uses one of the predefined encodings **MacRomanEncoding**, **MacExpertEncoding**, or **WinAnsiEncoding**, or that has an encoding whose **Differences** array includes only character names taken from the Adobe standard Latin character set and the set of named characters in the Symbol font (see Annex D):

    a) Map the character code to a character name according to Table D.1 and the font’s **Differences** array.

b) Look up the character name in the *Adobe Glyph List* (see the Bibliography) to obtain the corresponding Unicode value.

• If the font is a composite font that uses one of the predefined CMaps listed in Table 118 (except Identity–H and Identity–V) or whose descendant CIDFont uses the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, or Adobe-Korea1 character collection:

a) Map the character code to a character identifier (CID) according to the font's CMap.

b) Obtain the registry and ordering of the character collection used by the font's CMap (for example, Adobe and Japan1) from its **CIDSystemInfo** dictionary.

c) Construct a second CMap name by concatenating the registry and ordering obtained in step (b) in the format *registry–ordering*–UCS2 (for example, Adobe–Japan1–UCS2).

d) Obtain the CMap with the name constructed in step (c) (available from the ASN Web site; see the Bibliography).

e) Map the CID obtained in step (a) according to the CMap obtained in step (d), producing a Unicode value.

NOTE Type 0 fonts whose descendant CIDFonts use the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, or Adobe-Korea1 character collection (as specified in the **CIDSystemInfo** dictionary) shall have a supplement number corresponding to the version of PDF supported by the conforming reader. See Table 3 for a list of the character collections corresponding to a given PDF version. (Other supplements of these character collections can be used, but if the supplement is higher-numbered than the one corresponding to the supported PDF version, only the CIDs in the latter supplement are considered to be standard CIDs.)

If these methods fail to produce a Unicode value, there is no way to determine what the character code represents in which case a conforming reader may choose a character code of their choosing.

9.10.3 ToUnicode CMaps

The CMap defined in the **ToUnicode** entry of the font dictionary shall follow the syntax for CMaps introduced in 9.7.5, "CMaps" and fully documented in Adobe Technical Note #5014, *Adobe CMap and CIDFont Files Specification*. Additional guidance regarding the CMap defined in this entry is provided in Adobe Technical Note #5411, *ToUnicode Mapping File Tutorial*. This CMap differs from an ordinary one in these ways:

• The only pertinent entry in the CMap stream dictionary (see Table 120) is **UseCMap**, which may be used if the CMap is based on another **ToUnicode** CMap.

• The CMap file shall contain **begincodespacerange** and **endcodespacerange** operators that are consistent with the encoding that the font uses. In particular, for a simple font, the codespace shall be one byte long.

• It shall use the **beginbfchar**, **endbfchar**, **beginbfrange**, and **endbfrange** operators to define the mapping from character codes to Unicode character sequences expressed in UTF-16BE encoding.

EXAMPLE 1 This example illustrates a Type 0 font that uses the Identity-H CMap to map from character codes to CIDs and whose descendant CIDFont uses the Identity mapping from CIDs to TrueType glyph indices. Text strings shown using this font simply use a 2-byte glyph index for each glyph. In the absence of a ToUnicode entry, no information would be available about what the glyphs mean.

```
14 0 obj
   << /Type /Font
       /Subtype /Type0
       /BaseFont /Ryumin−Light
       /Encoding /Identity−H
       /DescendantFonts [15 0 R]
       /ToUnicode 16 0 R
```

```
    >>
endobj

15  0  obj
    <<  /Type  /Font
          /Subtype  /CIDFontType2
          /BaseFont  /Ryumin−Light
          /CIDSystemInfo  17 0 R
          /FontDescriptor  18 0 R
         /CIDToGIDMap  /Identity
    >>
endobj
```

EXAMPLE 2 In this example, the value of the ToUnicode entry is a stream object that contains the definition of the CMap.

The **begincodespacerange** and **endcodespacerange** operators define the source character code range to be the 2-byte character codes from <00 00> to <FF FF>. The specific mappings for several of the character codes are shown.

```
16  0  obj
    <<  /Length  433  >>
stream
/CIDInit  /ProcSet  findresource  begin
12  dict  begin
begincmap
/CIDSystemInfo
<<  /Registry  (Adobe)
/Ordering  (UCS)
/Supplement  0
>>  def
/CMapName  /Adobe−Identity−UCS  def
/CMapType  2  def
1  begincodespacerange
<0000>    <FFFF>
endcodespacerange
2  beginbfrange
<0000>    <005E>        <0020>
<005F>    <0061>        [<00660066>  <00660069>  <00660066006C>]
endbfrange
1 beginbfchar
<3A51>    <D840DC3E>
endbfchar
endcmap
CMapName  currentdict  /CMap  defineresource  pop
end
end
endstream
endobj
```

<00 00> to <00 5E> are mapped to the Unicode values U+0020 to U+007E This is followed by the definition of a mapping where each character code represents more than one Unicode value:

```
    <005F>  <0061>  [<00660066>  <00660069>  <00660066006C>]
```

In this case, the original character codes are the glyph indices for the ligatures ff, fi, and ffl. The entry defines the mapping from the character codes <00 5F>, <00 60>, and <00 61> to the strings of Unicode values with a Unicode scalar value for each character in the ligature: U+0066 U+0066 are the Unicode values for the character sequence f f, U+0066 U+0069 for f i, and U+0066 U+0066 U+006c for f f l.

Finally, the character code <3A 51> is mapped to the Unicode value U+2003E, which is expressed by the byte sequence <D840DC3E> in UTF-16BE encoding.

EXAMPLE 2 in this sub-clause illustrates several extensions to the way destination values may be defined. To support mappings from a source code to a string of destination codes, this extension has been made to the ranges defined after a **beginbfchar** operator:

> *n* **beginbfchar**
> *srcCode* *dstString*
> **endbfchar**

where *dstString* may be a string of up to 512 bytes. Likewise, mappings after the **beginbfrange** operator may be defined as:

> *n* **beginbfrange**
> $srcCode_1$ $srcCode_2$ *dstString*
> **endbfrange**

In this case, the last byte of the string shall be incremented for each consecutive code in the source code range.

When defining ranges of this type, the value of the last byte in the string shall be less than or equal to 255 – (*srcCode*2 – *srcCode*1). This ensures that the last byte of the string shall not be incremented past 255; otherwise, the result of mapping is undefined.

To support more compact representations of mappings from a range of source character codes to a discontiguous range of destination codes, the CMaps used for the **ToUnicode** entry may use this syntax for the mappings following a **beginbfrange** definition.

> *n* **beginbfrange**
> $srcCode_1$ $srcCode_2$ [$dstString_1$ $dstString_2$ … $dstString_m$]
> **endbfrange**

Consecutive codes starting with *srcCode*1 and ending with *srcCode*2 shall be mapped to the destination strings in the array starting with *dstString*1 and ending with *dstString*m. The value of dstString can be a string of up to 512 bytes. The value of m represents the number of continuous character codes in the source character code range.

$$m = srcCode_2 - srcCode_1 + 1$$

10 Rendering

10.1 General

Nearly all of the rendering facilities that are under the control of a PDF file pertain to the reproduction of colour. Colours shall be rendered by a conforming reader using the following multiple-step process outlined.

NOTE 1 The PDF imaging model separates *graphics* (the specification of shapes and colours) from rendering (controlling a raster output device). Figures 20 and 21 in 8.6.3, "Colour Space Families" illustrate this division. 8, "Graphics" describes the facilities for specifying the appearance of pages in a device-independent way. This clause describes the facilities for controlling how shapes and colours are rendered on the raster output device. All of the facilities discussed here depend on the specific characteristics of the output device. PDF files that are intended to be device-independent should limit themselves to the general graphics facilities described in 8, "Graphics".

Depending on the current colour space and on the characteristics of the device, it is not always necessary to perform every step.

a) If a colour has been specified in a CIE-based colour space (see 8.6.5, "CIE-Based Colour Spaces"), it shall first be transformed to the *native colour space* of the raster output device (also called its *process colour model*).

b) If a colour has been specified in a device colour space that is inappropriate for the output device (for example, *RGB* colour with a *CMYK* or grayscale device), a *colour conversion function* shall be invoked.

c) The device colour values shall now be mapped through *transfer functions*, one for each colour component.

NOTE 2 The transfer functions compensate for peculiarities of the output device, such as nonlinear gray-level response. This step is sometimes called *gamma correction*.

d) If the device cannot reproduce continuous tones, but only certain discrete colours such as black and white pixels, a *halftone function* shall be invoked, which approximates the desired colours by means of patterns of pixels.

e) Finally, *scan conversion* shall be performed to mark the appropriate pixels of the raster output device with the requested colours.

Once these operations have been performed for all graphics objects on the page, the resulting raster data shall be used to mark the physical output medium, such as pixels on a display or ink on a printed page. A PDF file may specify very little about the properties of the physical medium on which the output will be produced; that information may be obtained from the following sources by a conforming reader:

- The media box and a few other entries in the page dictionary (see 14.11.2, "Page Boundaries").
- An interactive dialogue conducted when the user requests viewing or printing.
- A *job ticket*, either embedded in the PDF file or provided separately, that may specify detailed instructions for imposing PDF pages onto media and for controlling special features of the output device. Various standards exist for the format of job tickets. Two of them, JDF (Job Definition Format) and PJTF (Portable Job Ticket Format), are described in the CIP4 document *JDF Specification* and in Adobe Technical Note #5620, *Portable Job Ticket Format* (see the Bibliography), respectively.

Table 58 in 8.4.5, "Graphics State Parameter Dictionaries" lists the various device-dependent graphics state parameters that may be used to control certain aspects of rendering. To invoke these parameters, the **gs** operator shall be used.

10.2 CIE-Based Colour to Device Colour

To render CIE-based colours on an output device, the conforming reader shall convert from the specified CIE-based colour space to the device's native colour space (typically **DeviceGray**, **DeviceRGB**, or **DeviceCMYK**), taking into account the known properties of the device.

NOTE 1 As discussed in 8.6.5, "CIE-Based Colour Spaces" CIE-based colour is based on a model of human colour perception. The goal of CIE-based colour rendering is to produce output in the device's native colour space that accurately reproduces the requested CIE-based colour values as perceived by a human observer. CIE-based colour specification and rendering are a feature of PDF 1.1 (**CalGray**, **CalRGB**, and **Lab**) and PDF 1.3 (ICCBased).

NOTE 2 The conversion from CIE-based colour to device colour is complex, and the theory on which it is based is beyond the scope of this specification. The algorithm has many parameters, including an optional, full three-dimensional colour lookup table. The colour fidelity of the output depends on having these parameters properly set, usually by a method that includes some form of calibration. The colours that a device can produce are characterized by a *device profile*, which is usually specified by an ICC profile associated with the device (and entirely separate from the profile that is specified in an **ICCBased** colour space).

NOTE 3 PDF has no equivalent of the PostScript colour rendering dictionary. The means by which a device profile is associated with a conforming reader's output device are implementation-dependent and not specified in a PDF file. Typically, this is done through a colour management system (CMS) that is provided by the operating system. Beginning with PDF 1.4, a PDF file can also specify one or more output intents providing possible profiles that may be used to process the file (see 14.11.5, "Output Intents").

Conversion from a CIE-based colour value to a device colour value requires two main operations:

a) The CIE-based colour value shall be adjusted according to a *CIE-based gamut mapping function.*

NOTE 4 A *gamut* is a subset of all possible colours in some colour space. A page description has a *source gamut* consisting of all the colours it uses. An output device has a *device gamut* consisting of all the colours it can reproduce. This step transforms colours from the source gamut to the device gamut in a way that attempts to preserve colour appearance, visual contrast, or some other explicitly specified *rendering intent* (see 8.6.5.8, "Rendering Intents").

b) A corresponding device colour value shall be generated according to a *CIE-based colour mapping function.* For a given CIE-based colour value, this function shall compute a colour value in the device's native colour space.

The CIE-based gamut and colour mapping functions shall be applied only to colour values presented in a CIE-based colour space. Colour values in device colour spaces directly control the device colour components though this may be altered by the **DefaultGray**, **DefaultRGB**, and **DefaultCMYK** colour space resources (see 8.6.5.6, "Default Colour Spaces").

The source gamut shall be specified by the information contained in the definition of the CIE-based colour space when selected. This specification shall be device-independent. The corresponding properties of the output device shall be given in the device profile associated with the device. The gamut mapping and colour mapping functions are part of the implementation of the conforming reader.

10.3 Conversions among Device Colour Spaces

10.3.1 General

Each raster output device has a *native colour space*, which typically is one of the standard device colour spaces (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**). In other words, most devices support reproduction of colours according to a grayscale (monochrome), *RGB* (red-green-blue), or *CMYK* (cyan-magenta-yellow-black) model. If the device supports continuous-tone output, reproduction shall occur directly. Otherwise, it shall be accomplished by means of halftoning.

A device’s native colour space is also called its *process colour model*. Process colours are ones that are produced by combinations of one or more standard *process colorants*. Colours specified in any device or CIE-based colour space shall be rendered as process colours. A device may also support additional *spot colorants*, which shall be painted only by means of **Separation** or **DeviceN** colour spaces. They shall not be involved in the rendering of device or CIE-based colour spaces, nor shall they be subject to the conversions described in the Note.

NOTE Some devices provide a native colour space that is not one of the three named previously but consists of a different combination of colorants. In that case, conversion from the standard device colour spaces to the device’s native colour space may be performed by the conforming reader in a manner of its own choosing.

Knowing the native colour space and other output capabilities of the device, the conforming reader shall automatically convert the colour values specified in a file to those appropriate for the device’s native colour space. If the file specifies colours directly in the device’s native colour space, no conversions shall be performed.

EXAMPLE If a file specifies colours in the **DeviceRGB** colour space but the device supports grayscale (such as a monochrome display) or *CMYK* (such as a colour printer), the conforming reader shall perform the necessary conversions.

The algorithms used to convert among device colour spaces are very simple. As perceived by a human viewer, these conversions produce only crude approximations of the original colours. More sophisticated control over colour conversion may be achieved by means of CIE-based colour specification and rendering. Additionally, device colour spaces may be remapped into CIE-based colour spaces (see 8.6.5.6, "Default Colour Spaces").

10.3.2 Conversion between DeviceGray and DeviceRGB

Black, white, and intermediate shades of gray can be considered special cases of *RGB* colour. A grayscale value shall be described by a single number: 0.0 corresponds to black, 1.0 to white, and intermediate values to different gray levels.

A gray level shall be equivalent to an *RGB* value with all three components the same. In other words, the *RGB* colour value equivalent to a specific gray value shall be

$$\begin{aligned} red &= gray \\ green &= gray \\ blue &= gray \end{aligned}$$

The gray value for a given *RGB* value shall be computed according to the NTSC video standard, which determines how a colour television signal is rendered on a black-and-white television set:

$$gray = 0.3 \times red + 0.59 \times green + 0.11 \times blu$$

10.3.3 Conversion between DeviceGray and DeviceCMYK

Nominally, a gray level is the complement of the black component of *CMYK*. Therefore, the *CMYK* colour value equivalent to a specific gray level shall be

$$\begin{aligned} cyan &= 0.0 \\ magenta &= 0.0 \\ yellow &= 0.0 \\ black &= 1.0 - gray \end{aligned}$$

To obtain the equivalent gray level for a given *CMYK* value, the contributions of all components shall be taken into account:

$$gray = 1.0 - \min(1.0, 0.3 \times cyan + 0.59 \times magenta + 0.11 \times yellow + black$$

The interactions between the black component and the other three are elaborated in 10.3.4.

10.3.4 Conversion from DeviceRGB to DeviceCMYK

Conversion of a colour value from *RGB* to *CMYK* is a two-step process. The first step shall be to convert the red-green-blue value to equivalent cyan, magenta, and yellow components. The second step shall be to generate a black component and alter the other components to produce a better approximation of the original colour.

NOTE 1 The subtractive colour primaries cyan, magenta, and yellow are the complements of the additive primaries red, green, and blue.

EXAMPLE A cyan ink subtracts the red component of white light. In theory, the conversion is very simple:

$$\begin{aligned} cyan &= 1.0 - red \\ magenta &= 1.0 - green \\ yellow &= 1.0 - blue \end{aligned}$$

A colour that is 0.2 red, 0.7 green, and 0.4 blue can also be expressed as 1.0 − 0.2 = 0.8 cyan, 1.0 − 0.7 = 0.3 magenta, and 1.0 − 0.4 = 0.6 yellow.

NOTE 2 Logically, only cyan, magenta, and yellow are needed to generate a printing colour. An equal level of cyan, magenta, and yellow should create the equivalent level of black. In practice, however, coloured printing inks do not mix perfectly; such combinations often form dark brown shades instead of true black. To obtain a truer colour rendition on a printer, true black ink is often substituted for the mixed-black portion of a colour. Most colour printers support a black component (the *K* component of *CMYK*). Computing the quantity of this component requires some additional steps:

Black generation calculates the amount of black to be used when trying to reproduce a particular colour.

Undercolor removal reduces the amounts of the cyan, magenta, and yellow components to compensate for the amount of black that was added by black generation.

The complete conversion from *RGB* to *CMYK* shall be as follows, where *BG*(*k*) and *UCR*(*k*) are invocations of the black-generation and undercolor-removal functions, respectively:

$$\begin{aligned} c &= 1.0 - red \\ m &= 1.0 - green \\ y &= 1.0 - blue \\ k &= \min(c, m, y) \end{aligned}$$

$$\begin{aligned} cyan &= \min(1.0, \max(0.0, c - UCR(k))) \\ magenta &= \min(1.0, \max(0.0, m - UCR(k))) \\ yellow &= \min(1.0, \max(0.0, y - UCR(k))) \\ black &= \min(1.0, \max(0.0, BG(k))) \end{aligned}$$

The black-generation and undercolor-removal functions shall be defined as PDF function dictionaries (see 7.10, "Functions") that are parameters in the graphics state. They shall be specified as the values of the **BG** and **UCR** (or **BG2** and **UCR2**) entries in a graphics state parameter dictionary (see Table 58). Each function shall be called with a single numeric operand and shall return a single numeric result.

The input of both the black-generation and undercolor-removal functions shall be *k*, the minimum of the intermediate *c*, *m*, and *y* values that have been computed by subtracting the original *red*, *green*, and *blue* components from 1.0.

NOTE 3 Nominally, *k* is the amount of black that can be removed from the cyan, magenta, and yellow components and substituted as a separate black component.

The black-generation function shall compute the black component as a function of the nominal *k* value. It may simply return its *k* operand unchanged, or it may return a larger value for extra black, a smaller value for less black, or 0.0 for no black at all.

The undercolor-removal function shall compute the amount to subtract from each of the intermediate *c*, *m*, and *y* values to produce the final cyan, magenta, and yellow components. It may simply return its *k* operand unchanged, or it may return 0.0 (so that no colour is removed), some fraction of the black amount, or even a negative amount, thereby adding to the total amount of colorant.

The final component values that result after applying black generation and undercolor removal should be in the range 0.0 to 1.0. If a value falls outside this range, the nearest valid value shall be substituted automatically without error indication.

NOTE 4 This substitution is indicated explicitly by the *min* and *max* operations in the preceding formulas.

The correct choice of black-generation and undercolor-removal functions depends on the characteristics of the output device. Each device shall be configured with default values that are appropriate for that device.

NOTE 5 See 11.7.5, "Rendering Parameters and Transparency" and, in particular, 11.7.5.3, "Rendering Intent and Colour Conversions" for further discussion of the role of black-generation and undercolor-removal functions in the transparent imaging model.

10.3.5 Conversion from DeviceCMYK to DeviceRGB

Conversion of a colour value from *CMYK* to *RGB* is a simple operation that does not involve black generation or undercolour removal:

$$\begin{aligned} red &= 1.0 - \min(1.0, cyan + black) \\ green &= 1.0 - \min(1.0, magenta + black) \\ blue &= 1.0 - \min(1.0, yellow + black) \end{aligned}$$

The black component shall be added to each of the other components, which shall then be converted to their complementary colours by subtracting them each from 1.0.

10.4 Transfer Functions

In the sequence of steps for processing colours, the conforming reader shall apply the transfer function *after* performing any needed conversions between colour spaces, but *before* applying a halftone function, if necessary. Each colour component shall have its own separate transfer function; there shall not be interaction between components.

NOTE 1 Starting with PDF 1.2, a *transfer function* may be used to adjust the values of colour components to compensate for nonlinear response in an output device and in the human eye. Each component of a device colour space—for example, the red component of the **DeviceRGB** space—is intended to represent the perceived lightness or intensity of that colour component in proportion to the component’s numeric value.

NOTE 2 Many devices do not actually behave this way, however; the purpose of a transfer function is to compensate for the device’s actual behaviour. This operation is sometimes called *gamma correction* (not to be confused with the *CIE-based gamut mapping function* performed as part of CIE-based colour rendering).

Transfer functions shall always operate in the native colour space of the output device, regardless of the colour space in which colours were originally specified. (For example, for a *CMYK* device, the transfer functions apply to the device’s cyan, magenta, yellow, and black colour components, even if the colours were originally specified in, for example, a **DeviceRGB** or **CalRGB** colour space.) The transfer function shall be called with a numeric operand in the range 0.0 to 1.0 and shall return a number in the same range. The input shall be the value of a colour component in the device’s native colour space, either specified directly or produced by conversion from some other colour space. The output shall be the transformed component value to be transmitted to the device (after halftoning, if necessary).

Both the input and the output of a transfer function shall always be interpreted as if the corresponding colour component were additive (red, green, blue, or gray): the greater the numeric value, the lighter the colour. If the component is subtractive (cyan, magenta, yellow, black, or a spot colour), it shall be converted to additive form by subtracting it from 1.0 before it is passed to the transfer function. The output of the function shall always be in additive form and shall be passed on to the halftone function in that form.

Starting with PDF 1.2, transfer functions shall be defined as PDF function objects (see 7.10, "Functions"). There are two ways to specify transfer functions:

- The *current transfer function* parameter in the graphics state shall consist of either a single transfer function or an array of four separate transfer functions, one each for red, green, blue, and gray or their complements cyan, magenta, yellow, and black. If only a single function is specified, it shall apply to all components. An *RGB* device shall use the first three, a monochrome device shall use the gray transfer function only, and a *CMYK* device shall use all four. The current transfer function may be specified as the value of the **TR** or **TR2** entry in a graphics state parameter dictionary; see Table 58.

- The *current halftone* parameter in the graphics state may specify transfer functions as optional entries in *halftone dictionaries* (see 10.5.5, "Halftone Dictionaries"). This is the only way to set transfer functions for nonprimary colour components or for any component in devices whose native colour space uses components other than the ones listed previously. A transfer function specified in a halftone dictionary shall override the corresponding one specified by the current transfer function parameter in the graphics state.

In addition to their intended use for gamma correction, transfer functions may be used to produce a variety of special, device-dependent effects. Because transfer functions produce device-dependent effects, a page description that is intended to be device-independent shall not alter them.

When the current colour space is DeviceGray and the output device's native colour space is DeviceCMYK, a conforming reader shall use only the gray transfer function. The normal conversion from DeviceGray to DeviceCMYK produces 0.0 for the cyan, magenta, and yellow components. These components shall not be passed through their respective transfer functions but are rendered directly, producing output containing no coloured inks. This special case exists for compatibility with existing conforming readers that use a transfer function to obtain special effects on monochrome devices, and shall apply only to colours specified in the DeviceGray colour space.

NOTE 3 See 11.7.5, "Rendering Parameters and Transparency" and, in particular, 11.7.5.2, "Halftone and Transfer Function" for further discussion of the role of transfer functions in the transparent imaging model.

10.5 Halftones

10.5.1 General

Halftoning is a process by which continuous-tone colours are approximated on an output device that can achieve only a limited number of discrete colours. Colours that the device cannot produce directly are simulated by using patterns of pixels in the colours available.

NOTE 1 Perhaps the most familiar example is the rendering of gray tones with black and white pixels, as in a newspaper photograph.

Some output devices can reproduce continuous-tone colours directly. Halftoning is not required for such devices; after gamma correction by the transfer functions, the colour components shall be transmitted directly to the device. On devices that do require halftoning, it shall occur after all colour components have been transformed by the applicable transfer functions. The input to the halftone function shall consist of continuous-tone, gamma-corrected colour components in the device's native colour space. Its output shall consist of pixels in colours the device can reproduce.

PDF provides a high degree of control over details of the halftoning process.

NOTE 2 When rendering on low-resolution displays, fine control over halftone patterns is needed to achieve the best approximations of gray levels or colours and to minimize visual artifacts.

NOTE 3 In colour printing, independent halftone screens can be specified for each of several colorants.

NOTE 4 Remember that everything pertaining to halftones is, by definition, device-dependent. In general, when a PDF file provides its own halftone specifications, it sacrifices portability. Associated with every output device is a default halftone definition that is appropriate for most purposes. Only relatively sophisticated files need to define their own halftones to achieve special effects. For correct results, a PDF file that defines a new halftone depends on certain assumptions about the resolution and orientation of device space. The best choice of halftone parameters often depends on specific physical properties of the output device, such as pixel shape, overlap between pixels, and the effects of electronic or mechanical noise.

All halftones are defined in device space, and shall be unaffected by the current transformation matrix.

10.5.2 Halftone Screens

In general, halftoning methods are based on the notion of a *halftone screen*, which divides the array of device pixels into *cells* that may be modified to produce the desired halftone effects. A screen is defined by conceptually laying a uniform rectangular grid over the device pixel array. Each pixel belongs to one cell of the grid; a single cell typically contains many pixels. The screen grid shall be defined entirely in device space and shall be unaffected by modifications to the current transformation matrix.

NOTE This property is essential to ensure that adjacent areas coloured by halftones are properly stitched together without visible seams.

On a bilevel (black-and-white) device, each cell of a screen may be made to approximate a shade of gray by painting some of the cell's pixels black and some white. Numerically, the gray level produced within a cell shall be the ratio of white pixels to the total number of pixels in the cell. A cell containing n pixels can render $n + 1$ different gray levels, ranging from all pixels black to all pixels white. A gray value g in the range 0.0 to 1.0 shall be produced by making i pixels white, where $i = \text{floor}(g \times n)$.

The foregoing description also applies to colour output devices whose pixels consist of primary colours that are either completely on or completely off. Most colour printers, but not colour displays, work this way. Halftoning shall be applied to each colour component independently, producing shades of that colour.

Colour components shall be presented to the halftoning machinery in additive form, regardless of whether they were originally specified additively (*RGB* or gray) or subtractively (*CMYK* or tint). Larger values of a colour component represent lighter colours—greater intensity in an additive device such as a display or less ink in a subtractive device such as a printer. Transfer functions produce colour values in additive form; see 10.4, "Transfer Functions".

10.5.3 Spot Functions

A common way of defining a halftone screen is by specifying a *frequency*, *angle*, and *spot function*. The frequency indicates the number of halftone cells per inch; the angle indicates the orientation of the grid lines relative to the device coordinate system. As a cell's desired gray level varies from black to white, individual pixels within the cell change from black to white in a well-defined sequence: if a particular gray level includes certain white pixels, lighter grays will include the same white pixels along with some additional ones. The order in which pixels change from black to white for increasing gray levels shall be determined by a *spot function*, which specifies that order in an indirect way that minimizes interactions with the screen frequency and angle.

Consider a halftone cell to have its own coordinate system: the centre of the cell is the origin and the corners are at coordinates ± 1.0 horizontally and vertically. Each pixel in the cell is centred at horizontal and vertical coordinates that both lie in the range -1.0 to $+1.0$. For each pixel, the spot function shall be invoked with the pixel's coordinates as input and shall return a single number in the range -1.0 to $+1.0$, defining the pixel's position in the whitening order.

The specific values the spot function returns are not significant; all that matters are the *relative* values returned for different pixels. As a cell's gray level varies from black to white, the first pixel whitened shall be the one for which the spot function returns the lowest value, the next pixel shall be the one with the next higher spot function value, and so on. If two pixels have the same spot function value, their relative order shall be chosen arbitrarily.

PDF provides built-in definitions for many of the most commonly used spot functions that a conforming reader shall implement. A halftone may simply specify any of these predefined spot functions by name instead of giving an explicit function definition.

EXAMPLE The name **SimpleDot** designates a spot function whose value is inversely related to a pixel’s distance from the center of the halftone cell. This produces a “dot screen” in which the black pixels are clustered within a circle whose area is inversely proportional to the gray level. The name **Line** designates a spot function whose value is the distance from a given pixel to a line through the center of the cell, producing a “line screen” in which the white pixels grow away from that line.

Table 128 shows the predefined spot functions. The table gives the mathematical definition of each function along with the corresponding PostScript language code as it would be defined in a PostScript calculator function (see 7.10.5, "Type 4 (PostScript Calculator) Functions"). The image accompanying each function shows how the relative values of the function are distributed over the halftone cell, indicating the approximate order in which pixels are whitened. Pixels corresponding to darker points in the image are whitened later than those corresponding to lighter points.

Table 128 – Predefined spot functions

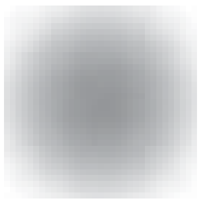
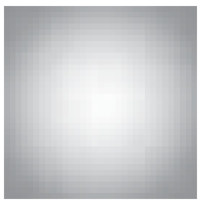
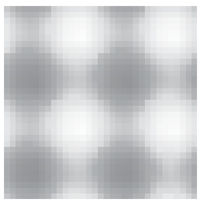
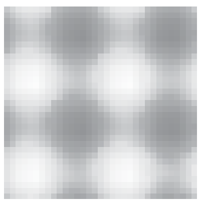
Name	Appearance	Definition
SimpleDot		$1 - (x^2 + y^2)$ { dup mul exch dup mul add 1 exch sub }
InvertedSimpleDot		$x^2 + y^2 - 1$ { dup mul exch dup mul add 1 sub }
DoubleDot		$\frac{\sin(360 \times x)}{2} + \frac{\sin(360 \times y)}{2}$ { 360 mul sin 2 div exch 360 mul sin 2 div add }
InvertedDoubleDot		$-\left(\frac{\sin(360 \times x)}{2} + \frac{\sin(360 \times y)}{2}\right)$ { 360 mul sin 2 div exch 360 mul sin 2 div add neg }

Table 128 – Predefined spot functions (continued)

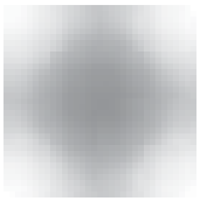
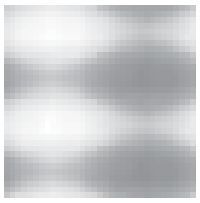
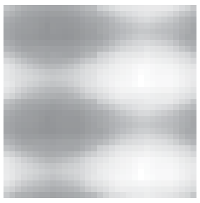
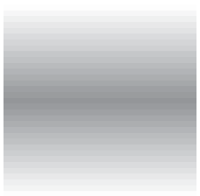
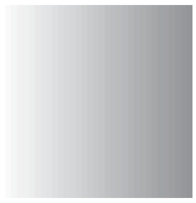
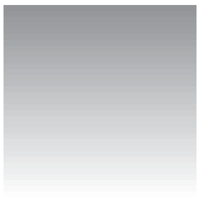
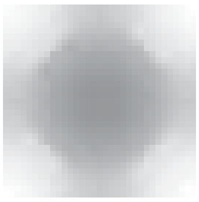
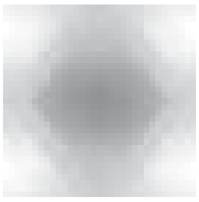
Name	Appearance	Definition
CosineDot		$\frac{\cos(180 \times x)}{2} + \frac{\cos(180 \times y)}{2}$ { 180 mul cos exch 180 mul cos add 2 div }
Double		$\frac{\sin\left(360 \times \frac{x}{2}\right)}{2} + \frac{\sin(360 \times y)}{2}$ { 360 mul sin 2 div exch 2 div 360 mul sin 2 div add }
InvertedDouble		$-\left(\frac{\sin\left(360 \times \frac{x}{2}\right)}{2} + \frac{\sin(360 \times y)}{2}\right)$ { 360 mul sin 2 div exch 2 div 360 mul sin 2 div add neg }
Line		$-\|y\|$ { exch pop abs neg }
LineX		*x* { pop }
LineY		*y* { exch pop }

Table 128 – Predefined spot functions (continued)

Name	Appearance	Definition

Round		*if* $\lvert x\rvert + \lvert y\rvert \le 1$ *then* $1 - (x^2 + y^2)$
		else $(\lvert x\rvert - 1)^2 + (\lvert y\rvert - 1)^2 - 1$
		`{ abs exch abs`
		`  2 copy add 1 le`
		`    { dup mul exch dup mul add 1 exch sub }`
		`    { 1 sub dup mul exch 1 sub dup mul add 1 sub }`
		`  ifelse }`

Ellipse		*let* $w = (3 \times \lvert x\rvert) + (4 \times \lvert y\rvert) - 3$
		if $w < 0$ *then* $1 - \frac{x^2 + \left(\frac{\lvert y\rvert}{0.75}\right)^2}{4}$
		else if $w > 1$ *then* $\frac{(1 - \lvert x\rvert)^2 + \left(\frac{1 - \lvert y\rvert}{0.75}\right)^2}{4} - 1$
		else $0.5 - w$
		`{ abs exch abs 2 copy 3 mul exch 4 mul add 3 sub dup 0 lt`
		`    { pop dup mul exch 0.75 div dup mul add`
		`      4 div 1 exch sub }`
		`    { dup 1 gt`
		`        { pop 1 exch sub dup mul`
		`          exch 1 exch sub 0.75 div dup mul add`
		`          4 div 1 sub }`
		`        { 0.5 exch sub exch pop exch pop }`
		`      ifelse }`
		`  ifelse }`

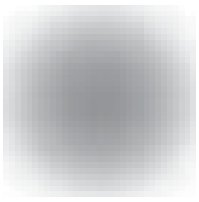
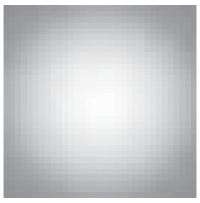
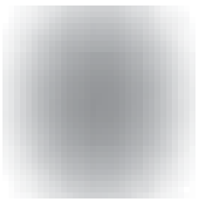
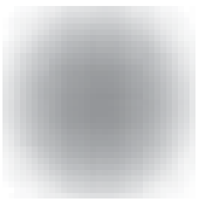
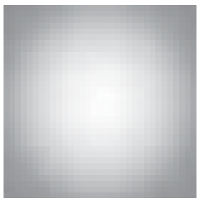
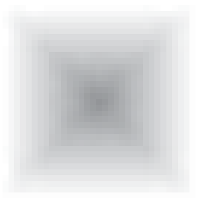
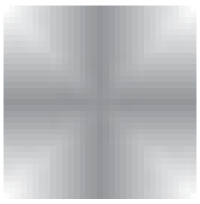
EllipseA		$1 - (x^2 + 0.9 \times y^2)$ `{ dup mul 0.9 mul exch dup mul add 1 exch sub }`
InvertedEllipseA		$x^2 + 0.9 \times y^2 - 1$ `{ dup mul 0.9 mul exch dup mul add 1 sub }`

Table 128 – Predefined spot functions (continued)

Name	Appearance	Definition
EllipseB		$1 - \sqrt{x^2 + \frac{5}{8} \times y^2}$ { dup 5 mul 8 div mul exch dup mul exch add sqrt 1 exch sub }
EllipseC		$1 - (0.9 \times x^2 + y^2)$ { dup mul exch dup mul 0.9 mul add 1 exch sub }
InvertedEllipseC		$0.9 \times x^2 + y^2 - 1$ { dup mul exch dup mul 0.9 mul add 1 sub }

Square		$-\max(\lvert x\rvert, \lvert y\rvert)$
		 { abs exch abs 2 copy lt
		 { exch }
		 if
		 pop neg }

Cross		$-\min(\lvert x\rvert, \lvert y\rvert)$
		 { abs exch abs 2 copy gt
		 { exch }
		 if
		 pop neg }

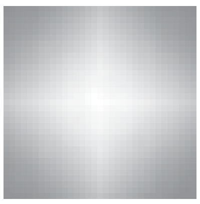
Rhomboid		$\frac{0.9 \times \lvert x\rvert + \lvert y\rvert}{2}$ { abs exch abs 0.9 mul add 2 div }

Table 128 – Predefined spot functions (continued)

Name	Appearance	Definition

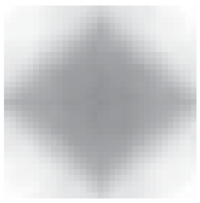
Diamond		*if* $\lvert x\rvert + \lvert y\rvert \le 0.75$ *then* $1 - (x^2 + y^2)$
		else if $\lvert x\rvert + \lvert y\rvert \le 1.23$ *then* $1 - (0.85 \times \lvert x\rvert + \lvert y\rvert)$
		else $(\lvert x\rvert - 1)^2 + (\lvert y\rvert - 1)^2 - 1$
		`{ abs exch abs 2 copy add 0.75 le`
		`    { dup mul exch dup mul add 1 exch sub }`
		`    { 2 copy add 1.23 le`
		`        { 0.85 mul add 1 exch sub }`
		`        { 1 sub dup mul exch 1 sub du`
		`        mul add 1 sub }`
		`      ifelse }`
		`  ifelse }`

Figure 49 illustrates the effects of some of the predefined spot functions.

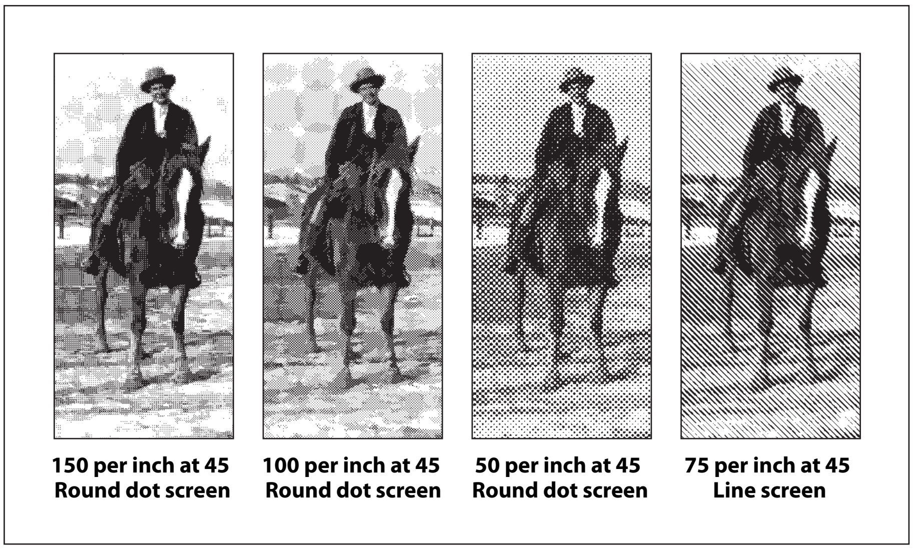


Figure 49 – Various halftoning effects

10.5.4 Threshold Arrays

Another way to define a halftone screen is with a *threshold array* that directly controls individual device pixels in a halftone cell. This technique provides a high degree of control over halftone rendering. It also permits halftone cells to be arbitrary rectangles, whereas those controlled by a spot function are always square.

A threshold array is much like a sampled image—a rectangular array of pixel values—but shall be defined entirely in device space. Depending on the halftone type, the threshold values occupy 8 or 16 bits each. Threshold values nominally represent gray levels in the usual way, from 0 for black up to the maximum (255 or 65,535) for white. The threshold array shall be replicated to tile the entire device space: each pixel in device space shall be mapped to a particular sample in the threshold array. On a bilevel device, where each pixel is either black or white, halftoning with a threshold array shall proceed as follows:

a) For each device pixel that is to be painted with some gray level, consult the corresponding threshold value from the threshold array.

b) If the requested gray level is less than the threshold value, paint the device pixel black; otherwise, paint it white. Gray levels in the range 0.0 to 1.0 correspond to threshold values from 0 to the maximum available (255 or 65,535).

A threshold value of 0 shall be treated as if it were 1; therefore, a gray level of 0.0 paints all pixels black, regardless of the values in the threshold array.

This scheme easily generalizes to monochrome devices with multiple bits per pixel, where each pixel can directly represent intermediate gray levels in addition to black and white. For any device pixel that is specified with some in-between gray level, the halftoning algorithm shall consult the corresponding value in the threshold array to determine whether to use the next-lower or next-higher representable gray level. In this situation, the threshold values do not represent absolute gray levels, but rather gradations between any two adjacent representable gray levels.

EXAMPLE If there are 2 bits per pixel, each pixel can directly represent one of four different gray levels: black, dark gray, light gray, or white, encoded as 0, 1, 2, and 3, respectively.

NOTE A halftone defined in this way can also be used with colour displays that have a limited number of values for each colour component. The red, green, and blue components are simply treated independently as gray levels, applying the appropriate threshold array to each. (This technique also works for a screen defined as a spot function, since the spot function is used to compute a threshold array internally.)

10.5.5 Halftone Dictionaries

10.5.5.1 General

In PDF 1.2, the graphics state includes a *current halftone* parameter, which determines the halftoning process that a conforming reader shall use to perform painting operations. The current halftone may be specified as the value of the **HT** entry in a graphics state parameter dictionary; see Table 58. It may be defined by either a dictionary or a stream, depending on the type of halftone; the term *halftone dictionary* is used generically throughout this clause to refer to either a dictionary object or the dictionary portion of a stream object. (The halftones that are defined by streams are specifically identified as such in the descriptions of particular halftone types; unless otherwise stated, they are understood to be defined by simple dictionaries instead.)

Every halftone dictionary shall have a **HalftoneType** entry whose value shall be an integer specifying the overall type of halftone definition. The remaining entries in the dictionary are interpreted according to this type. PDF supports the halftone types listed in Table 129.

Table 129 – PDF halftone types

Type	Meaning
1	Defines a single halftone screen by a *frequency, angle*, and *spot function*.
5	Defines an arbitrary number of halftone screens, one for each colorant or colour component (including both primary and spot colorants). The keys in this dictionary are names of colorants; the values are halftone dictionaries of other types, each defining the halftone screen for a single colorant.
6	Defines a single halftone screen by a threshold array containing 8-bit sample values.
10	Defines a single halftone screen by a threshold array containing 8-bit sample values, representing a halftone cell that may have a nonzero screen angle.
16	*(PDF 1.3)* Defines a single halftone screen by a threshold array containing 16-bit sample values, representing a halftone cell that may have a nonzero screen angle.

NOTE 1 The dictionaries representing these halftone types contain the same entries as the corresponding PostScript language halftone dictionaries (as described in Section 7.4 of the PostScript Language Reference, Third Edition), with the following exceptions:

The PDF dictionaries may contain a **Type** entry with the value *Halftone*, identifying the type of PDF object that the dictionary describes.

Spot functions and transfer functions are represented by function objects instead of PostScript procedures.

Threshold arrays are specified as streams instead of files.

In type 5 halftone dictionaries, the keys for colorants shall be name objects; they may not be strings as they may in PostScript.

Halftone dictionaries have an optional entry, **HalftoneName**, that identifies the halftone by name. In PDF 1.3, if this entry is present, all other entries, including **HalftoneType**, are optional. At rendering time, if the output device has a halftone with the specified name, that halftone shall be used, overriding any other halftone parameters specified in the dictionary.

NOTE 2 This provides a way for PDF files to select the proprietary halftones supplied by some device manufacturers, which would not otherwise be accessible because they are not explicitly defined in PDF.

If there is no HalftoneName entry, or if the requested halftone name does not exist on the device, the halftone's parameters shall be defined by the other entries in the dictionary, if any. If no other entries are present, the default halftone shall be used.

NOTE 3 See 11.7.5, "Rendering Parameters and Transparency" and, in particular, “Halftone and Transfer Function” in 11.7.5.2 for further discussion of the role of halftones in the transparent imaging model.

10.5.5.2 Type 1 Halftones

Table 130 describes the contents of a halftone dictionary of type 1, which defines a halftone screen in terms of its frequency, angle, and spot function.

Table 130 – Entries in a type 1 halftone dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Halftone** for a halftone dictionary.
HalftoneType	integer	*(Required)* A code identifying the halftone type that this dictionary describes; shall be 1 for this type of halftone.
HalftoneName	byte string	*(Optional)* The name of the halftone dictionary.
Frequency	number	*(Required)* The screen frequency, measured in halftone cells per inch in device space.
Angle	number	*(Required)* The screen angle, in degrees of rotation counterclockwise with respect to the device coordinate system. NOTE Most output devices have left-handed device spaces. On such devices, a counterclockwise angle in device space corresponds to a clockwise angle in default user space and on the physical medium.
SpotFunction	function or name	*(Required)* A function object defining the order in which device pixels within a screen cell shall be adjusted for different gray levels, or the name of one of the predefined spot functions (see Table 128).
AccurateScreens	boolean	*(Optional)* A flag specifying whether to invoke a special halftone algorithm that is extremely precise but computationally expensive; see Note 1 for further discussion. Default value: **false**.

Table 130 – Entries in a type 1 halftone dictionary (continued)

Key	Type	Value
TransferFunction	function or name	*(Optional)* A transfer function, which overrides the current transfer function in the graphics state for the same component. This entry shall be present if the dictionary is a component of a type 5 halftone (see "Type 5 Halftones" in 10.5.5.6) and represents either a nonprimary or nonstandard primary colour component (see 10.4, "Transfer Functions"). The name **Identity** may be used to specify the identity function.

If the **AccurateScreens** entry has a value of **true**, a highly precise halftoning algorithm shall be substituted in place of the standard one. If **AccurateScreens** is **false** or not present, ordinary halftoning shall be used.

NOTE 1 Accurate halftoning achieves the requested screen frequency and angle with very high accuracy, whereas ordinary halftoning adjusts them so that a single screen cell is quantized to device pixels. High accuracy is important mainly for making colour separations on high-resolution devices. However, it may be computationally expensive and therefore is ordinarily disabled.

NOTE 2 In principle, PDF permits the use of halftone screens with arbitrarily large cells—in other words, arbitrarily low frequencies. However, cells that are very large relative to the device resolution or that are oriented at unfavorable angles may exceed the capacity of available memory. If this happens, an error occurs. The **AccurateScreens** feature often requires very large amounts of memory to achieve the highest accuracy.

EXAMPLE The following shows a halftone dictionary for a type 1 halftone.

```
28 0 obj
   << /Type /Halftone
        /HalftoneType 1
        /Frequency 120
        /Angle 30
        /SpotFunction /CosineDot
        /TransferFunction /Identity
   >>
endobj
```

10.5.5.3 Type 6 Halftones

A type 6 halftone defines a halftone screen with a threshold array. The halftone shall be represented as a stream containing the threshold values; the parameters defining the halftone shall be specified by entries in the stream dictionary. This dictionary may contain the entries shown in Table 131 in addition to the usual entries common to all streams (see Table 5). The **Width** and **Height** entries shall specify the dimensions of the threshold array in device pixels; the stream shall contain **Width** × **Height** bytes, each representing a single threshold value. Threshold values are defined in device space in the same order as image samples in image space (see Figure 34), with the first value at device coordinates (0, 0) and horizontal coordinates changing faster than vertical coordinates.

Table 131 – Additional entries specific to a type 6 halftone dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Halftone** for a halftone dictionary.
HalftoneType	integer	*(Required)* A code identifying the halftone type that this dictionary describes; shall be 6 for this type of halftone.
HalftoneName	byte string	*(Optional)* The name of the halftone dictionary.
Width	integer	*(Required)* The width of the threshold array, in device pixels.
Height	integer	*(Required)* The height of the threshold array, in device pixels.

Table 131 – Additional entries specific to a type 6 halftone dictionary (continued)

Key	Type	Value
TransferFunction	function or name	*(Optional)* A transfer function, which shall override the current transfer function in the graphics state for the same component. This entry shall be present if the dictionary is a component of a type 5 halftone (see “Type 5 Halftones” in 10.5.5.6) and represents either a nonprimary or nonstandard primary colour component (see 10.4, "Transfer Functions"). The name **Identity** may be used to specify the identity function.

10.5.5.4 Type 10 Halftones

Type 6 halftones specify a threshold array with a zero screen angle; they make no provision for other angles. The type 10 halftone removes this restriction and allows the use of threshold arrays for halftones with nonzero screen angles as well.

Halftone cells at nonzero angles can be difficult to specify because they may not line up well with scan lines and because it may be difficult to determine where a given sampled point goes. The type 10 halftone addresses these difficulties by dividing the halftone cell into a pair of squares that line up at zero angles with the output device’s pixel grid. The squares contain the same information as the original cell but are much easier to store and manipulate. In addition, they can be mapped easily into the internal representation used for all rendering.

NOTE 1 Figure 50 shows a halftone cell with a frequency of 38.4 cells per inch and an angle of 50.2 degrees, represented graphically in device space at a resolution of 300 dots per inch. Each asterisk in the figure represents a location in device space that is mapped to a specific location in the threshold array.

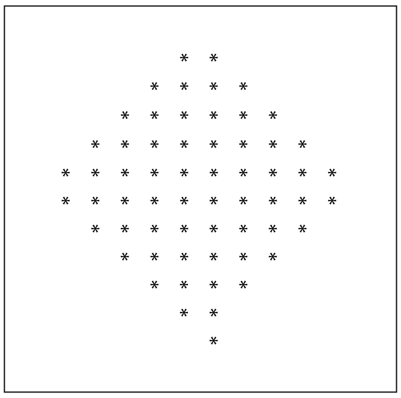


Figure 50 – Halftone cell with a nonzero angle

NOTE 2 Figure 51 shows how the halftone cell can be divided into two squares. If the squares and the original cell are tiled across device space, the area to the right of the upper square maps exactly into the empty area of the lower square, and vice versa (see Figure 52). The last row in the first square is immediately adjacent to the first row in the second square and starts in the same column.

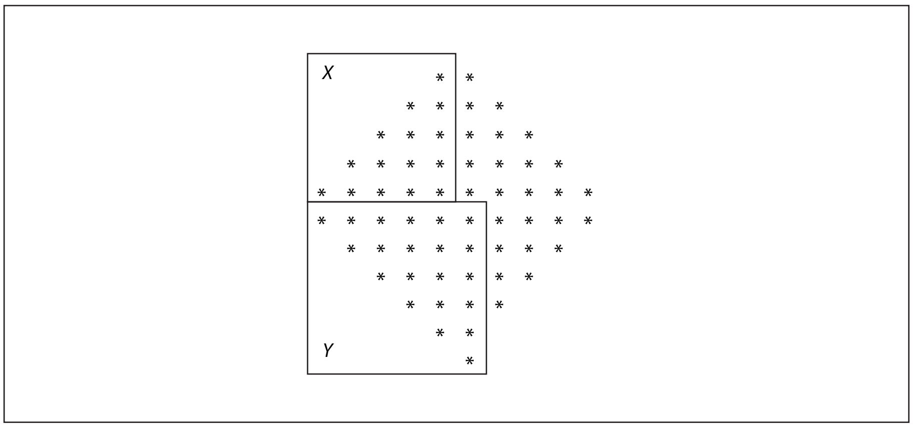


Figure 51 – Angled halftone cell divided into two squares

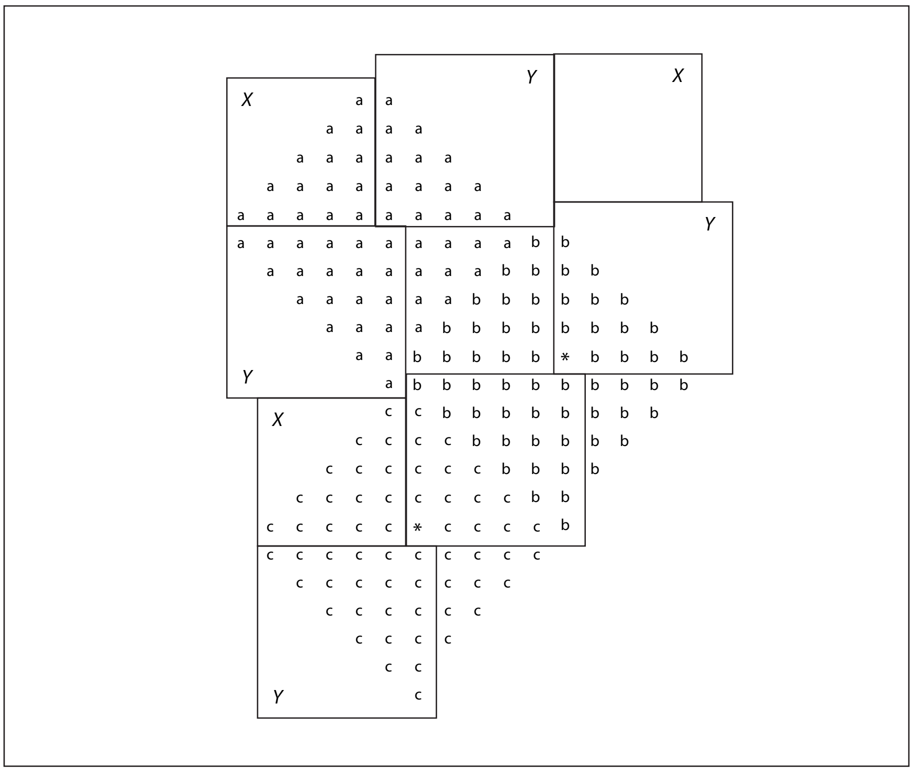


Figure 52 – Halftone cell and two squares tiled across device space

NOTE 3 Any halftone cell can be divided in this way. The side of the upper square (X) is equal to the horizontal displacement from a point in one halftone cell to the corresponding point in the adjacent cell, such as those marked by asterisks in Figure 52. The side of the lower square (*Y*) is the vertical displacement between the same two points. The frequency of a halftone screen constructed from squares with sides *X* and *Y* is thus given by

$$frequency = \frac{resolution}{\sqrt{X^2 + Y^2}}$$

and the angle by

$$angle = \operatorname{atan}\left(\frac{Y}{X}\right)$$

Like a type 6 halftone, a type 10 halftone shall be represented as a stream containing the threshold values, with the parameters defining the halftone specified by entries in the stream dictionary. This dictionary may contain the entries shown in Table 132 in addition to the usual entries common to all streams (see Table 5). The **Xsquare** and **Ysquare** entries replace the type 6 halftone's **Width** and **Height** entries.

Table 132 – Additional entries specific to a type 10 halftone dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Halftone** for a halftone dictionary.
HalftoneType	integer	*(Required)* A code identifying the halftone type that this dictionary describes; shall be 10 for this type of halftone.
HalftoneName	byte string	*(Optional)* The name of the halftone dictionary.
Xsquare	integer	*(Required)* The side of square *X,* in device pixels; see below.
Ysquare	integer	*(Required)* The side of square *Y*, in device pixels; see below.
TransferFunction	function or name	*(Optional)* A transfer function, which shall override the current transfer function in the graphics state for the same component. This entry shall be present if the dictionary is a component of a type 5 halftone (see "Type 5 Halftones" in 10.5.5.6) and represents either a nonprimary or nonstandard primary colour component (see 10.4, "Transfer Functions"). The name **Identity** may be used to specify the identity function.

The **Xsquare** and **Ysquare** entries shall specify the dimensions of the two squares in device pixels. The stream shall contain **Xsquare**2 + **Ysquare**2 bytes, each representing a single threshold value. The contents of square *X* shall be specified first, followed by those of square *Y*. Threshold values within each square shall be defined in device space in the same order as image samples in image space (see Figure 34), with the first value at device coordinates (0, 0) and horizontal coordinates changing faster than vertical coordinates.

10.5.5.5 Type 16 Halftones

Like type 10, a type 16 halftone *(PDF 1.3)* defines a halftone screen with a threshold array and allows nonzero screen angles. In type 16, however, each element of the threshold array shall be 16 bits wide instead of 8. This allows the threshold array to distinguish 65,536 levels of colour rather than only 256 levels. The threshold array may consist of either one rectangle or two rectangles. If two rectangles are specified, they shall tile the device space as shown in Figure 53. The last row in the first rectangle shall be immediately adjacent to the first row in the second and shall start in the same column.

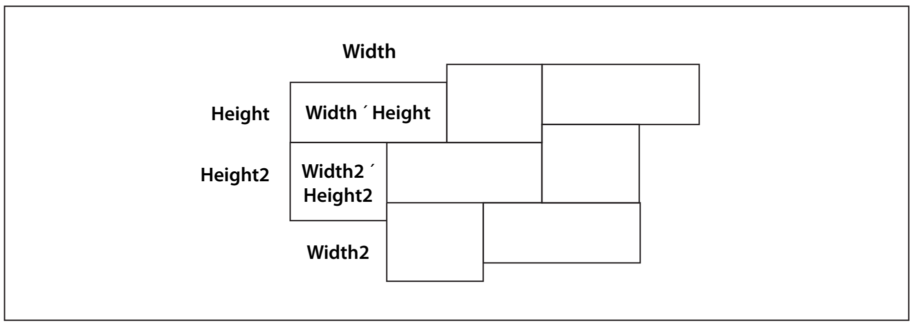


Figure 53 – Tiling of device space in a type 16 halftone

A type 16 halftone, like type 6 and type 10, shall be represented as a stream containing the threshold values, with the parameters defining the halftone specified by entries in the stream dictionary. This dictionary may contain the entries shown in Table 133 in addition to the usual entries common to all streams (see Table 5). The dictionary's **Width** and **Height** entries define the dimensions of the first (or only) rectangle. The dimensions of the second, optional rectangle are defined by the optional entries **Width2** and **Height2**. Each threshold value shall be represented as 2 bytes, with the high-order byte first. The stream shall contain 2 × **Width** × **Height** bytes if there is only one rectangle or 2 × (**Width** × **Height** + **Width2** × **Height2**) bytes if there are two rectangles. The contents of the first rectangle are specified first, followed by those of the second rectangle. Threshold values within each rectangle shall be defined in device space in the same order as image samples in image space (see Figure 34), with the first value at device coordinates (0, 0) and horizontal coordinates changing faster than vertical coordinates.

Table 133 – Additional entries specific to a type 16 halftone dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Halftone** for a halftone dictionary.
HalftoneType	integer	*(Required)* A code identifying the halftone type that this dictionary describes; shall be 16 for this type of halftone.
HalftoneName	byte string	*(Optional)* The name of the halftone dictionary.
Width	integer	*(Required)* The width of the first (or only) rectangle in the threshold array, in device pixels.
Height	integer	*(Required)* The height of the first (or only) rectangle in the threshold array, in device pixels.
Width2	integer	*(Optional)* The width of the optional second rectangle in the threshold array, in device pixels. If this entry is present, the **Height2** entry shall be present as well. If this entry is absent, the **Height2** entry shall also be absent, and the threshold array has only one rectangle.
Height2	integer	*(Optional)* The height of the optional second rectangle in the threshold array, in device pixels.
TransferFunction	function or name	*(Optional)* A transfer function, which shall override the current transfer function in the graphics state for the same component. This entry shall be present if the dictionary is a component of a type 5 halftone (see 10.5.5.6, "Type 5 Halftones") and represents either a nonprimary or nonstandard primary colour component (see 10.4, "Transfer Functions"). The name **Identity** may be used to specify the identity function.

10.5.5.6 Type 5 Halftones

Some devices, particularly colour printers, require separate halftones for each individual colorant. Also, devices that can produce named separations may require individual halftones for each separation. Halftone dictionaries of type 5 allow individual halftones to be specified for an arbitrary number of colorants or colour components.

A type 5 halftone dictionary (Table 134) is a composite dictionary containing independent halftone definitions for multiple colorants. Its keys shall be name objects representing the names of individual colorants or colour components. The values associated with these keys shall be other halftone dictionaries, each defining the halftone screen and transfer function for a single colorant or colour component. The component halftone dictionaries shall not be of halftone type 5.

Table 134 – Entries in a type 5 halftone dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Halftone** for a halftone dictionary.
HalftoneType	number	*(Required)* A code identifying the halftone type that this dictionary describes; shall be 5 for this type of halftone.
HalftoneName	byte string	*(Optional)* The name of the halftone dictionary.
any colorant name	dictionary or stream	*(Required, one per colorant)* The halftone corresponding to the colorant or colour component named by the key. The halftone may be of any type other than 5.
Default	dictionary or stream	*(Required)* A halftone to be used for any colorant or colour component that does not have an entry of its own. The value shall not be 5. If there are any nonprimary colorants, the default halftone shall have a transfer function.

The colorants or colour components represented in a type 5 halftone dictionary (aside from the Default entry) fall into two categories:

- Primary colour components for the standard native device colour spaces (Gray for **DeviceGray**; **Red**, **Green**, and **Blue** for **DeviceRGB**; **Cyan**, **Magenta**, **Yellow**, and **Black** for **DeviceCMYK**;).
- Nonstandard colour components for use as spot colorants in **Separation** and **DeviceN** colour spaces. Some of these may also be used as process colorants if the native colour space is nonstandard.

When a halftone dictionary of some other type appears as the value of an entry in a type 5 halftone dictionary, it shall apply only to the single colorant or colour component named by that entry's key. This is in contrast to such a dictionary's being used as the current halftone parameter in the graphics state, which shall apply to all colour components. If nonprimary colorants are requested when the current halftone is defined by any means other than a type 5 halftone dictionary, the gray halftone screen and transfer function shall be used for all such colorants.

EXAMPLE In this example, the halftone dictionaries for the colour components and for the default all use the same spot function. In this example, the halftone dictionaries for the colour components and for the default all use the same spot function.

```
27 0 obj
   << /Type /Halftone
        /HalftoneType 5
        /Cyan 31 0 R
        /Magenta 32 0 R
        /Yellow 33 0 R
        /Black 34 0 R
        /Default 35 0 R
   >>
endobj

31 0 obj
   << /Type /Halftone
        /HalftoneType 1
        /Frequency 89.827
        /Angle 15
        /SpotFunction /Round
        /AccurateScreens true
   >>
endobj

32 0 obj
```

```
    << /Type /Halftone
         /HalftoneType 1
         /Frequency 89.827
         /Angle 75
         /SpotFunction /Round
         /AccurateScreens true
    >>
endobj

33 0 obj
    << /Type /Halftone
         /HalftoneType 1
         /Frequency 90.714
         /Angle 0
         /SpotFunction /Round
         /AccurateScreens true
    >>
endobj

34 0 obj
    << /Type /Halftone
         /HalftoneType 1
         /Frequency 89.803
         /Angle 45
         /SpotFunction /Round
         /AccurateScreens true
    >>
endobj

35 0 obj
    << /Type /Halftone
         /HalftoneType 1
         /Frequency 90.000
         /Angle 45
         /SpotFunction /Round
         /AccurateScreens true
    >>
endobj
```

10.6 Scan Conversion Details

10.6.1 General

The final step of rendering shall be *scan conversion*. The conforming reader executes a scan conversion algorithm to paint graphics, text, and images in the raster memory of the output device.

NOTE The specifics of the scan conversion algorithm are not defined as part of PDF. Different implementations may perform scan conversion in different ways; techniques that are appropriate for one device may be inappropriate for another. Still, it is useful to have a general understanding of how scan conversion works, particularly when creating PDF files intended for viewing on a display. At the low resolutions typical of displays, variations of even one pixel's width can have a noticeable effect on the appearance of painted shapes.

Most scan conversion details are not under program control, but a few are; the parameters for controlling them are described here.

10.6.2 Flatness Tolerance

The *flatness tolerance* controls the maximum permitted distance in device pixels between the mathematically correct path and an approximation constructed from straight line segments, as shown in Figure 54. Flatness may be specified as the operand of the **i** operator (see Table 57) or as the value of the **FL** entry in a graphics state parameter dictionary (see Table 58). It shall be a positive number.

NOTE 1 Smaller values yield greater precision at the cost of more computation.

NOTE 2 Although the figure exaggerates the difference between the curved and flattened paths for the sake of clarity, the purpose of the flatness tolerance is to control the precision of curve rendering, not to draw inscribed polygons. If the parameter's value is large enough to cause visible straight line segments to appear, the result is unpredictable.

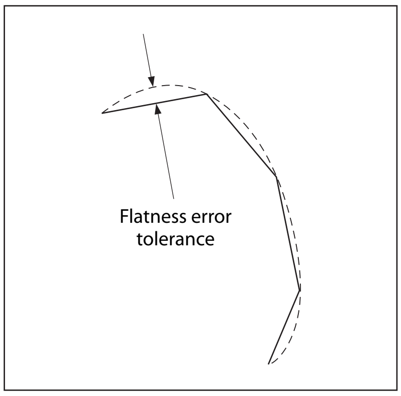


Figure 54 – Flatness tolerance

10.6.3 Smoothness Tolerance

The *smoothness tolerance (PDF 1.3)* controls the quality of smooth shading (type 2 patterns and the **sh** operator) and thus indirectly controls the rendering performance. Smoothness is the allowable colour error between a shading approximated by piecewise linear interpolation and the true value of a (possibly nonlinear) shading function. The error shall be measured for each colour component, and the maximum independent error shall be used. The allowable error (or tolerance) shall be expressed as a fraction of the range of the colour component, from 0.0 to 1.0. Thus, a smoothness tolerance of 0.1 represents a tolerance of 10 percent in each colour component. Smoothness may be specified as the value of the **SM** entry in a graphics state parameter dictionary (see Table 58).

EXAMPLE Each output device may have internal limits on the maximum and minimum tolerances attainable. setting smoothness to 1.0 may result in an internal smoothness of 0.5 on a high-quality colour device, while setting it to 0.0 on the same device may result in an internal smoothness of 0.01 if an error of that magnitude is imperceptible on the device.

NOTE 1 The smoothness tolerance may also interact with the accuracy of colour conversion. In the case of a colour conversion defined by a sampled function, 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 tolerance cannot be guaranteed. In most cases, however, where the conversion function is smooth and continuous, the accuracy should be within the specified tolerance.

NOTE 2 The effect of the smoothness tolerance is similar to that of the flatness tolerance. However, that flatness is measured in device-dependent units of pixel width, whereas smoothness is measured as a fraction of colour component range.

10.6.4 Scan Conversion Rules

The following rules determine which device pixels a painting operation affects. All references to coordinates and pixels are in device space. A *shape* is a path to be painted with the current colour or with an image. Its coordinates are mapped into device space but not rounded to device pixel boundaries. At this level, curves have been flattened to sequences of straight lines, and all "insideness" computations have been performed.

Pixel boundaries always fall on integer coordinates in device space. A pixel is a square region identified by the location of its corner with minimum horizontal and vertical coordinates. The region is *half-open*, meaning that it includes its lower but not its upper boundaries. More precisely, for any point whose real-number coordinates are (x, y), let $i = \text{floor}(x)$ and $j = \text{floor}(y)$. The pixel that contains this point is the one identified as (i, j). The region belonging to that pixel is defined to be the set of points (x', y') such that $i \leq x' < i + 1$ and $j \leq y' < j + 1$. Like pixels, shapes to be painted by filling and stroking operations are also treated as half-open regions that include the boundaries along their “floor” sides, but not along their “ceiling” sides.

A shape shall be scan-converted by painting any pixel whose square region intersects the shape, no matter how small the intersection is. This ensures that no shape ever disappears as a result of unfavourable placement relative to the device pixel grid, as might happen with other possible scan conversion rules. The area covered by painted pixels shall always be at least as large as the area of the original shape. This rule applies both to fill operations and to strokes with nonzero width. Zero-width strokes may be done in an implementation-defined manner that may include fewer pixels than the rule implies.

NOTE 1 Normally, the intersection of two regions is defined as the intersection of their interiors. However, for purposes of scan conversion, a filling region is considered to intersect every pixel through which its boundary passes, even if the interior of the filling region is empty.

EXAMPLE A zero-width or zero-height rectangle paints a line 1 pixel wide.

The region of device space to be painted by a sampled image is determined similarly to that of a filled shape, though not identically. The conforming reader transforms the image’s source rectangle into device space and defines a half-open region, just as for fill operations. However, only those pixels whose *centres* lie within the region shall be painted. The position of the centre of such a pixel—in other words, the point whose coordinate values have fractional parts of one-half—shall be mapped back into source space to determine how to colour the pixel. There shall not be averaging over the pixel area;

NOTE 2 If the resolution of the source image is higher than that of device space, some source samples may not be used.

For clipping, the clipping region consists of the set of pixels that would be included by a fill operation. Subsequent painting operations shall affect a region that is the intersection of the set of pixels defined by the clipping region with the set of pixels for the region to be painted.

Scan conversion of character glyphs may be performed by a different algorithm from the preceding one.

NOTE 3 That font rendering algorithm uses hints in the glyph descriptions and techniques that are specialized to glyph rasterization.

10.6.5 Automatic Stroke Adjustment

When a stroke is drawn along a path, the scan conversion algorithm may produce lines of nonuniform thickness because of rasterization effects. In general, the line width and the coordinates of the endpoints, transformed into device space, are arbitrary real numbers not quantized to device pixels. A line of a given width can intersect with different numbers of device pixels, depending on where it is positioned. Figure 55 illustrates this effect.

For best results, it is important to compensate for the rasterization effects to produce strokes of uniform thickness. This is especially important in low-resolution display applications. To meet this need, PDF 1.2 provides an optional *automatic stroke adjustment* feature. When stroke adjustment is enabled, the line width and the coordinates of a stroke shall automatically be adjusted as necessary to produce lines of uniform thickness. The thickness shall be as near as possible to the requested line width—no more than half a pixel different.

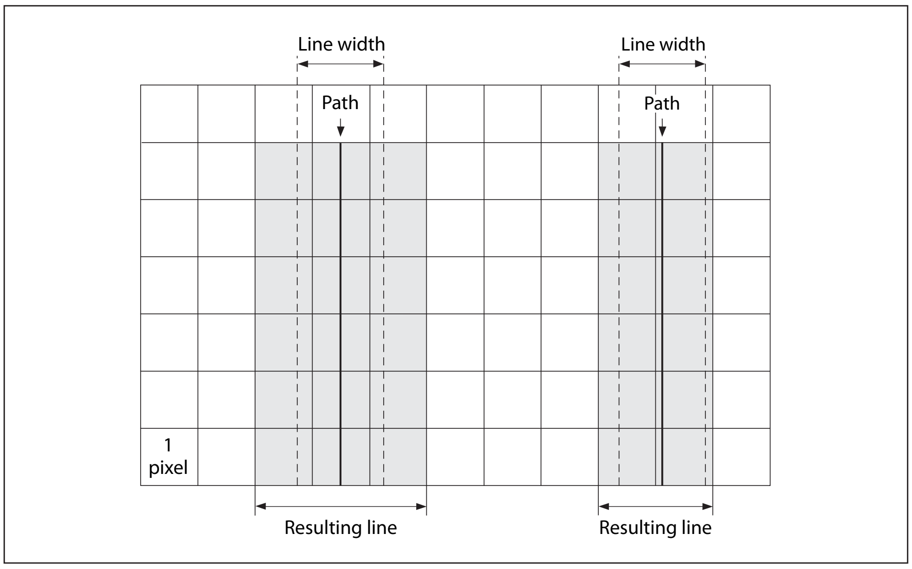


Figure 55 – Rasterization without stroke adjustment

If stroke adjustment is enabled and the requested line width, transformed into device space, is less than half a pixel, the stroke shall be rendered as a single-pixel line.

NOTE This is the thinnest line that can be rendered at device resolution. It is equivalent to the effect produced by setting the line width to 0 (see 10.6.4, "Scan Conversion Rules").

Because automatic stroke adjustment can have a substantial effect on the appearance of lines, PDF provides means to control whether the adjustment shall be performed. This may be specified with the stroke adjustment parameter in the graphics state, set by means of the **SA** entry in a graphics state parameter dictionary (see 8.4.5, "Graphics State Parameter Dictionaries").

11 Transparency

11.1 General

The PDF imaging model includes the notion of *transparency*. Transparent objects do not necessarily obey a strict opaque painting model but may blend *(composite)* in interesting ways with other overlapping objects. This clause describes the general transparency model but does not cover how it is implemented. At various points it uses implementation-like descriptions to describe how things work, for the purpose of elucidating the behaviour of the model. The actual implementation will almost certainly be different from what these descriptions might imply.

NOTE Transparency was added to PDF in version 1.4

The clause is organized as follows:

- 11.2, "Overview of Transparency," introduces the basic concepts of the transparency model and its associated terminology.
- 11.3, "Basic Compositing Computations," describes the mathematics involved in compositing a single object with its backdrop.
- 11.4, "Transparency Groups," introduces the concept of *transparency groups* and describes their properties and behaviour.
- 11.5, "Soft Masks," covers the creation and use of masks to specify position-dependent shape and opacity.
- 11.6, "Specifying Transparency in PDF," describes how transparency properties are represented in a PDF document.
- 11.7, "Colour Space and Rendering Issues," deals with some specific interactions between transparency and other aspects of colour specification and rendering.

11.2 Overview of Transparency

The original PDF imaging model paints objects (fills, strokes, text, and images), possibly clipped by a path, opaquely onto a page. The colour of the page at any point shall be that of the topmost enclosing object, disregarding any previous objects it may overlap. This effect may be—and often is—realized simply by rendering objects directly to the page in the order in which they are specified, with each object completely overwriting any others that it overlaps.

Under the transparent imaging model, all of the objects on a page may potentially contribute to the result. Objects at a given point may be thought of as forming a *transparency stack* (or *stack* for short). The objects shall be arranged from bottom to top in the order in which they are specified. The colour of the page at each point shall be determined by combining the colours of all enclosing objects in the stack according to *compositing* rules defined by the transparency model.

NOTE 1 The order in which objects are specified determines the stacking order but not necessarily the order in which the objects are actually painted onto the page. In particular, the transparency model does not require a conforming reader to rasterize objects immediately or to commit to a raster representation at any time before rendering the entire stack onto the page. This is important, since rasterization often causes significant loss of information and precision that is best avoided during intermediate stages of the transparency computation.

A given object shall be composited with a *backdrop*. Ordinarily, the backdrop consists of the stack of all objects that have been specified previously. The result of compositing shall then be treated as the backdrop for the next object. However, within certain kinds of transparency groups (see “Transparency Groups”), a different backdrop may be chosen.

During the compositing of an object with its backdrop, the colour at each point shall be computed using a specified *blend mode*, which is a function of both the object's colour and the backdrop colour. The blend mode shall determine how colours interact; different blend modes may be used to achieve a variety of useful effects. A single blend mode shall be in effect for compositing all of a given object, but different blend modes may be applied to different objects.

Two scalar quantities called *shape* and *opacity* mediate compositing of an object with its backdrop. Conceptually, for each object, these quantities shall be defined at every point in the plane, just as if they were additional colour components. (In actual practice, they may be obtained from auxiliary sources rather than being intrinsic to the object.)

Both shape and opacity vary from 0.0 (no contribution) to 1.0 (maximum contribution). At any point where either the shape or the opacity of an object is equal to 0.0, its colour shall be undefined. At points where the shape is equal to 0.0, the opacity shall also be undefined. The shape and opacity shall be subject to compositing rules; therefore, the stack as a whole also has a shape and opacity at each point.

An object's opacity, in combination with the backdrop's opacity, shall determine the relative contributions of the backdrop colour, the object's colour, and the blended colour to the resulting composite colour. The object's shape shall then determine the degree to which the composite colour replaces the backdrop colour. Shape values of 0.0 and 1.0 identify points that lie outside and inside a conventional sharp-edged object; intermediate values are useful in defining soft-edged objects.

Shape and opacity are conceptually very similar. In fact, they can usually be combined into a single value, called *alpha*, which controls both the colour compositing computation and the fading between an object and its backdrop. However, there are a few situations in which they shall be treated separately; see 11.4.6, "Knockout Groups."

NOTE 2 Raster-based implementations may need to maintain a separate shape parameter to do anti-aliasing properly; it is therefore convenient to have shape as an explicit part of the model.

One or more consecutive objects in a stack may be collected together into a *transparency group* (often referred to hereafter simply as a *group*). The group as a whole may have various properties that modify the compositing behaviour of objects within the group and their interactions with its backdrop. An additional blend mode, shape, and opacity may also be associated with the group as a whole and used when compositing it with its backdrop. Groups may be nested within other groups, forming a tree-structured hierarchy.

EXAMPLE Figure L.16 in Annex L illustrates the effects of transparency grouping. In the upper two figures, three coloured circles are painted as independent objects with no grouping. At the upper left, the three objects are painted opaquely (opacity = 1.0); each object completely replaces its backdrop (including previously painted objects) with its own colour. At the upper right, the same three independent objects are painted with an opacity of 0.5, causing them to composite with each other and with the gray and white backdrop. In the lower two figures, the three objects are combined as a transparency group. At the lower left, the individual objects have an opacity of 1.0 within the group, but the group as a whole is painted in the **Normal** blend mode with an opacity of 0.5. The objects thus completely overwrite each other within the group, but the resulting group then composites transparently with the gray and white backdrop. At the lower right, the objects have an opacity of 0.5 within the group and thus composite with each other. The group as a whole is painted against the backdrop with an opacity of 1.0 but in a different blend mode (**HardLight**), producing a different visual effect.

The colour result of compositing a group may be converted to a single-component luminosity value and treated as a *soft mask*. Such a mask may then be used as an additional source of shape or opacity values for subsequent compositing operations. When the mask is used as a shape, this technique is known as *soft clipping*; it is a generalization of the current clipping path in the opaque imaging model (see "Clipping Path Operators").

The notion of *current page* is generalized to refer to a transparency group consisting of the entire stack of objects placed on the page, composited with a backdrop that is pure white and fully opaque. Logically, this entire stack shall then be rasterized to determine the actual pixel values to be transmitted to the output device.

NOTE 3 In contexts where a PDF page is treated as a piece of artwork to be placed on some other page it is treated not as a page but as a group, whose backdrop may be defined differently from that of a page.

11.3 Basic Compositing Computations

11.3.1 General

This sub-clause describes the basic computations for compositing a single object with its backdrop. These computations are extended in 11.4, "Transparency Groups," to cover groups consisting of multiple objects.

11.3.2 Basic Notation for Compositing Computations

In general, variable names in this clause consisting of a lowercase letter denote a scalar quantity, such as an opacity. Uppercase letters denote a value with multiple scalar components, such as a colour. In the descriptions of the basic colour compositing computations, colour values are generally denoted by the letter *C*, with a mnemonic subscript indicating which of several colour values is being referred to; for instance, C_S stands for "source colour." Shape and opacity values are denoted respectively by the letters *f* (for "*f*orm factor") and *q* (for "opa*q*ueness")—again with a mnemonic subscript, such as q_S for "source opacity." The symbol α (alpha) stands for a product of shape and opacity values.

In certain computations, one or more variables may have undefined values; for instance, when opacity is equal to zero, the corresponding colour is undefined. A quantity can also be undefined if it results from division by zero. In any formula that uses such an undefined quantity, the quantity has no effect on the ultimate result because it is subsequently multiplied by zero or otherwise cancelled out. It is significant that while any arbitrary value may be chosen for such an undefined quantity, the computation shall not malfunction because of exceptions caused by overflow or division by zero. The further convention that $0 \div 0 = 0$ should also be adopted.

11.3.3 Basic Compositing Formula

The primary change in the imaging model to accommodate transparency is in how colours are painted. In the transparent model, the result of painting (the *result colour*) is a function of both the colour being painted (the *source colour*) and the colour it is painted over (the *backdrop colour*). Both of these colours may vary as a function of position on the page; however, this sub-clause focuses on some fixed point on the page and assumes a fixed backdrop and source colour.

This computation uses two other parameters: *alpha*, which controls the relative contributions of the backdrop and source colours, and the *blend function*, which specifies how they shall be combined in the painting operation. The resulting *basic colour compositing formula* (or just *basic compositing formula* for short) shall determine the result colour produced by the painting operation:

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times [(1 - \alpha_b) \times C_s + \alpha_b \times B(C_b, C_s)]$$

where the variables have the meanings shown in Table 135.

Table 135 – Variables used in the basic compositing formula

Variable	Meaning
C_b	Backdrop colour
C_s	Source colour
C_r	Result colour

Table 135 – Variables used in the basic compositing formula (continued)

Variable	Meaning
α_b	Backdrop alpha
α_s	Source alpha
α_r	Result alpha
$B(C_b, C_s)$	Blend function

This formula represents a simplified form of the compositing formula in which the shape and opacity values are combined and represented as a single alpha value; the more general form is presented later. This function is based on the **over** operation **defined in the article "Compositing Digital Images," by Porter and Duff (see the Bibliography)**, extended to include a blend mode in the region of overlapping coverage. The following sub-clauses elaborate on the meaning and implications of this formula.

11.3.4 Blending Colour Space

The compositing formula shown in 11.3.3, "Basic Compositing Formula," represents a vector function: the colours it operates on are represented in the form of *n*-element vectors, where *n* denotes the number of components required by the colour space in used in the compositing process. The *i*th component of the result colour C_r shall be obtained by applying the compositing formula to the *i*th components of the constituent colours C_b, C_s, and $B(C_b, C_s)$. The result of the computation thus depends on the colour space in which the colours are represented. For this reason, the colour space used for compositing, called the *blending colour space*, is explicitly made part of the transparent imaging model. When necessary, backdrop and source colours shall be converted to the blending colour space before the compositing computation.

Of the PDF colour spaces described in Section 8.6, the following shall be supported as blending colour spaces:

- **DeviceGray**
- **DeviceRGB**
- **DeviceCMYK**
- **CalGray**
- **CalRGB**
- **ICCBased** colour spaces equivalent to the preceding (including calibrated *CMYK*)

The **Lab** space and **ICCBased** spaces that represent lightness and chromaticity separately (such as *L*a*b**, *L*u*v**, and *HSV*) shall not be used as blending colour spaces because the compositing computations in such spaces do not give meaningful results when applied separately to each component. In addition, an **ICCBased** space used as a blending colour space shall be bidirectional; that is, the ICC profile shall contain both *AToB* and *BToA* transformations.

The blending colour space shall be consulted only for process colours. Although blending may also be done on individual spot colours specified in a **Separation** or **DeviceN** colour space, such colours shall not be converted to a blending colour space (except in the case where they first revert to their alternate colour space, as described under Section 8.6.6.4 and "DeviceN Colour Spaces"). Instead, the specified colour components shall be blended individually with the corresponding components of the backdrop.

The blend functions for the various blend modes are defined such that the range for each colour component shall be 0.0 to 1.0 and that the colour space shall be additive. When performing blending operations in subtractive colour spaces (**DeviceCMYK**, **Separation**, and **DeviceN**), the colour component values shall be complemented (subtracted from 1.0) before the blend function is applied and the results of the function shall then be complemented back before being used.

NOTE This adjustment makes the effects of the various blend modes numerically consistent across all colour spaces. However, the actual visual effect produced by a given blend mode still depends on the colour space. Blending in a device colour space produces device-dependent results, whereas in a CIE-based space it produces results that are consistent across all devices. See 11.7, "Colour Space and Rendering Issues," for additional details concerning colour spaces.

11.3.5 Blend Mode

In principle, any function of the backdrop and source colours that yields another colour, C_r, for the result may be used as a blend function B (C_b, C_s), in the compositing formula to customize the blending operation. PDF defines a standard set of named blend functions, or *blend modes*, listed in Tables 136 and 137. Figures L.18 and L.19 in Annex L illustrate the resulting visual effects for *RGB* and *CMYK* colours, respectively.

A blend mode is termed *separable* if each component of the result colour is completely determined by the corresponding components of the constituent backdrop and source colours—that is, if the blend mode function *B* is applied separately to each set of corresponding components:

$$c_r = B(c_b, c_s)$$

where the lowercase variables c_r, c_b, and c_s denote corresponding components of the colours C_r, C_b, and C_s, expressed in additive form. A separable blend mode may be used with any colour space, since it applies independently to any number of components. Only separable blend modes shall be used for blending spot colours.

NOTE 1 Theoretically, a blend mode could have a different function for each colour component and still be separable; however, none of the standard PDF blend modes have this property.

Table 136 lists the standard separable blend modes available in PDF and the algorithms/formulas that shall be used in the calculation of blended colours.

Table 136 – Standard separable blend modes

Name	Result
Normal	$B(c_b, c_s) = c_s$ NOTE Selects the source colour, ignoring the backdrop.
Compatible	Same as Normal. This mode exists only for compatibility and should not be used.
Multiply	$B(c_b, c_s) = c_b \times c_s$ NOTE 1 Multiplies the backdrop and source colour values. NOTE 2 The result colour is always at least as dark as either of the two constituent colours. Multiplying any colour with black produces black; multiplying with white leaves the original colour unchanged. Painting successive overlapping objects with a colour other than black or white produces progressively darker colours.

Table 136 – Standard separable blend modes (continued)

Name	Result
Screen	$B(c_b, c_s) = 1 - [(1 - c_b) \times (1 - c_s)]$ $= c_b + c_s - (c_b \times c_s)$ NOTE 3 Multiplies the complements of the backdrop and source colour values, then complements the result. NOTE 4 The result colour is always at least as light as either of the two constituent colours. Screening any colour with white produces white; screening with black leaves the original colour unchanged. The effect is similar to projecting multiple photographic slides simultaneously onto a single screen.
Overlay	$B(c_b, c_s) = \text{HardLight}(c_s, c_b)$ NOTE 5 Multiplies or screens the colours, depending on the backdrop colour value. Source colours overlay the backdrop while preserving its highlights and shadows. The backdrop colour is not replaced but is mixed with the source colour to reflect the lightness or darkness of the backdrop.
Darken	$B(c_b, c_s) = \min(c_b, c_s)$ NOTE 6 Selects the darker of the backdrop and source colours. NOTE 7 The backdrop is replaced with the source where the source is darker; otherwise, it is left unchanged.
Lighten	$B(c_b, c_s) = \max(c_b, c_s)$ NOTE 8 Selects the lighter of the backdrop and source colours. NOTE 9 The backdrop is replaced with the source where the source is lighter; otherwise, it is left unchanged.
ColorDodge	$B(c_b, c_s) = \begin{cases} \min(1, c_b / (1 - c_s)) & \text{if } c_s < 1 \\ 1 & \text{if } c_s = \end{cases}$ NOTE 10 Brightens the backdrop colour to reflect the source colour. Painting with black produces no changes.
ColorBurn	$B(c_b, c_s) = \begin{cases} 1 - \min(1, (1 - c_b) / c_s) & \text{if } c_s > 0 \\ 0 & \text{if } c_s = \end{cases}$ NOTE 11 Darkens the backdrop colour to reflect the source colour. Painting with white produces no change.
HardLight	$B(c_b, c_s) = \begin{cases} \text{Multiply}(c_b, 2 \times c_s) & \text{if } c_s \le 0.5 \\ \text{Screen}(c_b, 2 \times c_s - 1) & \text{if } c_s > 0.5 \end{cases}$ NOTE 12 Multiplies or screens the colours, depending on the source colour value. The effect is similar to shining a harsh spotlight on the backdrop.

Table 136 – Standard separable blend modes (continued)

Name	Result
SoftLight	$B(c_b, c_s) = \begin{cases} c_b - (1 - 2 \times c_s) \times c_b \times (1 - c_b) & \text{if } c_s \leq 0.5 \\ c_b + (2 \times c_s - 1) \times (D(c_b) - c_b) & \text{if } c_s > 0.5 \end{cases}$ **where** $D(x) = \begin{cases} ((16 \times x - 12) \times x + 4) \times x & \text{if } x \leq 0.25 \\ \sqrt{x} & \text{if } x > 0.25 \end{cases}$ NOTE 13 Darkens or lightens the colours, depending on the source colour value. The effect is similar to shining a diffused spotlight on the backdrop.
Difference	$B(c_b, c_s) = \lvert c_b - c_s \rvert$ NOTE 14 Subtracts the darker of the two constituent colours from the lighter colour: NOTE 15 Painting with white inverts the backdrop colour; painting with black produces no change.
Exclusion	$B(c_b, c_s) = c_b + c_s - 2 \times c_b \times c_s$ NOTE 16 Produces an effect similar to that of the **Difference** mode but lower in contrast. Painting with white inverts the backdrop colour; painting with black produces no change.

Table 137 lists the standard nonseparable blend modes. Since the nonseparable blend modes consider all colour components in combination, their computation depends on the blending colour space in which the components are interpreted. They may be applied to all multiple-component colour spaces that are allowed as blending colour spaces (see "Blending Colour Space").

NOTE 2 All of these blend modes conceptually entail the following steps:

a) Convert the backdrop and source colours from the blending colour space to an intermediate *HSL* (hue-saturation-luminosity) representation.

b) Create a new colour from some combination of hue, saturation, and luminosity components selected from the backdrop and source colours.

c) Convert the result back to the original (blending) colour space.

However, the following formulas given do not actually perform these conversions. Instead, they start with whichever colour (backdrop or source) is providing the hue for the result; then they adjust this colour to have the proper saturation and luminosity.

The nonseparable blend mode formulas make use of several auxiliary functions. These functions operate on colours that are assumed to have red, green, and blue components. Blending of *CMYK* colour spaces requires special treatment, as described in this sub-clause.

These functions shall have the following definitions:

$$\mathrm{Lum}(C) = 0.3 \times C_{red} + 0.59 \times C_{green} + 0.11 \times C_{blue}$$

```
$SetLum(C, l)$
    let $d = l - Lum(C)$
    $C_{red} = C_{red} + d$
    $C_{green} = C_{green} + d$
    $C_{blue} = C_{blue} + d$
    return $ClipColor(C)$

$ClipColor(C)$
    let $l = Lum(C)$
    let $n = min(C_{red}, C_{green}, C_{blue})$
    let $x = max(C_{red}, C_{green}, C_{blue})$
    if $n < 0.0$
        $C_{red} = l + (((C_{red} - l) \times l) \,/\, (l - n))$
        $C_{green} = l + (((C_{green} - l) \times l) \,/\, (l - n))$
        $C_{blue} = l + (((C_{blue} - l) \times l) \,/\, (l - n))$
    if $x > 1.0$
        $C_{red} = l + (((C_{red} - l) \times (1 - l)) \,/\, (x - l))$
        $C_{green} = l + (((C_{green} - l) \times (1 - l)) \,/\, (x - l))$
        $C_{blue} = l + (((C_{blue} - l) \times (1 - l)) \,/\, (x - l))$
    return C
```

$$Sat(C) = max(C_{red}, C_{green}, C_{blue}) - min(C_{red}, C_{green}, C_{blue})$$

The subscripts *min*, *mid*, and *max* (in the next function) refer to the colour components having the minimum, middle, and maximum values upon entry to the function.

```
$SetSat(C, s)$
    if $C_{max} > C_{min}$
        $C_{mid} = (((C_{mid} - C_{min}) \times s) \,/\, (C_{max} - C_{min}))$
        $C_{max} = s$
    else
        $C_{mid} = C_{max} = 0.0$
    $C_{min} = 0.0$
    return C
```

Table 137 – Standard nonseparable blend modes

Name	Result
Hue	$B(C_b, C_s) = SetLum(SetSat(C_s, Sat(C_b)), Lum(C_b))$ NOTE 1 Creates a colour with the hue of the source colour and the saturation and luminosity of the backdrop colour.
Saturation	$B(C_b, C_s) = SetLum(SetSat(C_b, Sat(C_s)), Lum(C_b))$ NOTE 2 Creates a colour with the saturation of the source colour and the hue and luminosity of the backdrop colour. Painting with this mode in an area of the backdrop that is a pure gray (no saturation) produces no change.

Table 137 – Standard nonseparable blend modes (continued)

Name	Result
Color	$B(C_b, C_s) = \text{SetLum}(C_s, \text{Lum}(C_b))$ NOTE 3 Creates a colour with the hue and saturation of the source colour and the luminosity of the backdrop colour. This preserves the gray levels of the backdrop and is useful for colouring monochrome images or tinting colour images.
Luminosity	$B(C_b, C_s) = \text{SetLum}(C_b, \text{Lum}(C_s))$ NOTE 4 Creates a colour with the luminosity of the source colour and the hue and saturation of the backdrop colour. This produces an inverse effect to that of the **Color** mode.

The formulas in this sub-clause apply to *RGB* spaces. Blending in *CMYK* spaces (including both **DeviceCMYK** and **ICCBased** calibrated *CMYK* spaces) shall be handled in the following way:

- The *C*, *M*, and *Y* components shall be converted to their complementary *R*, *G*, and *B* components in the usual way. The preceding formulas shall be applied to the *RGB* colour values. The results shall be converted back to *C*, *M*, and *Y*.
- For the *K* component, the result shall be the *K* component of C_b for the **Hue**, **Saturation**, and **Color** blend modes; it shall be the *K* component of C_s for the **Luminosity** blend mode.

11.3.6 Interpretation of Alpha

The colour compositing formula

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times [(1 - \alpha_b) \times C_s + \alpha_b \times B(C_b, C_s)]$$

produces a result colour that is a weighted average of the backdrop colour, the source colour, and the blended $B(C_b, C_s)$ term, with the weighting determined by the backdrop and source alphas α_β and α_s. For the simplest blend mode, **Normal**, defined by

$$B(c_b, c_s) = c_s$$

the compositing formula collapses to a simple weighted average of the backdrop and source colours, controlled by the backdrop and source alpha values. For more interesting blend functions, the backdrop and source alphas control whether the effect of the blend mode is fully realized or is toned down by mixing the result with the backdrop and source colours.

The result alpha, α_ρ, actually represents a computed result, described in 11.3.7, "Shape and Opacity Computations." The result colour shall be normalized by the result alpha, ensuring that when this colour and alpha are subsequently used together in another compositing operation, the colour's contribution is correctly represented.

NOTE 1 If α_ρ is zero, the result colour is undefined.

NOTE 2 The preceding formula represents a simplification of the following formula, which presents the relative contributions of backdrop, source, and blended colours in a more straightforward way:

$$\alpha_r \times C_r = [(1 - \alpha_s) \times \alpha_b \times C_b] + [(1 - \alpha_b) \times \alpha_s \times C_s] + [\alpha_b \times \alpha_s \times B(C_b, C_s)$$

(The simplification requires a substitution based on the alpha compositing formula, which is presented in the next sub-clause.) Thus, mathematically, the backdrop and source alphas control the influence of the backdrop and source colours, respectively, while their product controls the influence of the blend function. An alpha value of $\alpha_s = 0.0$ or $\alpha_\beta = 0.0$ results in no blend mode effect; setting $\alpha_s = 1.0$ and $\alpha_\beta = 1.0$ results in maximum blend mode effect.

11.3.7 Shape and Opacity Computations

11.3.7.1 General

As stated earlier, the alpha values that control the compositing process shall be defined as the product of shape and opacity:

$$\begin{aligned} \alpha_b &= f_b \times q_b \\ \alpha_r &= f_r \times q_r \\ \alpha_s &= f_s \times q_s \end{aligned}$$

This sub-clause examines the various shape and opacity values individually. Once again, keep in mind that conceptually these values are computed for every point on the page.

11.3.7.2 Source Shape and Opacity

Shape and opacity values may come from several sources. The transparency model provides for three independent sources for each. However, the PDF representation imposes some limitations on the ability to specify all of these sources independently (see “Specifying Shape and Opacity”).

- *Object shape*. Elementary objects such as strokes, fills, and text have an intrinsic shape, whose value shall be 1.0 for points inside the object and 0.0 outside. Similarly, an image with an explicit mask (see “Explicit Masking”) has a shape that shall be 1.0 in the unmasked portions and 0.0 in the masked portions. The shape of a group object shall be the union of the shapes of the objects it contains.

NOTE 1 Mathematically, elementary objects have “hard” edges, with a shape value of either 0.0 or 1.0 at every point. However, when such objects are rasterized to device pixels, the shape values along the boundaries may be anti-aliased, taking on fractional values representing fractional coverage of those pixels. When such anti-aliasing is performed, it is important to treat the fractional coverage as shape rather than opacity.

- *Mask shape*. Shape values for compositing an object may be taken from an additional source, or *soft mask*, independent of the object itself, as described in 11.5, "Soft Masks."

NOTE 2 The use of a soft mask to modify the shape of an object or group, called *soft clipping*, can produce effects such as a gradual transition between an object and its backdrop, as in a vignette.

- *Constant shape*. The source shape may be modified at every point by a scalar *shape constant*.

NOTE 3 This is merely a convenience, since the same effect could be achieved with a shape mask whose value is the same everywhere.

- *Object opacity*. Elementary objects have an opacity of 1.0 everywhere. The opacity of a group object shall be the result of the opacity computations for all of the objects it contains.

- *Mask opacity*. Opacity values, like shape values, may be provided by a soft mask independent of the object being composited.

- *Constant opacity*. The source opacity may be modified at every point by a scalar *opacity constant*.

NOTE 4 It is useful to think of this value as the “current opacity,” analogous to the current colour used when painting elementary objects.

All of the shape and opacity inputs shall have values in the range 0.0 to 1.0 (inclusive), with a default value of 1.0.

The three shape inputs shall be multiplied together, producing an intermediate value called the source shape.

$$f_s = f_j \times f_m \times f_k$$

The three opacity inputs shall be multiplied together, producing an intermediate value called the source opacity.

$$q_s = q_j \times q_m \times q_k$$

Where the variables have the meanings shown in Table 138.

Table 138 – Variables used in the source shape and opacity formulas

Variable	Meaning
f_s	Source shape
f_j	Object shape
f_m	Mask shape
f_k	Constant shape
q_s	Source opacity
q_j	Object opacity
q_m	Mask opacity
q_k	Constant opacity

NOTE 5 The effect of each of these inputs is that the painting operation becomes more transparent as the input values decreases.

When an object is painted with a tiling pattern, the object shape and object opacity for points in the object's interior are determined by those of corresponding points in the pattern, rather than being 1.0 everywhere (see "Patterns and Transparency").

11.3.7.3 Result Shape and Opacity

In addition to a result colour, the painting operation also shall compute an associated *result shape* and *result opacity*. These computations shall be based on the *union function*

$$\begin{aligned}\mathrm{Union}(b, s) &= 1 - [(1 - b) \times (1 - s)] \\ &= b + s - (b \times s)\end{aligned}$$

where *b* and *s* shall be the backdrop and source values to be composited.

NOTE 1 This is a generalization of the conventional concept of union for opaque shapes, and it can be thought of as an "inverted multiplication"—a multiplication with the inputs and outputs complemented. The result tends toward 1.0: if either input is 1.0, the result is 1.0.

The result shape and opacity shall be given by

$$f_r = \text{Union}(f_b, f_s)$$

$$q_r = \frac{\text{Union}(f_b \times q_b,\ f_s \times q_s)}{f_r}$$

where the variables have the meanings shown in Table 139.

Table 139 – Variables used in the result shape and opacity formulas

Variable	**Meaning**
f_r	Result shape
f_b	Backdrop shape
f_s	Source shape
q_r	Result opacity
q_b	Backdrop opacity
q_s	Source opacity

These formulas shall be interpreted as follows:

- The result shape shall be the union of the backdrop and source shapes.
- The result opacity shall be the union of the backdrop and source opacities, weighted by their respective shapes. The result shall then be divided by (normalized by) the result shape.

NOTE 2 Since alpha is just the product of shape and opacity, it can easily be shown that

$$\alpha_r = \text{Union}(\alpha_b, \alpha_s)$$

This formula can be used whenever the independent shape and opacity are not needed.

11.3.8 Summary of Basic Compositing Computations

This sub-clause is a summary of all the computations presented in this sub-clause. They are given in an order such that no variable is used before it is computed; also, some of the formulas have been rearranged to simplify them. See Tables 135, 138, and 139 for the meanings of the variables used in these formulas.

$$\begin{aligned}\text{Union}(b, s) &= 1 - [(1 - b) \times (1 - s)] \\ &= b + s - (b \times s)\end{aligned}$$

$$\begin{aligned}f_s &= f_j \times f_m \times f_k \\ q_s &= q_j \times q_m \times q_k \\ f_r &= \text{Union}(f_b, f_s)\end{aligned}$$

$$\alpha_b = f_b \times q_b$$
$$\alpha_s = f_s \times q_s$$
$$\alpha_r = \mathrm{Union}(\alpha_b, \alpha_s)$$

$$q_r = \frac{\alpha_r}{f_r}$$

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times [(1 - \alpha_b) \times C_s + \alpha_b \times B(C_b, C_s)]$$

11.4 Transparency Groups

11.4.1 General

A *transparency group* is a sequence of consecutive objects in a transparency stack that shall be collected together and composited to produce a single colour, shape, and opacity at each point. The result shall then be treated as if it were a single object for subsequent compositing operations. Groups may be nested within other groups to form a tree-structured group hierarchy.

NOTE This facilitates creating independent pieces of artwork, each composed of multiple objects, and then combining them, possibly with additional transparency effects applied during the combination.

The objects contained within a group shall be treated as a separate transparency stack called the *group stack*. The objects in the stack shall be composited against an initial backdrop (discussed later), producing a composite colour, shape, and opacity for the group as a whole. The result is an object whose shape is the union of the shapes of its constituent objects and whose colour and opacity are the result of the compositing operations. This object shall then be composited with the group's backdrop in the usual way.

In addition to its computed colour, shape, and opacity, the group as a whole may have several further attributes:

- All of the input variables that affect the compositing computation for individual objects may also be applied when compositing the group with its backdrop. These variables include mask and constant shape, mask and constant opacity, and blend mode.
- The group may be *isolated* or *non-isolated*, which shall determine the initial backdrop against which its stack is composited.
- The group may be *knockout* or *non-knockout*, which shall determine whether the objects within its stack are composited with one another or only with the group's backdrop.
- An isolated group may specify its own blending colour space, independent of that of the group's backdrop.
- Instead of being composited onto the current page, a group's results may be used as a source of shape or opacity values for creating a *soft mask* (see "Soft Masks").

11.4.2 Notation for Group Compositing Computations

This sub-clause introduces some notation for dealing with group compositing. Subsequent sub-clauses describe the group compositing formulas for a non-isolated, non-knockout group and the special properties of isolated and knockout groups.

Since we are now dealing with multiple objects at a time, it is useful to have some notation for distinguishing among them. Accordingly, the variables introduced earlier are altered to include a second-level subscript denoting an object's position in the transparency stack.

C_{s_i} stands for the source colour of the *i*th object in the stack. The subscript 0 represents the initial backdrop; subscripts 1 to *n* denote the bottommost to topmost objects in an n-element stack. In addition, the subscripts *b* and *r* are dropped from the variables C_b, f_β, q_β, α_β, C_r, f_ρ, q_ρ, and α_ρ; other variables retain their mnemonic subscripts.

These conventions permit the compositing formulas to be restated as recurrence relations among the elements of a stack. For instance, the result of the colour compositing computation for object *i* is denoted by C_i (formerly C_r). This computation takes as one of its inputs the immediate backdrop colour, which is the result of the colour compositing computation for object *i* – 1; this is denoted by C_{i-1} (formerly C_b).

The revised formulas for a simple *n*-element stack (not including any groups) shall be, for $i = 1, \ldots, n$:

$$\begin{aligned} f_{s_i} &= f_{j_i} \times f_{m_i} \times f_{k_i} \\ q_{s_i} &= q_{j_i} \times q_{m_i} \times q_{k_i} \end{aligned}$$

$$\begin{aligned} \alpha_{s_i} &= f_{s_i} \times q_{s_i} \\ \alpha_i &= \text{Union}(\alpha_{i-1}, \alpha_{s_i}) \end{aligned}$$

$$\begin{aligned} f_i &= \text{Union}(f_{i-1}, f_{s_i}) \\ q_i &= \frac{\alpha_i}{f_i} \end{aligned}$$

$$C_i = \left(1 - \frac{\alpha_{s_i}}{\alpha_i}\right) \times C_{i-1} + \frac{\alpha_{s_i}}{\alpha_i} \times [(1 - \alpha_{i-1}) \times C_{s_i} + \alpha_{i-1} \times B_i(C_{i-1}, C_{s_i})$$

where the variables have the meanings shown in Table 140.

NOTE Compare these formulas with those shown in 11.3.8, "Summary of Basic Compositing Computations."

Table 140 – Revised variables for the basic compositing formulas

Variable	Meaning
f_{s_i}	Source shape for object *i*
f_{j_i}	Object shape for object *i*
f_{m_i}	Mask shape for object *i*
f_{k_i}	Constant shape for object *i*
f_i	Result shape after compositing object *i*

Table 140 – Revised variables for the basic compositing formulas (continued)

Variable	Meaning
q_{s_i}	Source opacity for object *i*
q_{j_i}	Object opacity for object *i*
q_{m_i}	Mask opacity for object *i*
q_{k_i}	Constant opacity for object *i*
q_i	Result opacity after compositing object *i*
α_{s_i}	Source alpha for object *i*
α_i	Result alpha after compositing object *i*
C_{s_i}	Source colour for object *i*
C_i	Result colour after compositing object *i*
$B_i\,(C_{i-1}, C_{s_i})$	Blend function for object *i*

11.4.3 Group Structure and Nomenclature

As stated earlier, the elements of a group shall be treated as a separate transparency stack, referred to as the group stack. These objects shall be composited against a selected initial backdrop and the resulting colour, shape, and opacity shall then be treated as if they belonged to a single object. The resulting object is in turn composited with the group's backdrop in the usual way.

NOTE This computation entails interpreting the stack as a tree. For an *n*-element group that begins at position *i* in the stack, it treats the next *n* objects as an *n*-element substack, whose elements are given an independent numbering of 1 to *n*. These objects are then removed from the object numbering in the parent (containing) stack and replaced by the group object, numbered *i*, followed by the remaining objects to be painted on top of the group, renumbered starting at *i* + 1. This operation applies recursively to any nested subgroups.

The term *element* (denoted E_i) refers to a member of some group; it can be either an individual object or a contained subgroup.

From the perspective of a particular element in a nested group, there are three different backdrops of interest:

- *The group backdrop* is the result of compositing all elements up to but not including the first element in the group. (This definition is altered if the parent group is a knockout group; see 11.4.6, "Knockout Groups")
- *The initial backdrop* is a backdrop that is selected for compositing the group's first element. This is either the same as the group backdrop (for a non-isolated group) or a fully transparent backdrop (for an isolated group).
- *The immediate backdrop* is the result of compositing all elements in the group up to but not including the current element.

When all elements in a group have been composited, the result shall be treated as if the group were a single object, which shall then be composited with the group backdrop. This operation shall occur whether the initial backdrop chosen for compositing the elements of the group was the group backdrop or a transparent backdrop. A conforming reader shall ensure that the backdrop's contribution to the overall result is applied only once.

11.4.4 Group Compositing Computations

The colour and opacity of a group shall be defined by the *group compositing function*:

$$\langle C, f, \alpha \rangle = \text{Composite}(C_0, \alpha_0, G)$$

where the variables have the meanings shown in Table 141.

Table 141 – Arguments and results of the group compositing function

Variable	Meaning
G	The transparency group: a compound object consisting of all elements $E_1, \ldots, E_n$ of the group—the *n* constituent objects' colours, shapes, opacities, and blend modes
C_0	Colour of the group's backdrop
C	Computed colour of the group, which shall be used as the source colour when the group is treated as an object
f	Computed shape of the group, which shall be used as the object shape when the group is treated as an object
α_0	Alpha of the group's backdrop
α	Computed alpha of the group, which shall be used as the object alpha when the group is treated as an object

NOTE 1 The opacity is not given explicitly as an argument or result of this function. Almost all of the computations use the product of shape and opacity (alpha) rather than opacity alone; therefore, it is usually convenient to work directly with shape and alpha rather than shape and opacity. When needed, the opacity can be computed by dividing the alpha by the associated shape.

The result of applying the group compositing function shall then be treated as if it were a single object, which in turn is composited with the group's backdrop according to the formulas defined in this sub-clause. In those formulas, the colour, shape, and alpha *(C, f,* and α*)* calculated by the group compositing function shall be used, respectively, as the source colour C_s, the object shape f_j, and the object alpha α_j.

The group compositing formulas for a non-isolated, non-knockout group are defined as follows:

- Initialization:

$$f_{g_0} = \alpha_{g_0} = 0.0$$

- For each group element $E_i \in G$ $(i = 1, \ldots, n)$:

$$\langle C_{s_i}, f_{j_i}, \alpha_{j_i} \rangle = \begin{cases} \text{Composite}(C_{i-1}, \alpha_{i-1}, E_i) & \text{if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape} \times \text{opacity) of } E_i & \text{otherwise} \end{cases}$$

$$f_{s_i} = f_{j_i} \times f_{m_i} \times f_{k_i}$$
$$\alpha_{s_i} = \alpha_{j_i} \times (f_{m_i} \times q_{m_i}) \times (f_{k_i} \times q_{k_i})$$

$$\begin{aligned} f_{g_i} &= \mathrm{Union}(f_{g_{i-1}}, f_{s_i}) \\ \alpha_{g_i} &= \mathrm{Union}(\alpha_{g_{i-1}}, \alpha_{s_i}) \\ \alpha_i &= \mathrm{Union}(\alpha_0, \alpha_{g_i}) \end{aligned}$$

$$C_i = \left(1 - \frac{\alpha_{s_i}}{\alpha_i}\right) \times C_{i-1} + \frac{\alpha_{s_i}}{\alpha_i} \times ((1 - \alpha_{i-1}) \times C_{s_i} + \alpha_{i-1} \times B_i(C_{i-1}, C_{s_i}))$$

- Result:

$$\begin{aligned} C &= C_n + (C_n - C_0) \times \left(\frac{\alpha_0}{\alpha_{g_n}} - \alpha_0\right) \\ f &= f_{g_n} \\ \alpha &= \alpha_{g_n} \end{aligned}$$

where the variables have the meanings shown in Table 142 (in addition to those in Table 141).

For an element *E*$_i$ that is an elementary object, the colour, shape, and alpha values C_{s_i}, f_{j_i}, and α_{j_i} are intrinsic attributes of the object. For an element that is a group, the group compositing function shall be applied recursively to the subgroup and the resulting *C*, *f*, and α values shall be used for its C_{s_i}, f_{j_i}, and α_{j_i} in the calculations for the parent group.

Table 142 – Variables used in the group compositing formulas

Variable	Meaning
E_i	Element *i* of the group: a compound variable representing the element's colour, shape, opacity, and blend mode
f_{s_i}	Source shape for element *E*$_i$
f_{j_i}	Object shape for element *E*$_i$
f_{m_i}	Mask shape for element *E*$_i$
f_{k_i}	Constant shape for element *E*$_i$
f_{g_i}	Group shape: the accumulated source shapes of group elements *E*$_1$ to *E*$_i$, excluding the initial backdrop
q_{m_i}	Mask opacity for element *E*$_i$
q_{k_i}	Constant opacity for element *E*$_i$
α_{s_i}	Source alpha for element *E*$_i$
α_{j_i}	Object alpha for element *E*$_i$: the product of its object shape and object opacity

Table 142 – Variables used in the group compositing formulas (continued)

Variable	Meaning
α_{g_i}	Group alpha: the accumulated source alphas of group elements E_1 to E_i, excluding the initial backdrop
α_i	Accumulated alpha after compositing element E_i, including the initial backdrop
C_{s_i}	Source colour for element E_i
C_i	Accumulated colour after compositing element E_i, including the initial backdrop
$B_i(C_{i-1}, C_{s_i})$	Blend function for element E_i

NOTE 2 The elements of a group are composited onto a backdrop that includes the group's initial backdrop. This is done to achieve the correct effects of the blend modes, most of which are dependent on both the backdrop and source colours being blended. This feature is what distinguishes non-isolated groups from isolated groups, discussed in the next sub-clause.

NOTE 3 Special attention should be directed to the formulas at the end that compute the final results C, f, and α, of the group compositing function. Essentially, these formulas remove the contribution of the group backdrop from the computed results. This ensures that when the group is subsequently composited with that backdrop (possibly with additional shape or opacity inputs or a different blend mode), the backdrop's contribution is included only once.

For colour, the backdrop removal is accomplished by an explicit calculation, whose effect is essentially the reverse of compositing with the **Normal** blend mode. The formula is a simplification of the following formulas, which present this operation more intuitively:

$$\phi_b = \frac{(1 - \alpha_{g_n}) \times \alpha_0}{\text{Union}(\alpha_0, \alpha_{g_n})}$$

$$C = \frac{C_n - \phi_b \times C_0}{1 - \phi_b}$$

where ϕ_b is the *backdrop fraction*, the relative contribution of the backdrop colour to the overall colour.

NOTE 4 For shape and alpha, backdrop removal is accomplished by maintaining two sets of variables to hold the accumulated values. There is never any need to compute the corresponding complete shape, f_i, that includes the backdrop contribution.

The group shape and alpha, f_{g_i} and α_{g_i}, shall accumulate only the shape and alpha of the group elements, excluding the group backdrop. Their final values shall become the group results returned by the group compositing function. The complete alpha, α_i, includes the backdrop contribution as well; its value is used in the colour compositing computations.

NOTE 5 As a result of these corrections, the effect of compositing objects as a group is the same as that of compositing them separately (without grouping) if the following conditions hold:

The group is non-isolated and has the same knockout attribute as its parent group (see 11.4.5, "Isolated Groups," and “Knockout Groups”).

When compositing the group's results with the group backdrop, the **Normal** blend mode is used, and the shape and opacity inputs are always 1.0.

11.4.5 Isolated Groups

An *isolated group* is one whose elements shall be composited onto a fully transparent initial backdrop rather than onto the group's backdrop. The resulting source colour, object shape, and object alpha for the group shall be therefore independent of the group backdrop. The only interaction with the group backdrop shall occur when the group's computed colour, shape, and alpha are composited with it.

In particular, the special effects produced by the blend modes of objects within the group take into account only the intrinsic colours and opacities of those objects; they shall not be influenced by the group's backdrop.

EXAMPLE Applying the **Multiply** blend mode to an object in the group produces a darkening effect on other objects lower in the group's stack but not on the group's backdrop.

Figure L.17 in Annex L illustrates this effect for a group consisting of four overlapping circles in a light gray colour ($C = M = Y = 0.0$; $K = 0.15$). The circles are painted within the group with opacity 1.0 in the **Multiply** blend mode; the group itself is painted against its backdrop in **Normal** blend mode. In the top row, the group is isolated and thus does not interact with the rainbow backdrop. In the bottom row, the group is non-isolated and composites with the backdrop. The figure also illustrates the difference between knockout and non-knockout groups (see "Knockout Groups").

NOTE 1 Conceptually, the effect of an isolated group could be represented by a simple object that directly specifies a colour, shape, and opacity at each point. This *flattening* of an isolated group is sometimes useful for importing and exporting fully composited artwork in applications. Furthermore, a group that specifies an explicit blending colour space shall be an isolated group.

For an isolated group, the group compositing formulas shall be altered by adding one statement to the initialization:

$$\alpha_0 = 0.0 \qquad \text{if the group is isolated}$$

That is, the initial backdrop on which the elements of the group are composited shall be transparent rather than inherited from the group's backdrop.

NOTE 2 This substitution also makes C_0 undefined, but the normal compositing formulas take care of that. Also, the result computation for C automatically simplifies to $C = C_n$, since there is no backdrop contribution to be factored out.

11.4.6 Knockout Groups

In a knockout group, each individual element shall be composited with the group's initial backdrop rather than with the stack of preceding elements in the group. When objects have binary shapes (1.0 for inside, 0.0 for outside), each object shall overwrite (knocks out) the effects of any earlier elements it overlaps within the same group. At any given point, only the topmost object enclosing the point shall contribute to the result colour and opacity of the group as a whole.

EXAMPLE Figure L.17 in Annex L about 11.4.5, "Isolated Groups," illustrates the difference between knockout and non-knockout groups. In the left column, the four overlapping circles are defined as a knockout group and therefore do not composite with each other within the group. In the right column, the circles form a non-knockout group and thus do composite with each other. In each column, the upper and lower figures depict an isolated and a non-isolated group, respectively.

NOTE 1 This model is similar to the opaque imaging model, except that the "topmost object wins" rule applies to both the colour and the opacity. Knockout groups are useful in composing a piece of artwork from a collection of overlapping objects, where the topmost object in any overlap completely obscures those beneath. At the same time, the topmost object interacts with the group's initial backdrop in the usual way, with its opacity and blend mode applied as appropriate.

The concept of knockout is generalized to accommodate fractional shape values. In that case, the immediate backdrop shall be only partially knocked out and shall be replaced by only a fraction of the result of compositing the object with the initial backdrop.

The restated group compositing formulas deal with knockout groups by introducing a new variable, *b*, which is a subscript that specifies which previous result to use as the backdrop in the compositing computations: 0 in a knockout group or *i* – 1 in a non-knockout group. When *b* = *i* – 1, the formulas simplify to the ones given in 11.4.4, "Group Compositing Computations."

In the general case, the computation shall proceed in two stages:

a) Composite the source object with the group's initial backdrop, disregarding the object's shape and using a source shape value of 1.0 everywhere. This produces unnormalized temporary alpha and colour results, α_t and C_t.

NOTE 2 For colour, this computation is essentially the same as the unsimplified colour compositing formula given in 11.3.6, "Interpretation of Alpha," but using a source shape of 1.0.

$$\alpha_t = \text{Union}(\alpha_{g_b}, q_{s_i})$$

$$C_t = (1 - q_{s_i}) \times \alpha_b \times C_b + q_{s_i} \times ((1 - \alpha_b) \times C_{s_i} + \alpha_b \times B_i(C_b, C_{s_i}))$$

b) Compute a weighted average of this result with the object's immediate backdrop, using the source shape as the weighting factor. Then normalize the result colour by the result alpha:

$$\alpha_{g_i} = (1 - f_{s_i}) \times \alpha_{g_{i-1}} + f_{s_i} \times \alpha_t$$

$$\alpha_i = \text{Union}(\alpha_0, \alpha_{g_i})$$

$$C_i = \frac{(1 - f_{s_i}) \times \alpha_{i-1} \times C_{i-1} + f_{s_i} \times C_t}{\alpha_i}$$

This averaging computation shall be performed for both colour and alpha.

NOTE 3 The preceding formulas show this averaging directly. The formulas in 11.4.8, "Summary of Group Compositing Computations," are slightly altered to use source shape and alpha rather than source shape and opacity, avoiding the need to compute a source opacity value explicitly.

NOTE 4 C_t in Group Compositing Computations is slightly different from the preceding C_t: it is premultiplied by f_{s_i}.

NOTE 5 The extreme values of the source shape produce the straightforward knockout effect. That is, a shape value of 1.0 (inside) yields the colour and opacity that result from compositing the object with the initial backdrop. A shape value of 0.0 (outside) leaves the previous group results unchanged.

The existence of the knockout feature is the main reason for maintaining a separate shape value rather than only a single alpha that combines shape and opacity. The separate shape value shall be computed in any group that is subsequently used as an element of a knockout group.

A knockout group may be isolated or non-isolated; that is, *isolated* and *knockout* are independent attributes. A non-isolated knockout group composites its topmost enclosing element with the group's backdrop. An isolated knockout group composites the element with a transparent backdrop.

NOTE 6 When a non-isolated group is nested within a knockout group, the initial backdrop of the inner group is the same as that of the outer group; it is not the immediate backdrop of the inner group. This behaviour, although perhaps unexpected, is a consequence of the group compositing formulas when b = 0.

11.4.7 Page Group

All of the elements painted directly onto a page—both top-level groups and top-level objects that are not part of any group—shall be treated as if they were contained in a transparency group *P*, which in turn is composited with a context-dependent backdrop. This group is called the *page group*.

The page group shall be treated in one of two distinctly different ways:

- Ordinarily, the page shall be imposed directly on an output medium, such as paper or a display screen. The page group shall be treated as an isolated group, whose results shall then be composited with a backdrop colour appropriate for the medium. The backdrop is nominally white, although varying according to the actual properties of the medium. However, some conforming readers may choose to provide a different backdrop, such as a checker board or grid to aid in visualizing the effects of transparency in the artwork.

- A “page” of a PDF file may be treated as a graphics object to be used as an element of a page of some other document.

EXAMPLE This case arises, for example, when placing a PDF file containing a piece of artwork produced by a drawing program into a page layout produced by a layout program. In this situation, the PDF “page” is not composited with the media colour; instead, it is treated as an ordinary transparency group, which can be either isolated or non-isolated and is composited with its backdrop in the normal way.

The remainder of this sub-clause pertains only to the first use of the page group, where it is to be imposed directly on the medium.

The colour *C* of the page at a given point shall be defined by a simplification of the general group compositing formula:

$$\begin{aligned} \langle C_g, f_g, \alpha_g \rangle &= \text{Composite}(\dot{U}, 0, P) \\ C &= (1 - \alpha_g) \times W + \alpha_g \times C_g \end{aligned}$$

where the variables have the meanings shown in Table 143. The first formula computes the colour and alpha for the group given a transparent backdrop—in effect, treating *P* as an isolated group. The second formula composites the results with the context-dependent backdrop (using the equivalent of the **Normal** blend mode).

Table 143 – Variables used in the page group compositing formulas

Variable	Meaning
P	The page group, consisting of all elements $E_1, \ldots, E_n$ in the page’s top-level stack
C_g	Computed colour of the page group
f_g	Computed shape of the page group
α_g	Computed alpha of the page group
C	Computed colour of the page
W	Initial colour of the page (nominally white but may vary depending on the properties of the medium or the needs of the application)
U	An undefined colour (which is not used, since the α_0 argument of Composite is 0)

If not otherwise specified, the page group's colour space shall be inherited from the native colour space of the output device—that is, a device colour space, such as **DeviceRGB** or **DeviceCMYK**. An explicit colour space should be specified, particularly a CIE-based space, to ensure more predictable results of the compositing computations within the page group. In this case, all page-level compositing shall be done in the specified colour space, and the entire result shall then be converted to the native colour space of the output device before being composited with the context-dependent backdrop.

NOTE This case also arises when the page is not actually being rendered but is converted to a flattened representation in an opaque imaging model, such as PostScript.

11.4.8 Summary of Group Compositing Computations

This sub-clause is a restatement of the group compositing formulas that also takes isolated groups and knockout groups into account. See Tables 141 and 142 in 11.4.4, "Group Compositing Computations," for the meanings of the variables.

$$\langle C, f, \alpha \rangle = \text{Composite}(C_0, \alpha_0, G)$$

Initialization:

$$f_{g_0} = \alpha_{g_0} = 0$$

$$\alpha_0 = 0 \qquad \text{if the group is isolated}$$

For each group element $E_i \in G$ $(i = 1, \ldots, n)$:

$$b = \begin{cases} 0 & \text{if the group is knockout} \\ i-1 & \text{otherwise} \end{cases}$$

$$\langle C_{s_i}, f_{j_i}, \alpha_{j_i} \rangle = \begin{cases} \text{Composite}(C_b, \alpha_b, E_i) & \text{if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape} \times \text{opacity) of } E_i & \text{otherwise} \end{cases}$$

$$f_{s_i} = f_{j_i} \times f_{m_i} \times f_{k_i}$$
$$\alpha_{s_i} = \alpha_{j_i} \times (f_{m_i} \times q_{m_i}) \times (f_{k_i} \times q_{k_i})$$

$$f_{g_i} = \text{Union}(f_{g_{i-1}}, f_{s_i})$$
$$\alpha_{g_i} = (1 - f_{s_i}) \times \alpha_{g_{i-1}} + (f_{s_i} - \alpha_{s_i}) \times \alpha_{g_b} + \alpha_s$$
$$\alpha_i = \text{Union}(\alpha_0, \alpha_{g_i})$$

$$C_t = (f_{s_i} - \alpha_{s_i}) \times \alpha_b \times C_b + \alpha_{s_i} \times ((1 - \alpha_b) \times C_{s_i} + \alpha_b \times B_i(C_b, C_{s_i})$$

$$C_i = \frac{(1 - f_{s_i}) \times \alpha_{i-1} \times C_{i-1} + C_t}{\alpha_i}$$

Result:

$$C = C_n + (C_n - C_0) \times \left(\frac{\alpha_0}{\alpha_{g_n}} - \alpha_0 \right)$$
$$f = f_{g_n}$$
$$\alpha = \alpha_{g_n}$$

NOTE Once again, keep in mind that these formulas are in their most general form. They can be significantly simplified when some sources of shape and opacity are not present or when shape and opacity need not be maintained separately. Furthermore, in each specific type of group (isolated or not, knockout or not), some terms of these formulas cancel or drop out. An efficient implementation should use the simplified derived formulas.

11.5 Soft Masks

11.5.1 General

As stated in earlier sub-clauses, the shape and opacity values used in compositing an object may include components called the mask shape *(f_m)* and mask opacity *(q_m)*, which may be supplied in a PDF file from a source independent of the object. Such an independent source, called a *soft mask*, defines values that may vary across different points on the page.

NOTE 1 The word *soft* emphasizes that the mask value at a given point is not limited to just 0.0 or 1.0 but can take on intermediate fractional values as well. Such a mask is typically the only means of providing position-dependent opacity values, since elementary objects do not have intrinsic opacity of their own.

NOTE 2 A mask used as a source of shape values is also called a *soft clip*, by analogy with the “hard” clipping path of the opaque imaging model (see Section 8.5.4). The soft clip is a generalization of the hard clip: a hard clip can be represented as a soft clip having shape values of 1.0 inside and 0.0 outside the clipping path. Everywhere inside a hard clipping path, the source object’s colour replaces the backdrop; everywhere outside, the backdrop shows through unchanged. With a soft clip, by contrast, a gradual transition can be created between an object and its backdrop, as in a vignette.

A mask may be defined by creating a transparency group and painting objects into it, thereby defining colour, shape, and opacity in the usual way. The resulting group may then be used to derive the mask in either of two ways, as described in the following sub-clauses.

11.5.2 Deriving a Soft Mask from Group Alpha

In the first method of defining a soft mask, the colour, shape, and opacity of a transparency group *G* shall be first computed by the usual formula

$$\langle C, f, \alpha \rangle = \text{Composite}(C_0, \alpha_0, G)$$

where C_0 and α_0 represent an arbitrary backdrop whose value does not contribute to the eventual result. The *C*, *f*, and α results shall be the group’s colour, shape, and alpha, respectively, with the backdrop factored out.

The mask value at each point shall then be derived from the alpha of the group. The alpha value shall be passed through a separately specified transfer function, allowing the masking effect to be customized.

NOTE Since the group’s colour is not used in this case, there is no need to compute it.

11.5.3 Deriving a Soft Mask from Group Luminosity

The second method of deriving a soft mask from a transparency group shall begin by compositing the group with a fully opaque backdrop of a specified colour. The mask value at any given point shall then be defined to be the luminosity of the resulting colour.

NOTE 1 This allows the mask to be derived from the shape and colour of an arbitrary piece of artwork drawn with ordinary painting operators.

The colour *C* used to create the mask from a group *G* shall be defined by

$$\begin{aligned} \langle C_g, f_g, \alpha_g \rangle &= \text{Composite}(\dot{C}_0, 1, G) \\ C &= (1 - \alpha_g) \times C_0 + \alpha_g \times C_g \end{aligned}$$

where C_0 is the selected backdrop colour.

G may be any kind of group—isolated or not, knockout or not—producing various effects on the *C* result in each case. The colour *C* shall then be converted to luminosity in one of the following ways, depending on the group's colour space:

- For CIE-based spaces, convert to the CIE 1931 *XYZ* space and use the *Y* component as the luminosity. This produces a colourimetrically correct luminosity.

NOTE 2 In the case of a PDF **CalRGB** space, the formula is

$$Y = Y_A \times A^{G_R} + Y_B \times B^{G_G} + Y_C \times C^{G}$$

using components of the **Gamma** and **Matrix** entries of the colour space dictionary (see Table 64 in "CIE-Based Colour Spaces"). An analogous computation applies to other CIE-based colour spaces.

- For device colour spaces, convert the colour to **DeviceGray** by implementation-defined means and use the resulting gray value as the luminosity, with no compensation for gamma or other colour calibration.

NOTE 3 This method makes no pretence of colourimetric correctness; it merely provides a numerically simple means to produce continuous-tone mask values. The following are formulas for converting from **DeviceRGB** and **DeviceCMYK**, respectively:

$$Y = 0.30 \times R + 0.59 \times G + 0.11 \times B$$

$$\begin{aligned} Y = {} & 0.30 \times (1 - C) \times (1 - K) \\ & + 0.59 \times (1 - M) \times (1 - K) \\ & + 0.11 \times (1 - Y) \times (1 - K) \end{aligned}$$

Following this conversion, the result shall be passed through a separately specified transfer function, allowing the masking effect to be customized.

NOTE 4 The backdrop colour most likely to be useful is black, which causes any areas outside the group's shape to have zero luminosity values in the resulting mask. If the contents of the group are viewed as a positive mask, this produces the results that would be expected with respect to points outside the shape.

11.6 Specifying Transparency in PDF

11.6.1 General

The preceding sub-clauses have presented the transparent imaging model at an abstract level, with little mention of its representation in PDF. This sub-clause describes the facilities available for specifying transparency in PDF.

11.6.2 Specifying Source and Backdrop Colours

Single graphics objects, as defined in “Graphics Objects”, shall be treated as elementary objects for transparency compositing purposes (subject to special treatment for text objects, as described in “Text Knockout”). That is, all of a given object shall be considered to be one element of a transparency stack. Portions of an object shall not be composited with one another, even if they are described in a way that would seem to cause overlaps (such as a self-intersecting path, combined fill and stroke of a path, or a shading pattern containing an overlap or fold-over). An object’s source colour C_s, used in the colour compositing formula, shall be specified in the same way as in the opaque imaging model: by means of the current colour in the graphics state or the source samples in an image. The backdrop colour C_b shall be the result of previous painting operations.

11.6.3 Specifying Blending Colour Space and Blend Mode

The blending colour space shall be an attribute of the transparency group within which an object is painted; its specification is described in 11.6.6, "Transparency Group XObjects." The page as a whole shall also be treated as a group, the *page group* (see “Page Group”), with a colour space attribute of its own. If not otherwise specified, the page group’s colour space shall be inherited from the native colour space of the output device.

The blend mode $B(C_b, C_s)$ shall be determined by the *current blend mode* parameter in the graphics state (see “Graphics State”), which is specified by the **BM** entry in a graphics state parameter dictionary (“Graphics State Parameter Dictionaries”). Its value shall be either a name object, designating one of the standard blend modes listed in Tables 136 and 137 in 11.3.5, "Blend Mode," or an array of such names. In the latter case, the application shall use the first blend mode in the array that it recognizes (or **Normal** if it recognizes none of them).

NOTE New blend modes may be introduced in the future, and conforming readers that do not recognize them should have reasonable fallback behavior.

The current blend mode shall always apply to process colour components; but only sometimes may apply to spot colorants, see 11.7.4.2, "Blend Modes and Overprinting," for details.

11.6.4 Specifying Shape and Opacity

11.6.4.1 General

As discussed under 11.3.7.2, "Source Shape and Opacity," the shape *(f)* and opacity *(q)* values used in the compositing computation shall come from one or more of the following sources:

- The intrinsic shape (f_j) and opacity (q_j) of the object being composited
- A separate shape (f_m) or opacity (q_m) mask independent of the object itself
- A scalar shape (f_k) or opacity (q_k) constant to be added at every point

The following sub-clauses describe how each of these shape and opacity sources shall be specified in PDF.

11.6.4.2 Object Shape and Opacity

The shape value f_j of an object painted with PDF painting operators shall be defined as follows:

- For objects defined by a path or a glyph and painted in a uniform colour with a path-painting or text-showing operator (“Path-Painting Operators”, and “Text-Showing Operators”), the shape shall always be 1.0 inside and 0.0 outside the path.
- For images (“Images”), the shape shall be 1.0 inside the image rectangle and 0.0 outside it. This may be further modified by an explicit or colour key mask (“Explicit Masking” and “Colour Key Masking”).
- For image masks (“Stencil Masking”), the shape shall be 1.0 for painted areas and 0.0 for masked areas.
- For objects painted with a tiling pattern (“Tiling Patterns”) or a shading pattern (“Shading Patterns”), the shape shall be further constrained by the objects that define the pattern (see “Patterns and Transparency”).
- For objects painted with the **sh** operator (“Shading Operator”), the shape shall be 1.0 inside and 0.0 outside the bounds of the shading’s painting geometry, disregarding the **Background** entry in the shading dictionary (see “Shading Dictionaries”).

All elementary objects shall have an intrinsic opacity q_j of 1.0 everywhere. Any desired opacity less than 1.0 shall be applied by means of an opacity mask or constant, as described in the following sub-clauses.

11.6.4.3 Mask Shape and Opacity

At most one mask input—called a *soft mask*, or *alpha mask*—shall be provided to any PDF compositing operation. The mask may serve as a source of either shape (f_m) or opacity (q_m) values, depending on the setting of the *alpha source* parameter in the graphics state (see “Graphics State”). This is a boolean flag, set with the **AIS** (“alpha is shape”) entry in a graphics state parameter dictionary (“Graphics State Parameter Dictionaries”): **true** if the soft mask contains shape values, **false** for opacity.

The soft mask shall be specified in one of the following ways:

- The *current soft mask* parameter in the graphics state, set with the **SMask** entry in a graphics state parameter dictionary, contains a *soft-mask dictionary* (see “Soft-Mask Dictionaries”) defining the contents of the mask. The name **None** may be specified in place of a soft-mask dictionary, denoting the absence of a soft mask. In this case, the mask shape or opacity shall be implicitly 1.0 everywhere.
- An image XObject may contain its own *soft-mask image* in the form of a subsidiary image XObject in the **SMask** entry of the image dictionary (see “Image Dictionaries”). This mask, if present, shall override any explicit or colour key mask specified by the image dictionary’s **Mask** entry. Either form of mask in the image dictionary shall override the current soft mask in the graphics state.
- An image XObject that has a **JPXDecode** filter as its data source may specify an **SMaskInData** entry, indicating that the soft mask is embedded in the data stream (see “JPXDecode Filter”).

NOTE The current soft mask in the graphics state is intended to be used to clip only a single object at a time (either an elementary object or a transparency group). If a soft mask is applied when painting two or more overlapping objects, the effect of the mask multiplies with itself in the area of overlap (except in a knockout group), producing a result shape or opacity that is probably not what is intended. To apply a soft mask to multiple objects, it is usually best to define the objects as a transparency group and apply the mask to the group as a whole. These considerations also apply to the current alpha constant (see the next sub-clause).

11.6.4.4 Constant Shape and Opacity

The *current alpha constant* parameter in the graphics state (see “Graphics State”) shall be two scalar values—one for strokes and one for all other painting operations—to be used for the constant shape *(f_k)* or constant opacity *(q_k)* component in the colour compositing formulas.

NOTE 1 This parameter is analogous to the current colour used when painting elementary objects.

The nonstroking alpha constant shall also be applied when painting a transparency group’s results onto its backdrop.

The stroking and nonstroking alpha constants shall be set, respectively, by the **CA** and **ca** entries in a graphics state parameter dictionary (see “Graphics State Parameter Dictionaries”). As described previously for the soft mask, the alpha source flag in the graphics state shall determine whether the alpha constants are interpreted as shape values *(***true***)* or opacity values (**false**).

NOTE 2 The note at the end of 11.6.4.3, "Mask Shape and Opacity," applies to the current alpha constant parameter as well as the current soft mask.

11.6.5 Specifying Soft Masks

11.6.5.1 General

As noted under 11.6.4.3, "Mask Shape and Opacity," soft masks for use in compositing computations may be specified in one of the following ways:

- As a soft-mask dictionary in the current soft mask parameter of the graphics state; see 11.6.5.2, "Soft-Mask Dictionaries," for more details.

- As a soft-mask image associated with a sampled image; see 11.6.5.3, "Soft-Mask Images," for more details.

- (*PDF 1.5*) as a mask channel embedded in JPEG2000 encoded data; see “JPXDecode Filter”, and the **SMaskInData** entry of Table 89 for more details.

11.6.5.2 Soft-Mask Dictionaries

The most common way of defining a soft mask is with a *soft-mask dictionary* specified as the current soft mask in the graphics state (see “Graphics State”). Table 144 shows the contents of this type of dictionary.

The mask values shall be derived from those of a transparency group, using one of the two methods described in 11.5.2, "Deriving a Soft Mask from Group Alpha," and 11.5.3, "Deriving a Soft Mask from Group Luminosity." The group shall be defined by a transparency group XObject (see “Transparency Group XObjects”) designated by the **G** entry in the soft-mask dictionary. The **S** (subtype) entry shall specify which of the two derivation methods to use:

- If the subtype is **Alpha**, the transparency group XObject **G** shall be evaluated to compute a group alpha only. The colours of the constituent objects shall be ignored and the colour compositing computations shall not be performed. The transfer function **TR** shall then be applied to the computed group alpha to produce the mask values. Outside the bounding box of the transparency group, the mask value shall be the result of applying the transfer function to the input value 0.0.

- If the subtype is **Luminosity**, the transparency group XObject **G** shall be composited with a fully opaque backdrop whose colour is everywhere defined by the soft-mask dictionary’s **BC** entry. The computed result colour shall then be converted to a single-component luminosity value, and the transfer function **TR** shall be applied to this luminosity to produce the mask values. Outside the transparency group’s bounding box, the mask value shall be derived by transforming the **BC** colour to luminosity and applying the transfer function to the result.

The mask's coordinate system shall be defined by concatenating the transformation matrix specified by the **Matrix** entry in the transparency group's form dictionary (see "Form Dictionaries") with the current transformation matrix at the moment the soft mask is established in the graphics state with the **gs** operator.

In a transparency group XObject that defines a soft mask, spot colour components shall never be available, even if they are available in the group or page on which the soft mask is used. If the group XObject's content stream specifies a ***Separation*** or ***DeviceN*** colour space that uses spot colour components, the alternate colour space shall be substituted (see "Separation Colour Spaces" and "DeviceN Colour Spaces").

Table 144 – Entries in a soft-mask dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Mask** for a soft-mask dictionary.
S	name	*(Required)* A subtype specifying the method to be used in deriving the mask values from the transparency group specified by the **G** entry: **Alpha** The group's computed alpha shall be used, disregarding its colour (see "Deriving a Soft Mask from Group Alpha"). **Luminosity** The group's computed colour shall be converted to a single-component luminosity value (see "Deriving a Soft Mask from Group Luminosity").
G	stream	*(Required)* A transparency group XObject (see "Transparency Group XObjects") to be used as the source of alpha or colour values for deriving the mask. If the subtype **S** is **Luminosity**, the group attributes dictionary shall contain a **CS** entry defining the colour space in which the compositing computation is to be performed.
BC	array	*(Optional)* An array of component values specifying the colour to be used as the backdrop against which to composite the transparency group XObject **G**. This entry shall be consulted only if the subtype **S** is **Luminosity**. The array shall consist of *n* numbers, where *n* is the number of components in the colour space specified by the **CS** entry in the group attributes dictionary (see "Transparency Group XObjects"). Default value: the colour space's initial value, representing black.
TR	function or name	*(Optional)* A function object (see "Functions") specifying the transfer function to be used in deriving the mask values. The function shall accept one input, the computed group alpha or luminosity (depending on the value of the subtype **S**), and shall return one output, the resulting mask value. The input shall be in the range 0.0 to 1.0. The computed output shall be in the range 0.0 to 1.0; if it falls outside this range, it shall be forced to the nearest valid value. The name **Identity** may be specified in place of a function object to designate the identity function. Default value: **Identity**.

11.6.5.3 Soft-Mask Images

The second way to define a soft mask is by associating a *soft-mask image* with an image XObject. This is a subsidiary image XObject specified in the **SMask** entry of the parent XObject's image dictionary (see "Image Dictionaries"). Entries in the subsidiary image dictionary for such a soft-mask image shall have the same format and meaning as in that of an ordinary image XObject (as described in Table 89 in "Image Dictionaries"), subject to the restrictions listed in Table 145. This type of image dictionary may contain an additional entry, **Matte**.

When an image is accompanied by a soft-mask image, it is sometimes advantageous for the image data to be *preblended* with some background colour, called the *matte colour*. Each image sample represents a weighted average of the original source colour and the matte colour, using the corresponding mask sample as the weighting factor. (This is a generalization of a technique commonly called *premultiplied alpha*.)

If the image data is preblended, the matte colour shall be specified by a **Matte** entry in the soft-mask image dictionary (see Table 145). The preblending computation, performed independently for each component, shall be

$$c' = m + \alpha \times (c - m)$$

where

c′ is the value to be provided in the image source data

c is the original image component value

m is the matte colour component value

α is the corresponding mask sample

This computation shall use actual colour component values, with the effects of the ***Filter*** and ***Decode*** transformations already performed. The computation shall be the same whether the colour space is additive or subtractive.

Table 145 – Restrictions on the entries in a soft-mask image dictionary

Key	Restriction
Type	If present, shall be **XObject**.
Subtype	Shall be **Image**.
Width	If a **Matte** entry (see Table 146) is present, shall be the same as the **Width** value of the parent image; otherwise independent of it. Both images shall be mapped to the unit square in user space (as are all images), regardless of whether the samples coincide individually.
Height	Same considerations as for **Width**.
ColorSpace	Required; shall be **DeviceGray**.
BitsPerComponent	Required.
Intent	Ignored.
ImageMask	Shall be **false** or absent.
Mask	Shall be absent.
SMask	Shall be absent.
Decode	Default value: [0 1].
Interpolate	Optional.
Alternates	Ignored.
Name	Ignored.
StructParent	Ignored.
ID	Ignored.
OPI	Ignored.

Table 146 – Additional entry in a soft-mask image dictionary

Key	Type	Value
Matte	array	*(Optional; PDF 1.4)* An array of component values specifying the matte colour with which the image data in the parent image shall have been preblended. The array shall consist of *n* numbers, where *n* is the number of components in the colour space specified by the **ColorSpace** entry in the parent image's image dictionary; the numbers shall be valid colour components in that colour space. If this entry is absent, the image data shall not be preblended.

When preblended image data is used in transparency blending and compositing computations, the results shall be the same as if the original, unblended image data were used and no matte colour were specified. In particular, the inputs to the blend function shall be the original colour values. To derive *c* from *c*′, the conforming reader may sometimes need to invert the formula shown previously. The resulting *c* value shall lie within the range of colour component values for the image colour space.

The preblending computation shall be done in the colour space specified by the parent image's **ColorSpace** entry. This is independent of the group colour space into which the image may be painted. If a colour conversion is required, inversion of the preblending shall precede the colour conversion. If the image colour space is an **Indexed** space (see "Indexed Colour Spaces"), the colour values in the colour table (not the index values themselves) shall be preblended.

11.6.6 Transparency Group XObjects

A transparency group is represented in PDF as a special type of group XObject (see "Group XObjects") called a *transparency group XObject*. A group XObject is in turn a type of form XObject, distinguished by the presence of a **Group** entry in its form dictionary (see "Form Dictionaries"). The value of this entry is a subsidiary *group attributes dictionary* defining the properties of the group. The format and meaning of the dictionary's contents shall be determined by its *group subtype*, which is specified by the dictionary's **S** entry. The entries for a transparency group (subtype **Transparency**) are shown in Table 147.

A page object (see "Page Objects") may also have a ***Group*** entry, whose value is a group attributes dictionary specifying the attributes of the page group (see "Page Group"). Some of the dictionary entries are interpreted slightly differently for a page group than for a transparency group XObject; see their descriptions in the table for details.

Table 147 – Additional entries specific to a transparency group attributes dictionary

Key	Type	Value
S	name	*(Required)* The *group subtype*, which identifies the type of group whose attributes this dictionary describes; shall be **Transparency** for a transparency group.

Table 147 – Additional entries specific to a transparency group attributes dictionary (continued)

Key	Type	Value
CS	name or array	*(Sometimes required)* The group colour space, which is used for the following purposes: • As the colour space into which colours shall be converted when painted into the group • As the blending colour space in which objects shall be composited within the group (see “Blending Colour Space”) • As the colour space of the group as a whole when it in turn is painted as an object onto its backdrop The group colour space shall be any device or CIE-based colour space that treats its components as independent additive or subtractive values in the range 0.0 to 1.0, subject to the restrictions described in 11.3.4, "Blending Colour Space." These restrictions exclude **Lab** and lightness-chromaticity **ICCBased** colour spaces, as well as the special colour spaces **Pattern**, **Indexed**, **Separation**, and **DeviceN**. Device colour spaces shall be subject to remapping according to the **DefaultGray**, **DefaultRGB**, and **DefaultCMYK** entries in the **ColorSpace** subdictionary of the current resource dictionary (see “Default Colour Spaces”). Ordinarily, the **CS** entry may be present only for isolated transparency groups (those for which **I** is **true**), and even then it is optional. However, this entry shall be present in the group attributes dictionary for any transparency group XObject that has no parent group or page from which to inherit—in particular, one that is the value of the **G** entry in a soft-mask dictionary of subtype **Luminosity** (see “Soft-Mask Dictionaries”). Additionally, the **CS** entry may be present in the group attributes dictionary associated with a page object, even if **I** is **false** or absent. In the normal case in which the page is imposed directly on the output medium, the page group is effectively isolated regardless of the **I** value, and the specified **CS** value shall therefore be honoured. But if the page is in turn used as an element of some other page and if the group is non-isolated, **CS** shall be ignored and the colour space shall be inherited from the actual backdrop with which the page is composited (see “Page Group”). Default value: the colour space of the parent group or page into which this transparency group is painted. (The parent’s colour space in turn may be either explicitly specified or inherited.) For a transparency group XObject used as an annotation appearance (see “Appearance Streams”), the default colour space shall be inherited from the page on which the annotation appears.
I	boolean	*(Optional)* A flag specifying whether the transparency group is isolated (see “Isolated Groups”). If this flag is **true**, objects within the group shall be composited against a fully transparent initial backdrop; if **false**, they shall be composited against the group’s backdrop. Default value: **false**. In the group attributes dictionary for a page, the interpretation of this entry shall be slightly altered. In the normal case in which the page is imposed directly on the output medium, the page group is effectively isolated and the specified **I** value shall be ignored. But if the page is in turn used as an element of some other page, it shall be treated as if it were a transparency group XObject; the **I** value shall be interpreted in the normal way to determine whether the page group is isolated.

Table 147 – Additional entries specific to a transparency group attributes dictionary (continued)

Key	Type	Value
K	boolean	*(Optional)* A flag specifying whether the transparency group is a knockout group (see "Knockout Groups"). If this flag is **false**, later objects within the group shall be composited with earlier ones with which they overlap; if **true**, they shall be composited with the group's initial backdrop and shall overwrite ("knock out") any earlier overlapping objects. Default value: **false**.

The transparency group XObject's content stream shall define the graphics objects belonging to the group. When applied to a transparency group XObject, the **Do** operator shall execute its content stream and shall composite the resulting group colour, shape, and opacity into the group's parent group or page as if they had come from an elementary graphics object. **Do** shall perform the following actions in addition to the normal ones for a form XObject (as described in "Form XObjects"):

- If the transparency group is non-isolated (the value of the **I** entry in its group attributes dictionary is **false***)*, its initial backdrop, within the bounding box specified by the XObject's **BBox** entry, shall be defined to be the accumulated colour and alpha of the parent group or page—that is, the result of everything that has been painted in the parent up to that point. However, if the parent is a knockout group, the initial backdrop shall be the same as that of the parent. If the group is isolated (**I** is **true**), its initial backdrop shall be defined to be transparent.

- Before execution of the transparency group XObject's content stream, the current blend mode in the graphics state shall be initialized to **Normal**, the current stroking and nonstroking alpha constants to 1.0, and the current soft mask to **None**.

NOTE 1 The purpose of initializing these graphics state parameters at the beginning of execution is to ensure that they are not applied twice: once when member objects are painted into the group and again when the group is painted into the parent group or page.

- Objects painted by operators in the transparency group XObject's content stream shall be composited into the group according to the rules described in 11.3.3, "Basic Compositing Formula." The knockout flag (**K**) in the group attributes dictionary and the transparency-related parameters of the graphics state shall be honoured during this computation.

- If a group colour space (**CS**) is specified in the group attributes dictionary, all painting operators shall convert source colours to that colour space before compositing objects into the group, and the resulting colour at each point shall be interpreted in that colour space. If no group colour space is specified, the prevailing colour space shall be dynamically inherited from the parent group or page. (If not otherwise specified, the page group's colour space shall be inherited from the native colour space of the output device.)

- After execution of the transparency group XObject's content stream, the graphics state shall revert to its former state before the invocation of the ***Do*** operator (as it does for any form XObject). The group's shape—the union of all objects painted into the group, clipped by the group XObject's bounding box—shall then be painted into the parent group or page, using the group's accumulated colour and opacity at each point.

If the ***Do*** operator is invoked more than once for a given transparency group XObject, each invocation shall be treated as a separate transparency group. That is, the result shall be as if the group were independently composited with the backdrop on each invocation.

NOTE 2 Applications that perform caching of rendered form XObjects shall take this requirement into account.

The actions described previously shall occur only for a transparency group XObject—a form XObject having a **Group** entry that designates a group attributes subdictionary whose group subtype (**S**) is **Transparency**. An ordinary form XObject—one having no **Group** entry—shall not be subject to any grouping behaviour for transparency purposes. That is, the graphics objects it contains shall be composited individually, just as if they were painted directly into the parent group or page.

11.6.7 Patterns and Transparency

In the transparent imaging model, the graphics objects making up the pattern cell of a tiling pattern (see “Tiling Patterns”) may include transparent objects and transparency groups. Transparent compositing may occur both within the pattern cell and between it and the backdrop wherever the pattern is painted. Similarly, a shading pattern (“Shading Patterns”) composites with its backdrop as if the shading dictionary were applied with the **sh** operator.

In both cases, the pattern definition shall be treated as if it were implicitly enclosed in a non-isolated transparency group: a non-knockout group for tiling patterns, a knockout group for shading patterns. The definition shall not inherit the current values of the graphics state parameters at the time it is evaluated; those parameters shall take effect only when the resulting pattern is later used to paint an object. Instead, the graphics state parameters shall be initialized as follows:

- As always for transparency groups, those parameters related to transparency (blend mode, soft mask, and alpha constant) shall be initialized to their standard default values.
- All other parameters shall be initialized to their values at the beginning of the content stream (such as a page or a form XObject) in which the pattern shall be defined as a resource. (This is the normal behaviour for all patterns, in both the opaque and transparent imaging models.)
- In the case of a shading pattern, the parameter values may be augmented by the contents of the **ExtGState** entry in the pattern dictionary (see “Shading Patterns”). Only those parameters that affect the **sh** operator, such as the current transformation matrix and rendering intent, shall be used. Parameters that affect path-painting operators shall not be used, since the execution of **sh** does not entail painting a path.
- If the shading dictionary has a **Background** entry, the pattern’s implicit transparency group shall be filled with the specified background colour before the **sh** operator is invoked.

When the pattern is later used to paint a graphics object, the colour, shape, and opacity values resulting from the evaluation of the pattern definition shall be used as the object’s source colour (C_s), object shape (f_j), and object opacity (q_j) in the transparency compositing formulas. This painting operation is subject to the values of the graphics state parameters in effect at the time, just as in painting an object with a constant colour.

NOTE 1 Unlike the opaque imaging model, in which the pattern cell of a tiling pattern may be evaluated once and then replicated indefinitely to fill the painted area, the effect in the general transparent case is as if the pattern definition were reexecuted independently for each tile, taking into account the colour of the backdrop at each point. However, in the common case in which the pattern consists entirely of objects painted with the **Normal** blend mode, this behaviour can be optimized by treating the pattern cell as if it were an isolated group. Since in this case the results depend only on the colour, shape, and opacity of the pattern cell and not on those of the backdrop, the pattern cell can be evaluated once and then replicated, just as in opaque painting.

NOTE 2 In a raster-based implementation of tiling, all tiles should be treated as a single transparency group. This avoids artifacts due to multiple marking of pixels along the boundaries between adjacent tiles.

The foregoing discussion applies to both coloured (**PaintType** 1) and uncoloured (**PaintType** 2) tiling patterns. In the latter case, the restriction that an uncoloured pattern’s definition shall not specify colours extends as well to any transparency group that the definition may include. There are no corresponding restrictions, however, on specifying transparency-related parameters in the graphics state.

11.7 Colour Space and Rendering Issues

11.7.1 General

This sub-clause describes the interactions between transparency and other aspects of colour specification and rendering in the PDF imaging model.

11.7.2 Colour Spaces for Transparency Groups

As discussed in 11.6.6, "Transparency Group XObjects," a transparency group shall either have an explicitly declared colour space of its own or inherit that of its parent group. In either case, the colours of source objects within the group shall be converted to the group's colour space, if necessary, and all blending and compositing computations shall be done in that space (see "Blending Colour Space"). The resulting colours shall then be interpreted in that colour space when the group is subsequently composited with its backdrop.

NOTE 1 Under this arrangement, it is envisioned that all or most of a given piece of artwork will be created in a single colour space—most likely, the working colour space of the application generating it. The use of multiple colour spaces typically will arise only when assembling independently produced artwork onto a page. After all the artwork has been placed on the page, the conversion from the group's colour space to the page's device colour space will be done as the last step, without any further transparency compositing. The transparent imaging model does not require that this convention be followed, however; the reason for adopting it is to avoid the loss of colour information and the introduction of errors resulting from unnecessary colour space conversions.

Only an isolated group may have an explicitly declared colour space of its own. Non-isolated groups shall inherit their colour space from the parent group (subject to special treatment for the page group, as described in "Page Group").

NOTE 2 This is because the use of an explicit colour space in a non-isolated group would require converting colours from the backdrop's colour space to that of the group in order to perform the compositing computations. Such conversion may not be possible (since some colour conversions can be performed only in one direction), and even if possible, it would entail an excessive number of colour conversions.

NOTE 3 The choice of a group colour space has significant effects on the results that are produced:

As noted in 11.3.4, "Blending Colour Space," the results of compositing in a device colour space is device-dependent. For the compositing computations to work in a device-independent way, the group's colour space should be CIE-based.

A consequence of choosing a CIE-based group colour space is that only CIE-based spaces can be used to specify the colours of objects within the group. This is because conversion from device to CIE-based colours is not possible in general; the defined conversions work only in the opposite direction. See further discussion subsequently.

The compositing computations and blend functions generally compute linear combinations of colour component values, on the assumption that the component values themselves are linear. For this reason, it is usually best to choose a group colour space that has a linear gamma function. If a nonlinear colour space is chosen, the results are still well-defined, but the appearance may not match the user's expectations.

NOTE 4 The CIE-based *sRGB* colour space (see "CIE-Based Colour Spaces") is nonlinear and hence may be unsuitable for use as a group colour space.

NOTE 5 Implementations of the transparent imaging model should use as much precision as possible in representing colours during compositing computations and in the accumulated group results. To minimize the accumulation of roundoff errors and avoid additional errors arising from the use of linear group colour spaces, more precision is needed for intermediate results than is typically used to represent either the original source data or the final rasterized results.

If a group's colour space—whether specified explicitly or inherited from the parent group—is CIE-based, any use of device colour spaces for painting objects shall be subject to special treatment. Device colours cannot be painted directly into such a group, since there is no generally defined method for converting them to the CIE-based colour space. This problem arises in the following cases:

- **DeviceGray**, **DeviceRGB**, and **DeviceCMYK** colour spaces, unless remapped to default CIE-based colour spaces (see “Default Colour Spaces”)
- Operators (such as **rg**) that specify a device colour space implicitly, unless that space is remapped
- Special colour spaces whose base or underlying space is a device colour space, unless that space is remapped

The default colour space remapping mechanism should always be employed when defining a transparency group whose colour space is CIE-based. If a device colour is specified and is not remapped, it shall be converted to the CIE-based colour space in an implementation-dependent fashion, producing unpredictable results.

NOTE 6 The foregoing restrictions do not apply if the group’s colour space is implicitly converted to ***DeviceCMYK***, as discussed in “Implicit Conversion of CIE-Based Colour Spaces”.

11.7.3 Spot Colours and Transparency

The foregoing discussion of colour spaces has been concerned with *process colours*—those produced by combinations of an output device’s process colorants. Process colours may be specified directly in the device’s native colour space (such as **DeviceCMYK**), or they may be produced by conversion from some other colour space, such as a CIE-based (**CalRGB** or **ICCBased**) space. Whatever means is used to specify them, process colours shall be subject to conversion to and from the group’s colour space.

A *spot colour* is an additional colour component, independent of those used to produce process colours. It may represent either an additional separation to be produced or an additional colorant to be applied to the composite page (see “Separation Colour Spaces” and “DeviceN Colour Spaces”). The colour component value, or *tint*, for a spot colour specifies the concentration of the corresponding spot colorant. Tints are conventionally represented as subtractive, rather than additive, values.

Spot colours are inherently device-dependent and are not always available. In the opaque imaging model, each use of a spot colour component in a **Separation** or **DeviceN** colour space is accompanied by an *alternate colour space* and a *tint transformation function* for mapping tint values into that space. This enables the colour to be approximated with process colorants when the corresponding spot colorant is not available on the device.

Spot colours can be accommodated straightforwardly in the transparent imaging model (except for issues relating to overprinting, discussed in “Overprinting and Transparency”). When an object is painted transparently with a spot colour component that is available in the output device, that colour shall be composited with the corresponding spot colour component of the backdrop, independently of the compositing that is performed for process colours. A spot colour retains its own identity; it shall not be subject to conversion to or from the colour space of the enclosing transparency group or page. If the object is an element of a transparency group, one of two things shall happen:

- The group shall maintain a separate colour value for each spot colour component, independently of the group’s colour space. In effect, the spot colour passes directly through the group hierarchy to the device, with no colour conversions performed. However, it shall still be subject to blending and compositing with other objects that use the same spot colour.
- The spot colour shall be converted to its alternate colour space. The resulting colour shall then be subject to the usual compositing rules for process colours. In particular, spot colours shall not be available in a transparency group XObject that is used to define a soft mask; the alternate colour space shall always be substituted in that case.

Only a single shape value and opacity value shall be maintained at each point in the computed group results; they shall apply to both process and spot colour components. In effect, every object shall be considered to paint every existing colour component, both process and spot. Where no value has been explicitly specified for a given component in a given object, an additive value of 1.0 (or a subtractive tint value of 0.0) shall be assumed. For instance, when painting an object with a colour specified in a **DeviceCMYK** or **ICCBased** colour space, the process colour components shall be painted as specified and the spot colour components shall be painted with an additive value of 1.0. Likewise, when painting an object with a colour specified in a **Separation** colour space, the named spot colour shall be painted as specified and all other components (both process colours and other spot colours) shall be painted with an additive value of 1.0. The consequences of this are discussed in 11.7.4, "Overprinting and Transparency."

Under the opaque imaging model, a **Separation** or DeviceN colour space may specify the individual process colour components of the output device, as if they were spot colours. However, within a transparency group, this should be done only if the group inherits the native colour space of the output device (or is implicitly converted to **DeviceCMYK**, as discussed in 8.6.5.7, "Implicit Conversion of CIE-Based Colour Spaces"). If any other colour space has been specified for the group, the Separation or **DeviceN** colour space shall be converted to its alternate colour space.

NOTE In general, within a transparency group containing an explicitly-specified colour space, the group's process colour components are different from the device's process colour components. Conversion to the device's process colour components occurs only after all colour compositing computations for the group have been completed. Consequently, the device's process colour components are not accessible within the group.

For instance, outside of any transparency group, a device whose native colour space is **DeviceCMYK** has a **Cyan** component that may be specified in a **Separation** or **DeviceN** colour space. On the other hand, within a transparency group whose colour space is **ICCBased**, the group has no **Cyan** component available to be painted.

11.7.4 Overprinting and Transparency

11.7.4.1 General

In the opaque imaging model, overprinting is controlled by two parameters of the graphics state: the *overprint parameter* and the *overprint mode* (see “Overprint Control”). Painting an object causes some specific set of device colorants to be marked, as determined by the current colour space and current colour in the graphics state. The remaining colorants shall be either erased or left unchanged, depending on whether the overprint parameter is **false** or **true**. When the current colour space is **DeviceCMYK**, the overprint mode parameter additionally enables this selective marking of colorants to be applied to individual colour components according to whether the component value is zero or nonzero.

NOTE 1 Because this model of overprinting deals directly with the painting of device colorants, independently of the colour space in which source colours have been specified, it is highly device-dependent and primarily addresses production needs rather than design intent. Overprinting is usually reserved for opaque colorants or for very dark colours, such as black. It is also invoked during late-stage production operations such as trapping (see “Trapping Support”), when the actual set of device colorants has already been determined.

NOTE 2 Consequently, it is best to think of transparency as taking place in appearance space, but overprinting of device colorants in device space. This means that colorant overprint decisions should be made at output time, based on the actual resultant colorants of any transparency compositing operation. On the other hand, effects similar to overprinting can be achieved in a device-independent manner by taking advantage of blend modes, as described in the next sub-clause.

11.7.4.2 Blend Modes and Overprinting

As stated in 11.7.3, "Spot Colours and Transparency," each graphics object that is painted shall affect all existing colour components: all process colorants in the transparency group’s colour space as well as any available spot colorants. For colour components whose value has not been specified, a source colour value of 1.0 shall be assumed; when objects are fully opaque and the **Normal** blend mode is used, this shall have the effect of erasing those components. This treatment is consistent with the behaviour of the opaque imaging model with the overprint parameter set to **false**.

The transparent imaging model defines some blend modes, such as **Darken**, that can be used to achieve effects similar to overprinting. The blend function for **Darken** is

$$B(c_b, c_s) = \min(c_b, c_s)$$

In this blend mode, the result of compositing shall always be the same as the backdrop colour when the source colour is 1.0, as it is for all unspecified colour components. When the backdrop is fully opaque, this shall leave the result colour unchanged from that of the backdrop. This is consistent with the behaviour of the opaque imaging model with the overprint parameter set to **true**.

If the object or backdrop is not fully opaque, the actions described previously are altered accordingly. That is, the erasing effect shall be reduced, and overprinting an object with a colour value of 1.0 may affect the result colour. While these results may or may not be useful, they lie outside the realm of the overprinting and erasing behaviour defined in the opaque imaging model.

When process colours are overprinted or erased (because a spot colour is being painted), the blending computations described previously shall be done independently for each component in the group’s colour space. If that space is different from the native colour space of the output device, its components are not the device’s actual process colorants; the blending computations shall affect the process colorants only after the group’s results have been converted to the device colour space. Thus the effect is different from that of overprinting or erasing the device’s process colorants directly. On the other hand, this is a fully general operation that works uniformly, regardless of the type of object or of the computations that produced the source colour.

NOTE 1 The discussion so far has focused on those colour components whose values are not specified and that are to be either erased or left unchanged. However, the **Normal** or **Darken** blend modes used for these purposes may not be suitable for use on those components whose colour values are specified. In particular, using the **Darken** blend mode for such components would preclude overprinting a dark colour with a lighter one. Moreover, some other blend mode may be specifically desired for those components.

The PDF graphics state specifies only one current blend mode parameter, which shall always apply to process colorants and sometimes to spot colorants as well. Specifically, only separable, white-preserving blend modes shall be used for spot colours. If the specified blend mode is not separable and white-preserving, it shall apply only to process colour components, and the **Normal** blend mode shall be substituted for spot colours.

A blend mode is *white-preserving* if its blend function B has the property that $B(1.0, 1.0) = 1.0$.

NOTE 2 Of the standard separable blend modes listed in Table 136 in 11.3.5, "Blend Mode," all except **Difference** and **Exclusion** are white-preserving. This ensures that when objects accumulate in an isolated transparency group, the accumulated values for unspecified components remain 1.0 as long as only white-preserving blend modes are used. The group’s results can then be overprinted using **Darken** (or other useful modes) while avoiding unwanted interactions with components whose values were never specified within the group.

11.7.4.3 Compatibility with Opaque Overprinting

Because the use of blend modes to achieve effects similar to overprinting does not make direct use of the overprint control parameters in the graphics state, such methods are usable only by transparency-aware applications. For compatibility with the methods of overprint control used in the opaque imaging model, a special blend mode, CompatibleOverprint, is provided that consults the overprint-related graphics state parameters to compute its result. This mode shall apply only when painting elementary graphics objects (fills, strokes, text, images, and shadings). It shall not be invoked explicitly and shall not be identified by any PDF name object; rather, it shall be implicitly invoked whenever an elementary graphics object is painted while overprinting is enabled (that is, when the overprint parameter in the graphics state is **true**).

NOTE 1 Earlier designs of the transparent imaging model included an additional blend mode named ***Compatible***, which explicitly invoked the CompatibleOverprint blend mode described here. Because CompatibleOverprint is now invoked implicitly whenever appropriate, it is never necessary to specify the ***Compatible*** blend mode for use in compositing.

The Compatible blend mode shall be treated as equivalent to ***Normal***.

The value of the blend function $B(c_b, c_s)$ in the CompatibleOverprint mode shall be either c_b or c_s, depending on the setting of the overprint mode parameter, the current and group colour spaces, and the source colour value c_s:

- If the overprint mode is 1 (nonzero overprint mode) and the current colour space and group colour space are both **DeviceCMYK**, then process colour components with nonzero values shall replace the corresponding component values of the backdrop; components with zero values leave the existing backdrop value unchanged. That is, the value of the blend function $B(c_b, c_s)$ shall be the source component c_s for any process (**DeviceCMYK**) colour component whose (subtractive) colour value is nonzero; otherwise it shall be the backdrop component c_b. For spot colour components, the value shall always be c_b.

- In all other cases, the value of $B(c_b, c_s)$ shall be c_s for all colour components specified in the current colour space, otherwise c_b.

EXAMPLE 1 If the current colour space is **DeviceCMYK** or **CalRGB**, the value of the blend function is c_s for process colour components and c_b for spot components. On the other hand, if the current colour space is a **Separation** space representing a spot colour component, the value is c_s for that spot component and c_b for all process components and all other spot components.

NOTE 2 In the previous descriptions, the term *current colour space* refers to the colour space used for a painting operation. This may be specified by the current colour space parameter in the graphics state (see “Colour Values”), implicitly by colour operators such as ***rg*** (“Colour Operators”), or by the **ColorSpace** entry of an image XObject (“Image Dictionaries”). In the case of an ***Indexed*** space, it refers to the base colour space (see “Indexed Colour Spaces”); likewise for ***Separation*** and ***DeviceN*** spaces that revert to their alternate colour space, as described under “Separation Colour Spaces” and “DeviceN Colour Spaces”.

If the current blend mode when CompatibleOverprint is invoked is any mode other than **Normal**, the object being painted shall be implicitly treated as if it were defined in a non-isolated, non-knockout transparency group and painted using the CompatibleOverprint blend mode. The group’s results shall then be painted using the current blend mode in the graphics state.

NOTE 3 It is not necessary to create such an implicit transparency group if the current blend mode is ***Normal***; simply substituting the CompatibleOverprint blend mode while painting the object produces equivalent results. There are some additional cases in which the implicit transparency group can be optimized out.

EXAMPLE 2 Figure L.20 in Annex L shows the effects of all four possible combinations of blending and overprinting, using the **Screen** blend mode in the **DeviceCMYK** colour space. The label “overprint enabled” means that the overprint parameter in the graphics state is **true** and the overprint mode is 1. In the upper half of the figure, a light green oval is painted opaquely (opacity = 1.0) over a backdrop shading from pure yellow to pure magenta. In the lower half, the same object is painted with transparency (opacity = 0.5).

11.7.4.4 Special Path-Painting Considerations

The overprinting considerations discussed in 11.7.4.3, "Compatibility with Opaque Overprinting," also affect those path-painting operations that combine filling and stroking a path in a single operation. These include the **B**, **B***, **b**, and **b*** operators (see “Path-Painting Operators”) and the painting of glyphs with text rendering mode 2 or 6 (“Text Rendering Mode”). For transparency compositing purposes, the combined fill and stroke shall be treated as a single graphics object, as if they were enclosed in a transparency group. This implicit group is established and used as follows:

- If overprinting is enabled (the overprint parameter in the graphics state is **true**) and the current stroking and nonstroking alpha constants are equal, a non-isolated, non-knockout transparency group shall be established. Within the group, the fill and stroke shall be performed with an alpha value of 1.0 but with the CompatibleOverprint blend mode. The group results shall then be composited with the backdrop, using the originally specified alpha and blend mode.

- In all other cases, a non-isolated knockout group shall be established. Within the group, the fill and stroke shall be performed with their respective prevailing alpha constants and the prevailing blend mode. The group results shall then be composited with the backdrop, using an alpha value of 1.0 and the **Normal** blend mode.

NOTE 1 In the case of showing text with the combined filling and stroking text rendering modes, this behaviour is independent of the text knockout parameter in the graphics state (see “Text Knockout”).

NOTE 2 The purpose of these rules is to avoid having a non-opaque stroke composite with the result of the fill in the region of overlap, which would produce a double border effect that is usually undesirable. The special case that applies when the overprint parameter is **true** is for backward compatibility with the overprinting behavior of the opaque imaging model. If a desired effect cannot be achieved with a combined filling and stroking operator or text rendering mode, it can be achieved by specifying the fill and stroke with separate path objects and an explicit transparency group.

NOTE 3 Overprinting of the stroke over the fill does not work in the second case described previously (although either the fill or the stroke can still overprint the backdrop). Furthermore, if the overprint graphics state parameter is **true**, the results are discontinuous at the transition between equal and unequal values of the stroking and nonstroking alpha constants. For this reason, it is best not to use overprinting for combined filling and stroking operations if the stroking and nonstroking alpha constants are being varied independently.

11.7.4.5 Summary of Overprinting Behaviour

Tables 148 and 149 summarize the overprinting and erasing behaviour in the opaque and transparent imaging models, respectively. Table 148 shows the overprinting rules used in the opaque model, as described in “Overprint Control”. Table 149 shows the equivalent rules as implemented by the CompatibleOverprint blend mode in the transparent model. The names **OP** and **OPM** in the tables refer to the overprint and overprint mode parameters of the graphics state.

Table 148 – Overprinting behavior in the opaque imaging model

Source colour space	**Affected colour component**	**Effect on colour component**		
		OP false	**OP true, OPM 0**	**OP true, OPM 1**
DeviceCMYK, specified directly, not in a sampled image	*C*, *M*, *Y*, or *K*	Paint source	Paint source	Paint source if ≠ 0.0 Do not paint if = 0.0
	Process colorant other than *CMYK*	Paint source	Paint source	Paint source
	Spot colorant	Paint 0.0	Do not paint	Do not paint
Any process colour space (including other cases of **DeviceCMYK**)	Process colorant	Paint source	Paint source	Paint source
	Spot colorant	Paint 0.0	Do not paint	Do not paint
Separation or **DeviceN**	Process colorant	Paint 0.0	Do not paint	Do not paint
	Spot colorant named in source space	Paint source	Paint source	Paint source
	Spot colorant not named in source space	Paint 0.0	Do not paint	Do not paint

Table 149 – Overprinting behavior in the transparent imaging model

Source color space	Affected colour component of group colour space	Value of blend function $B(c_b, c_s)$ expressed as tint		
		OP false	OP true, OPM 0	OP true, OPM 1
DeviceCMYK, specified directly, not in a sampled image	*C*, *M*, *Y*, or *K*	c_s	c_s	c_s if $c_s \neq 0.0$ c_b if $c_s = 0.0$
	Process colour component other than *CMYK*	c_s	c_s	c_s
	Spot colorant	c_s (= 0.0)	c_b	c_b
Any process colour space (including other cases of **DeviceCMYK**)	Process colour component	c_s	c_s	c_s
	Spot colorant	c_s (= 0.0)	c_b	c_b
Separation or **DeviceN**	Process colour component	c_s (= 0.0)	c_b	c_b
	Spot colorant named in source space	c_s	c_s	c_s
	Spot colorant not named in source space	c_s (= 0.0)	c_b	c_b
A group (not an elementary object)	All colour components	c_s	c_s	c_s

Colour component values are represented in these tables as subtractive tint values because overprinting is typically applied to subtractive colorants such as inks rather than to additive ones such as phosphors on a display screen. The CompatibleOverprint blend mode is therefore described as if it took subtractive arguments and returned subtractive results. In reality, however, CompatibleOverprint (like all blend modes) shall treat colour components as additive values; subtractive components shall be complemented before and after application of the blend function.

NOTE 1 This note describes an important difference between Table 148 and Table 149. In Table 148, the process colour components being discussed are the actual device colorants—the colour components of the output device's native colour space (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**). In Table 149, the process colour components are those of the group's colour space, which is not necessarily the same as that of the output device (and can even be something like **CalRGB** or **ICCBased**). For this reason, the process colour components of the group colour space cannot be treated as if they were spot colours in a **Separation** or **DeviceN** colour space (see "Spot Colours and Transparency"). This difference between opaque and transparent overprinting and erasing rules arises only within a transparency group (including the page group, if its colour space is different from the native colour space of the output device). There is no difference in the treatment of spot colour components.

NOTE 2 Table 149 has one additional row at the bottom. It applies when painting an object that is a transparency group rather than an elementary object (fill, stroke, text, image, or shading). As stated in 11.7.3, "Spot Colours and Transparency," a group is considered to paint all colour components, both process and spot. Colour components that were not explicitly painted by any object in the group have an additive colour value of 1.0 (subtractive tint 0.0). Since no information is retained about which components were actually painted within the group, compatible overprinting is not possible in this case; the CompatibleOverprint blend mode reverts to **Normal**, with no consideration of the overprint and overprint mode parameters. A transparency-aware conforming writer can choose a more suitable blend mode, such as **Darken**, to produce an effect similar to overprinting.

11.7.5 Rendering Parameters and Transparency

11.7.5.1 General

The opaque imaging model has several graphics state parameters dealing with the rendering of colour: the current halftone (see “Halftone Dictionaries”), transfer functions (“Transfer Functions”), rendering intent (“Rendering Intents”), and black-generation and undercolor-removal functions (“Conversion from DeviceRGB to DeviceCMYK”). All of these rendering parameters may be specified on a per-object basis; they control how a particular object is rendered. When all objects are opaque, it is easy to define what this means. But when they are transparent, more than one object may contribute to the colour at a given point; it is unclear which rendering parameters to apply in an area where transparent objects overlap. At the same time, the transparent imaging model should be consistent with the opaque model when only opaque objects are painted.

There are two categories of rendering parameters that are treated somewhat differently in the presence of transparency. In the first category are halftone and transfer functions, which are applied only when the final colour at a given point on the page is known. In the second category are rendering intent, black generation, and undercolor removal, which are applied whenever colours are converted from one colour space to another.

11.7.5.2 Halftone and Transfer Function

When objects are transparent, rendering of an object does not occur when the object is specified but at some later time. Hence, the implementation shall keep track of the halftone and transfer function parameters at each point on the page from the time they are specified until the time rendering actually occurs. This means that these rendering parameters shall be associated with regions of the page rather than with individual objects.

The halftone and transfer function to be used at any given point on the page shall be those in effect at the time of painting the last (topmost) elementary graphics object enclosing that point, but only if the object is fully opaque. Only elementary objects shall be relevant; the rendering parameters associated with a group object are ignored. The *topmost object* at any point shall be defined to be the topmost elementary object in the entire page stack that has a nonzero object shape value (f_j) at that point (that is, for which the point is inside the object). An object shall be considered to be *fully opaque* if all of the following conditions hold at the time the object is painted:

- The current alpha constant in the graphics state (stroking or nonstroking, depending on the painting operation) is 1.0.
- The current blend mode in the graphics state is **Normal** (or **Compatible**, which is treated as equivalent to **Normal**).
- The current soft mask in the graphics state is **None**. If the object is an image XObject, there is not an **SMask** entry in its image dictionary.
- The foregoing three conditions were also true at the time the **Do** operator was invoked for the group containing the object, as well as for any direct ancestor groups.
- If the current colour is a tiling pattern, all objects in the definition of its pattern cell also satisfy the foregoing conditions.

Together, these conditions ensure that only the object itself shall contribute to the colour at the given point, completely obscuring the backdrop. For portions of the page whose topmost object is not fully opaque or that are never painted at all, the default halftone and transfer function for the page shall be used.

If a graphics object is painted with overprinting enabled—that is, if the applicable (stroking or nonstroking) overprint parameter in the graphics state is **true**—the halftone and transfer function to use at a given point shall be determined independently for each colour component. Overprinting implicitly invokes the CompatibleOverprint blend mode (see “Compatibility with Opaque Overprinting”). An object shall be considered opaque for a given component only if CompatibleOverprint yields the source colour (not the backdrop colour) for that component.

11.7.5.3 Rendering Intent and Colour Conversions

The rendering intent, black-generation, and undercolor-removal parameters control certain colour conversions. In the presence of transparency, they may need to be applied earlier than the actual rendering of colour onto the page.

The rendering intent influences the conversion from a CIE-based colour space to a target colour space, taking into account the target space's colour gamut (the range of colours it can reproduce). Whereas in the opaque imaging model the target space shall always be the native colour space of the output device, in the transparent model it may instead be the group colour space of a transparency group into which an object is being painted.

The rendering intent is needed at the moment such a conversion is performed—that is, when painting an elementary or group object specified in a CIE-based colour space into a parent group having a different colour space.

NOTE 1 This differs from the current halftone and transfer function, whose values are used only when all colour compositing has been completed and rasterization is being performed.

In all cases, the rendering intent to use for converting an object's colour (whether that of an elementary object or of a transparency group) shall be determined by the rendering intent parameter associated with the object. In particular:

- When painting an elementary object with a CIE-based colour into a transparency group having a different colour space, the rendering intent used shall be the current rendering intent in effect in the graphics state at the time of the painting operation.
- When painting a transparency group whose colour space is CIE-based into a parent group having a different colour space, the rendering intent used shall be the current rendering intent in effect at the time the **Do** operator is applied to the group.
- When the colour space of the page group is CIE-based, the rendering intent used to convert colours to the native colour space of the output device shall be the default rendering intent for the page.

NOTE 2 Since there may be one or more nested transparency groups having different CIE-based colour spaces, the colour of an elementary source object may be converted to the device colour space in multiple stages, controlled by the rendering intent in effect at each stage. The proper choice of rendering intent at each stage depends on the relative gamuts of the source and target colour spaces. It is specified explicitly by the document producer, not prescribed by the PDF specification, since no single policy for managing rendering intents is appropriate for all situations.

A similar approach works for the black-generation and undercolor-removal functions, which shall be applied only during conversion from **DeviceRGB** to **DeviceCMYK** colour spaces:

- When painting an elementary object with a **DeviceRGB** colour directly into a transparency group whose colour space is **DeviceCMYK**, the functions used shall be the current black-generation and undercolor-removal functions in effect in the graphics state at the time of the painting operation.
- When painting a transparency group whose colour space is **DeviceRGB** into a parent group whose colour space is **DeviceCMYK**, the functions used shall be the ones in effect at the time the **Do** operator is applied to the group.
- When the colour space of the page group is **DeviceRGB** and the native colour space of the output device is **DeviceCMYK**, the functions used to convert colours to the device's colour space shall be the default functions for the page.

12 Interactive Features

12.1 General

For purposes of the trigger events **E** (enter), **X** (exit), **D** (down), and **U** (up), the term *mouse* denotes a generic pointing device with the following characteristics:

- A selection button that can be *pressed*, *held down*, and *released*. If there is more than one mouse button, the selection button is typically the left button.
- A notion of *location*—that is, an indication of where on the screen the device is pointing. Location is typically denoted by a screen cursor.
- A notion of *focus*—that is, which element in the document is currently interacting with the with the user. In many systems, this element is denoted by a blinking caret, a focus rectangle, or a colour change.

This clause describes the PDF features that allow a user to interact with a document on the screen, using the mouse and keyboard (with the exception of multimedia features, which are described in 13, “Multimedia Features”):

- *Preference settings* to control the way the document is presented on the screen (12.2, “Viewer Preferences”)
- *Navigation* facilities for moving through the document in a variety of ways (Sections 12.3, “Document-Level Navigation” and 12.4, “Page-Level Navigation”)
- *Annotations* for adding text notes, sounds, movies, and other ancillary information to the document (12.5, “Annotations”)
- *Actions* that can be triggered by specified events (12.6, “Actions”)
- *Interactive forms* for gathering information from the user (12.7, “Interactive Forms”)
- *Digital signatures* that authenticate the identity of a user and the validity of the document’s contents (12.8, “Digital Signatures”)
- *Measurement properties* that enable the display of real-world units corresponding to objects on a page (12.9, “Measurement Properties”)

12.2 Viewer Preferences

The **ViewerPreferences** entry in a document’s catalogue (see 7.7.2, “Document Catalog”) *designates a viewer preferences dictionary (PDF 1.2)* controlling the way the document shall be presented on the screen or in print. If no such dictionary is specified, conforming readers should behave in accordance with their own current user preference settings. Table 150 shows the contents of the viewer preferences dictionary.

Table 150 – Entries in a viewer preferences dictionary

Key	Type	Value
HideToolbar	boolean	*(Optional)* A flag specifying whether to hide the conforming reader’s tool bars when the document is active. Default value: **false**.
HideMenubar	boolean	*(Optional)* A flag specifying whether to hide the conforming reader’s menu bar when the document is active. Default value: **false**.

Table 150 – Entries in a viewer preferences dictionary (continued)

Key	Type	Value
HideWindowUI	boolean	*(Optional)* A flag specifying whether to hide user interface elements in the document's window (such as scroll bars and navigation controls), leaving only the document's contents displayed. Default value: **false**.
FitWindow	boolean	*(Optional)* A flag specifying whether to resize the document's window to fit the size of the first displayed page. Default value: **false**.
CenterWindow	boolean	*(Optional)* A flag specifying whether to position the document's window in the center of the screen. Default value: **false**.
DisplayDocTitle	boolean	*(Optional; PDF 1.4)* A flag specifying whether the window's title bar should display the document title taken from the **Title** entry of the document information dictionary (see 14.3.3, "Document Information Dictionary"). If **false**, the title bar should instead display the name of the PDF file containing the document. Default value: **false**.
NonFullScreenPageMode	name	*(Optional)* The document's *page mode*, specifying how to display the document on exiting full-screen mode: UseNone Neither document outline nor thumbnail images visible UseOutlines Document outline visible UseThumbs Thumbnail images visible UseOC Optional content group panel visible This entry is meaningful only if the value of the **PageMode** entry in the **Catalog** dictionary (see 7.7.2, "Document Catalog") is FullScreen; it shall be ignored otherwise. Default value: UseNone.
Direction	name	*(Optional; PDF 1.3)* The predominant reading order for text: L2R Left to right R2L Right to left (including vertical writing systems, such as Chinese, Japanese, and Korean) This entry has no direct effect on the document's contents or page numbering but may be used to determine the relative positioning of pages when displayed side by side or printed *n*-up. Default value: L2R.
ViewArea	name	*(Optional; PDF 1.4)* The name of the page boundary representing the area of a page that shall be displayed when viewing the document on the screen. The value is the key designating the relevant page boundary in the page object (see 7.7.3, "Page Tree" and 14.11.2, "Page Boundaries"). If the specified page boundary is not defined in the page object, its default value shall be used, as specified in Table 30. Default value: CropBox. This entry is intended primarily for use by prepress applications that interpret or manipulate the page boundaries as described in 14.11.2, "Page Boundaries." NOTE 1 Most conforming readers disregard it.

Table 150 – Entries in a viewer preferences dictionary (continued)

Key	Type	Value
ViewClip	name	*(Optional; PDF 1.4)* The name of the page boundary to which the contents of a page shall be clipped when viewing the document on the screen. The value is the key designating the relevant page boundary in the page object (see 7.7.3, “Page Tree” and 14.11.2, “Page Boundaries”). If the specified page boundary is not defined in the page object, its default value shall be used, as specified in Table 30. Default value: CropBox. This entry is intended primarily for use by prepress applications that interpret or manipulate the page boundaries as described in 14.11.2, “Page Boundaries.” NOTE 2 Most conforming readers disregard it.
PrintArea	name	*(Optional; PDF 1.4)* The name of the page boundary representing the area of a page that shall be rendered when printing the document. The value is the key designating the relevant page boundary in the page object (see 7.7.3, “Page Tree” and 14.11.2, “Page Boundaries”). If the specified page boundary is not defined in the page object, its default value shall be used, as specified in Table 30. Default value: CropBox. This entry is intended primarily for use by prepress applications that interpret or manipulate the page boundaries as described in 14.11.2, “Page Boundaries.” NOTE 3 Most conforming readers disregard it.
PrintClip	name	*(Optional; PDF 1.4)* The name of the page boundary to which the contents of a page shall be clipped when printing the document. The value is the key designating the relevant page boundary in the page object (see 7.7.3, “Page Tree” and 14.11.2, “Page Boundaries”). If the specified page boundary is not defined in the page object, its default value shall be used, as specified in Table 30. Default value: CropBox. This entry is intended primarily for use by prepress applications that interpret or manipulate the page boundaries as described in 14.11.2, “Page Boundaries.” NOTE 4 Most conforming readers disregard it.
PrintScaling	name	*(Optional; PDF 1.6)* The page scaling option that shall be selected when a print dialog is displayed for this document. Valid values are None, which indicates no page scaling, and AppDefault, which indicates the conforming reader’s default print scaling. If this entry has an unrecognized value, AppDefault shall be used. Default value: AppDefault. If the print dialog is suppressed and its parameters are provided from some other source, this entry nevertheless shall be honored.
Duplex	name	(*Optional; PDF 1.7*) The paper handling option that shall be used when printing the file from the print dialog. The following values are valid: **Simplex** Print single-sided **DuplexFlipShortEdge** Duplex and flip on the short edge of the sheet **DuplexFlipLongEdge** Duplex and flip on the long edge of the sheet Default value: none

Table 150 – Entries in a viewer preferences dictionary (continued)

Key	Type	Value
PickTrayByPDFSize	boolean	(*Optional; PDF 1.7*) A flag specifying whether the PDF page size shall be used to select the input paper tray. This setting influences only the preset values used to populate the print dialog presented by a conforming reader. If **PickTrayByPDFSize** is **true**, the check box in the print dialog associated with input paper tray shall be checked. This setting has no effect on operating systems that do not provide the ability to pick the input tray by size. Default value: as defined by the conforming reader
PrintPageRange	array	(*Optional; PDF 1.7*) The page numbers used to initialize the print dialog box when the file is printed. The array shall contain an even number of integers to be interpreted in pairs, with each pair specifying the first and last pages in a sub-range of pages to be printed.The first page of the PDF file shall be denoted by 1. Default value: as defined by the conforming reader
NumCopies	integer	(*Optional; PDF 1.7*) The number of copies that shall be printed when the print dialog is opened for this file. Values outside this range shall be ignored. Default value: as defined by the conforming reader, but typically 1

12.3 Document-Level Navigation

12.3.1 General

The features described in this sub-clause allow a conforming reader to present the user with an interactive, global overview of a document in either of two forms:

- As a hierarchical *outline* showing the document's internal structure
- As a collection of *thumbnail images* representing the pages of the document in miniature form

Each item in the outline or each thumbnail image may be associated with a corresponding *destination* in the document, so that the user can jump directly to the destination by clicking with the mouse.

12.3.2 Destinations

12.3.2.1 General

A *destination* defines a particular view of a document, consisting of the following items:

- The page of the document that shall be displayed
- The location of the document window on that page
- The magnification (zoom) factor

Destinations may be associated with outline items (see 12.3.3, “Document Outline”), annotations (12.5.6.5, “Link Annotations”), or actions (12.6.4.2, “Go-To Actions” and 12.6.4.3, “Remote Go-To Actions”). In each case, the destination specifies the view of the document that shall be presented when the outline item or annotation is opened or the action is performed. In addition, the optional **OpenAction** entry in a document's catalogue (7.7.2, “Document Catalog”) may specify a destination that shall be displayed when the document is opened. A destination may be specified either explicitly by an array of parameters defining its properties or indirectly by name.

12.3.2.2 Explicit Destinations

Table 151 shows the allowed syntactic forms for specifying a destination explicitly in a PDF file. In each case, *page* is an indirect reference to a page object (except in a remote go-to action; see 12.6.4.3, “Remote Go-To Actions”). All coordinate values (*left*, *right*, *top*, and *bottom*) shall be expressed in the default user space coordinate system. The page’s *bounding box* is the smallest rectangle enclosing all of its contents. (If any side of the bounding box lies outside the page’s crop box, the corresponding side of the crop box shall be used instead; see 14.11.2, “Page Boundaries,” for further discussion of the crop box.)

No page object can be specified for a destination associated with a remote go-to action (see 12.6.4.3, “Remote Go-To Actions”) because the destination page is in a different PDF document. In this case, the *page* parameter specifies an integer page number within the remote document instead of a page object in the current document.

Table 151 – Destination syntax

Syntax	Meaning
[*page* /XYZ *left top zoom*]	Display the page designated by *page*, with the coordinates (*left*, *top*) positioned at the upper-left corner of the window and the contents of the page magnified by the factor *zoom*. A null value for any of the parameters *left*, *top*, or *zoom* specifies that the current value of that parameter shall be retained unchanged. A *zoom* value of 0 has the same meaning as a null value.
[*page* /Fit]	Display the page designated by *page*, with its contents magnified just enough to fit the entire page within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the page within the window in the other dimension.
[*page* /FitH *top*]	Display the page designated by *page*, with the vertical coordinate *top* positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of the page within the window. A null value for *top* specifies that the current value of that parameter shall be retained unchanged.
[*page* /FitV *left*]	Display the page designated by *page*, with the horizontal coordinate *left* positioned at the left edge of the window and the contents of the page magnified just enough to fit the entire height of the page within the window. A null value for *left* specifies that the current value of that parameter shall be retained unchanged.
[*page* /FitR *left bottom right top*]	Display the page designated by *page*, with its contents magnified just enough to fit the rectangle specified by the coordinates *left*, *bottom*, *right*, and *top* entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the rectangle within the window in the other dimension.
[*page* /FitB]	*(PDF 1.1)* Display the page designated by *page*, with its contents magnified just enough to fit its bounding box entirely within the window both horizontally and vertically. If the required horizontal and vertical magnification factors are different, use the smaller of the two, centering the bounding box within the window in the other dimension.
[*page* /FitBH *top*]	*(PDF 1.1)* Display the page designated by *page*, with the vertical coordinate *top* positioned at the top edge of the window and the contents of the page magnified just enough to fit the entire width of its bounding box within the window. A null value for *top* specifies that the current value of that parameter shall be retained unchanged.

Table 151 – Destination syntax (continued)

Syntax	Meaning
[*page* /FitBV *left*]	*(PDF 1.1)* Display the page designated by *page*, with the horizontal coordinate *left* positioned at the left edge of the window and the contents of the page magnified just enough to fit the entire height of its bounding box within the window. A null value for *left* specifies that the current value of that parameter shall be retained unchanged.

12.3.2.3 Named Destinations

Instead of being defined directly with the explicit syntax shown in Table 151, a destination may be referred to indirectly by means of a name object *(PDF 1.1)* or a byte string *(PDF 1.2)*. This capability is especially useful when the destination is located in another PDF document.

NOTE 1 A link to the beginning of Chapter 6 in another document might refer to the destination by a name, such as Chap6.begin, instead of by an explicit page number in the other document. Then, the location of the chapter in the other document could change without invalidating the link. If an annotation or outline item that refers to a named destination has an associated action, such as a remote go-to action (see 12.6.4.3, “Remote Go-To Actions”) or a thread action (12.6.4.6, “Thread Actions”), the destination is in the file specified by the action’s **F** entry, if any; if there is no **F** entry, the destination is in the current file.

In PDF 1.1, the correspondence between name objects and destinations shall be defined by the **Dests** entry in the document catalogue (see 7.7.2, “Document Catalog”). The value of this entry shall be a dictionary in which each key is a destination name and the corresponding value is either an array defining the destination, using the syntax shown in Table 151, or a dictionary with a **D** entry whose value is such an array.

NOTE 2 The latter form allows additional attributes to be associated with the destination, as well as enabling a go-to action (see 12.6.4.2, “Go-To Actions”) that shall be used as the target of a named destination.

In PDF 1.2 and later, the correspondence between strings and destinations may alternatively be defined by the **Dests** entry in the document’s name dictionary (see 7.7.4, “Name Dictionary”). The value of this entry shall be a name tree (7.9.6, “Name Trees”) mapping name strings to destinations. (The keys in the name tree may be treated as text strings for display purposes.) The destination value associated with a key in the name tree may be either an array or a dictionary, as described in the preceding paragraph.

NOTE 3 The use of strings as destination names is a PDF 1.2 feature. If compatibility with earlier versions of PDF is required, only name objects may be used to refer to named destinations. A document that supports PDF 1.2 can contain both types. However, if backward compatibility is not a consideration, applications should use the string form of representation in the **Dests** name tree.

12.3.3 Document Outline

A PDF document may contain a *document outline* that the conforming reader may display on the screen, allowing the user to navigate interactively from one part of the document to another. The outline consists of a tree-structured hierarchy of *outline items* (sometimes called *bookmarks*), which serve as a visual table of contents to display the document’s structure to the user. The user may interactively open and close individual items by clicking them with the mouse. When an item is open, its immediate children in the hierarchy shall become visible on the screen; each child may in turn be open or closed, selectively revealing or hiding further parts of the hierarchy. When an item is closed, all of its descendants in the hierarchy shall be hidden. Clicking the text of any visible item *activates* the item, causing the conforming reader to jump to a destination or trigger an action associated with the item.

The root of a document’s outline hierarchy is an *outline dictionary* specified by the **Outlines** entry in the document catalogue (see 7.7.2, “Document Catalog”). Table 152 shows the contents of this dictionary. Each individual outline item within the hierarchy shall be defined by an *outline item dictionary* (Table 153). The items at each level of the hierarchy form a linked list, chained together through their **Prev** and **Next** entries and accessed through the **First** and **Last** entries in the parent item (or in the outline dictionary in the case of top-level items). When displayed on the screen, the items at a given level shall appear in the order in which they occur in the linked list.

Table 152 – Entries in the outline dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Outlines** for an outline dictionary.
First	dictionary	*(Required if there are any open or closed outline entries;* shall *be an indirect reference)* An outline item dictionary representing the first top-level item in the outline.
Last	dictionary	*(Required if there are any open or closed outline entries;* shall *be an indirect reference)* An outline item dictionary representing the last top-level item in the outline.
Count	integer	(*Required if the document has any open outline entries*) Total number of visible outline items at all levels of the outline. The value cannot be negative. This entry shall be omitted if there are no open outline items.

Table 153 – Entries in an outline item dictionary

Key	Type	Value
Title	text string	*(Required)* The text that shall be displayed on the screen for this item.
Parent	dictionary	*(Required; shall be an indirect reference)* The parent of this item in the outline hierarchy. The parent of a top-level item shall be the outline dictionary itself.
Prev	dictionary	*(Required for all but the first item at each level; shall be an indirect reference)* The previous item at this outline level.
Next	dictionary	*(Required for all but the last item at each level; shall be an indirect reference)* The next item at this outline level.
First	dictionary	*(Required if the item has any descendants; shall be an indirect reference)* The first of this item’s immediate children in the outline hierarchy.
Last	dictionary	*(Required if the item has any descendants; shall be an indirect reference)* The last of this item’s immediate children in the outline hierarchy.
Count	integer	*(Required if the item has any descendants)* If the outline item is open, **Count** is the sum of the number of visible descendent outline items at all levels. The number of visible descendent outline items shall be determined by the following recursive process: Step 1. Initialize **Count** to zero. Step 2. Add to **Count** the number of immediate children. During repetitions of this step, update only the **Count** of the original outline item. Step 3. For each of those immediate children whose **Count** is positive and non-zero, repeat steps 2 and 3. If the outline item is closed, **Count** is negative and its absolute value is the number of descendants that would be visible if the outline item were opened.

Table 153 – Entries in an outline item dictionary (continued)

Key	Type	Value
Dest	name, byte string, or array	*(Optional; shall not be present if an **A** entry is present)* The destination that shall be displayed when this item is activated (see 12.3.2, "Destinations").
A	dictionary	*(Optional; PDF 1.1; shall not be present if a **Dest** entry is present)* The action that shall be performed when this item is activated (see 12.6, "Actions").
SE	dictionary	*(Optional; PDF 1.3; shall be an indirect reference)* The structure element to which the item refers (see 14.7.2, "Structure Hierarchy"). *(PDF 1.0)* An item may also specify a destination (***Dest***) corresponding to an area of a page where the contents of the designated structure element are displayed.
C	array	*(Optional; PDF 1.4)* An array of three numbers in the range 0.0 to 1.0, representing the components in the **DeviceRGB** colour space of the colour that shall be used for the outline entry's text. Default value: [0.0 0.0 0.0].
F	integer	*(Optional; PDF 1.4)* A set of flags specifying style characteristics for displaying the outline item's text (see Table 154). Default value: 0.

The value of the outline item dictionary's **F** entry *(PDF 1.4)* shall be an integer interpreted as one-bit flags specifying style characteristics for displaying the item. Bit positions within the flag word are numbered from low-order to high-order bits, with the lowest-order bit numbered 1. Table 154 shows the meanings of the flags; all other bits of the integer shall be 0.

Table 154 – Outline item flags

Bit position	Name	Meaning
1	Italic	If set to 1, display the item in italic.
2	Bold	If set to 1, display the item in bold.

EXAMPLE The following example shows a typical outline dictionary and outline item dictionary. See H.6, "Outline Hierarchy Example" for an example of a complete outline hierarchy.

```
21  0  obj
    <<  /Count  6
        /First  22 0 R
        /Last  29 0 R
    >>
endobj

22  0  obj
    <<  /Title  (Chapter 1)
        /Parent  21 0 R
        /Next  26 0 R
        /First  23 0 R
        /Last  25 0 R
        /Count  3
        /Dest  [3 0 R  /XYZ  0 792 0]
    >>
endobj
```

12.3.4 Thumbnail Images

A PDF document may contain *thumbnail images* representing the contents of its pages in miniature form. A conforming reader may display these images on the screen, allowing the user to navigate to a page by clicking its thumbnail image:

NOTE Thumbnail images are not required, and may be included for some pages and not for others.

The thumbnail image for a page shall be an image XObject specified by the **Thumb** entry in the page object (see 7.7.3, “Page Tree”). It has the usual structure for an image dictionary (8.9.5, “Image Dictionaries”), but only the **Width**, **Height**, **ColorSpace**, **BitsPerComponent**, and **Decode** entries are significant; all of the other entries listed in Table 89 shall be ignored if present. (If a **Subtype** entry is specified, its value shall be **Image**.) The image’s colour space shall be either **DeviceGray** or **DeviceRGB**, or an **Indexed** space based on one of these.

EXAMPLE This example shows a typical thumbnail image definition.

```
12  0  obj
    <<  /Width  76
        /Height  99
        /ColorSpace  /DeviceRGB
        /BitsPerComponent  8
        /Length  13 0 R
        /Filter  [/ASCII85Decode  /DCTDecode]
    >>

stream
s4IA>!"M;*Ddm8XA,lT0!!3,S!/(=R!<E3%!<N<(!WrK*!WrN,
…Omitted data…
endstream
endobj

13  0  obj                                      % Length of stream
…
endobj
```

12.3.5 Collections

Beginning with PDF 1.7, PDF documents may specify how a conforming reader’s user interface presents collections of file attachments, where the attachments are related in structure or content. Such a presentation is called a portable collection.

NOTE 1 The intent of portable collections is to present, sort, and search collections of related documents embedded in the containing PDF document, such as email archives, photo collections, and engineering bid sets. There is no requirement that documents in a collection have an implicit relationship or even a similarity; however, showing differentiating characteristics of related documents can be helpful for document navigation.

A *collection dictionary* specifies the viewing and organizational characteristics of portable collections. If this dictionary is present in a PDF document, the conforming reader shall present the document as a portable collection. The **EmbeddedFiles** name tree specifies file attachments (see 7.11.4, “Embedded File Streams”).

When a conforming reader first opens a PDF document containing a collection, it shall display the contents of the initial document, along with a list of the documents present in the **EmbeddedFiles** name tree. The document list shall include the additional document information specified by the collection schema. The initial document may be the container PDF or one of the embedded documents.

NOTE 2 The page content in the initial document should contain information that helps the user understand what is contained in the collection, such as a title and an introductory paragraph.

The file attachments comprising a collection shall be located in the **EmbeddedFiles** name tree. All attachments in that tree are in the collection; any attachments not in that tree are not.

Table 155 describes the entries in a collection dictionary.

Table 155 – Entries in a collection dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be Collection for a collection dictionary.
Schema	dictionary	(*Optional*) A collection schema dictionary (see Table 156). If absent, the conforming reader may choose useful defaults that are known to exist in a file specification dictionary, such as the file name, file size, and modified date.
D	byte string	(*Optional*) A string that identifies an entry in the **EmbeddedFiles** name tree, determining the document that shall be initially presented in the user interface. If the **D** entry is missing or in error, the initial document shall be the one that contains the collection dictionary.
View	name	(*Optional*) The initial view. The following values are valid: **D** The collection view shall be presented in details mode, with all information in the Schema dictionary presented in a multi-column format. This mode provides the most information to the user. **T** The collection view shall be presented in tile mode, with each file in the collection denoted by a small icon and a subset of information from the **Schema** dictionary. This mode provides top-level information about the file attachments to the user. **H** The collection view shall be initially hidden, The conforming reader shall provide means for the user to view the collection by some explicit action. Default value: **D**
Sort	dictionary	(*Optional*) A collection sort dictionary, which specifies the order in which items in the collection shall be sorted in the user interface (see Table 158).

A *collection schema dictionary* consists of a variable number of individual collection field dictionaries. Each collection field dictionary has a key chosen by the conforming writer, which shall be used to associate a field with data in a file specification. Table 156 describes the entries in a collection schema dictionary.

Table 156 – Entries in a collection schema dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be CollectionSchema for a collection schema dictionary.
Other keys	dictionary	(*Optional*) A collection field dictionary. Each key name is chosen at the discretion of the conforming writer. The key name shall be used to identify a corresponding collection item dictionary referenced from the file specification dictionary's CI entry (see CI key in Table 44).

A *collection field dictionary* describes the attributes of a particular field in a portable collection, including the type of data stored in the field and the lookup key used to locate the field data in the file specification dictionary. Table 157 describes the entries in a collection field dictionary.

Table 157 – Entries in a collection field dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be CollectionField for a collection field dictionary.
Subtype	name	(*Required*) The subtype of collection field or file-related field that this dictionary describes. This entry identifies the type of data that shall be stored in the field. The following values identify the types of fields in the collection item or collection subitem dictionary: **S** A text field. The field data shall be stored as a PDF text string. **D** A date field. The field data shall be stored as a PDF date string. **N** A number field. The field data shall be stored as a PDF number. The following values identify the types of file-related fields: **F** The field data shall be the file name of the embedded file stream, as identified by the **UF** entry of the file specification, if present; otherwise by the **F** entry of the file specification (see Table 44). **Desc** The field data shall be the description of the embedded file stream, as identified by the **Desc** entry in the file specification dictionary (see Table 44). **ModDate** The field data shall be the modification date of the embedded file stream, as identified by the **ModDate** entry in the embedded file parameter dictionary (see Table 46). **CreationDate** The field data shall be the creation date of the embedded file stream, as identified by the **CreationDate** entry in the embedded file parameter dictionary (see Table 46). **Size** The field data shall be the size of the embedded file, as identified by the **Size** entry in the embedded file parameter dictionary (see Table 46).
N	text string	(*Required*) The textual field name that shall be presented to the user by the conforming reader.
O	integer	(*Optional*) The relative order of the field name in the user interface. Fields shall be sorted by the conforming reader in ascending order.
V	boolean	(*Optional*) The initial visibility of the field in the user interface. Default value: **true**.
E	boolean	(*Optional*) A flag indicating whether the conforming reader should provide support for editing the field value. Default value: **false**.

A *collection sort dictionary* identifies the fields that shall be used to sort items in the collection. The type of sorting depends on the type of data:

- Text strings shall be ordered lexically from smaller to larger, if ascending order is specified.

NOTE 3 Lexical ordering is an implementation dependency for conforming readers.

- Numbers shall be ordered numerically from smaller to larger, if ascending order is specified.
- Dates shall be ordered from oldest to newest, if ascending order is specified.

Table 158 describes the entries in a collection sort dictionary.

Table 158 – Entries in a collection sort dictionary

Key	Type	Value
Type	name	(*Optional*) The type of PDF object that this dictionary describes; if present, shall be CollectionSort for a collection sort dictionary.
S	name or array	(*Required*) The name or names of fields that the conforming reader shall use to sort the items in the collection. If the value is a name, it identifies a field described in the parent collection dictionary. If the value is an array, each element of the array shall be a name that identifies a field described in the parent collection dictionary. The array form shall be used to allow additional fields to contribute to the sort, where each additional field shall be used to break ties. More specifically, if multiple collection item dictionaries have the same value for the first field named in the array, the values for successive fields named in the array shall be used for sorting, until a unique order is determined or until the named fields are exhausted.
A	boolean or array	(*Optional*) If the value is a boolean, it specifies whether the conforming reader shall sort the items in the collection in ascending order (**true**) or descending order (**false**). If the value is an array, each element of the array shall be a boolean value that specifies whether the entry at the same index in the **S** array shall be sorted in ascending or descending order. If the number of entries in the **A** array is larger than the number of entries in the **S** array the extra entries in the **A** array shall be ignored. If the number of entries in the **A** array is less than the number of entries in the **S** array the missing entries in the **A** array shall be assumed to be **true**. Default value: **true**.

EXAMPLE 1 This example shows a collection dictionary representing an email in-box, where each item in the collection is an email message. The actual email messages are contained in file specification dictionaries. The organizational data associated with each email is described in a collection schema dictionary. Most actual organizational data (from, to, date, and subject) is provided in a collection item dictionary, but the size data comes from the embedded file parameter dictionary.

```
/Collection <<
  /Type /Collection
  /Schema <<
      /Type /CollectionSchema
      /from << /Subtype /S /N (From) /O 1 /V true /E false>>
      /to << /Subtype /S /N (To) /O 2 /V true /E false >>
      /date << /Subtype /D /N (Date received) /O 3 /V true /E false >>
      /subject << /Subtype /S /N (Subject) /O 4 /V true /E false >>
      /size << /Subtype /Size /N (Size) /O 5 /V true /E false >>
      >>
  /D (Doc1)
  /View /D
  /Sort << /S /date /A false >>
>>
```

EXAMPLE 2 This example shows a collection item dictionary and a collection subitem dictionary. These dictionaries contain entries that correspond to the schema entries specified in the Example in 12.4.2, “Page Labels.”. 7.11.6, “Collection Items” specifies the collection item and collection subitem dictionaries.

```
/CI <<
  /Type /CollectionItem
  /from (Rob McAfee)
  /to (Patty McAfee)
  /subject <<
     /Type /CollectionSubitem
     /P (Re:)
     /D (Let's have lunch on Friday!)
  >>
  /date (D:20050621094703-07’00’)
>>
```

12.4 Page-Level Navigation

12.4.1 General

This sub-clause describes PDF facilities that enable the user to navigate from page to page within a document:

- *Page labels* for numbering or otherwise identifying individual pages (see 12.4.2, “Page Labels”).
- *Article threads*, which chain together items of content within the document that are logically connected but not physically sequential (see 12.4.3, “Articles”).
- *Presentations* that display the document in the form of a slide show, advancing from one page to the next either automatically or under user control (see 12.4.4, “Presentations”).

For another important form of page-level navigation, see 12.5.6.5, “Link Annotations.”

12.4.2 Page Labels

Each page in a PDF document shall be identified by an integer *page index* that expresses the page’s relative position within the document. In addition, a document may optionally define *page labels (PDF 1.3)* to identify each page visually on the screen or in print. Page labels and page indices need not coincide: the indices shall be fixed, running consecutively through the document starting from 0 for the first page, but the labels may be specified in any way that is appropriate for the particular document.

NOTE 1 If the document begins with 12 pages of front matter numbered in roman numerals and the remainder of the document is numbered in arabic, the first page would have a page index of 0 and a page label of i, the twelfth page would have index 11 and label xii, and the thirteenth page would have index 12 and label 1.

For purposes of page labelling, a document shall be divided into *labelling ranges*, each of which is a series of consecutive pages using the same numbering system. Pages within a range shall be numbered sequentially in ascending order. A page’s label consists of a numeric portion based on its position within its labelling range, optionally preceded by a *label prefix denoting the range itself.*

NOTE 2 The pages in an appendix might be labeled with decimal numeric portions prefixed with the string A-; the resulting page labels would be A-1, A-2, and so on.

A document’s labelling ranges shall be defined by the **PageLabels** entry in the document catalogue (see 7.7.2, “Document Catalog”). The value of this entry shall be a number tree (7.9.7, “Number Trees”), each of whose keys is the page index of the first page in a labelling range. The corresponding value shall be a *page label dictionary* defining the labelling characteristics for the pages in that range. The tree shall include a value for page index 0. Table 159 shows the contents of a page label dictionary.

Table 159 – Entries in a page label dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **PageLabel** for a page label dictionary.
S	name	*(Optional)* The numbering style that shall be used for the numeric portion of each page label: D Decimal arabic numerals R Uppercase roman numerals r Lowercase roman numerals A Uppercase letters (A to Z for the first 26 pages, AA to ZZ for the next 26, and so on) a Lowercase letters (a to z for the first 26 pages, aa to zz for the next 26, and so on) There is no default numbering style; if no **S** entry is present, page labels shall consist solely of a label prefix with no numeric portion. NOTE If the **P** entry (next) specifies the label prefix Contents, each page is simply labeled Contents with no page number. (If the **P** entry is also missing or empty, the page label is an empty string.)
P	text string	*(Optional)* The label prefix for page labels in this range.
St	integer	*(Optional)* The value of the numeric portion for the first page label in the range. Subsequent pages shall be numbered sequentially from this value, which shall be greater than or equal to 1. Default value: 1.

EXAMPLE The following example shows a document with pages labeled

i, ii, iii, iv, 1, 2, 3, A-8, A-9, …

```
1 0 obj
   << /Type /Catalog
      /PageLabels   << /Nums [   0 <<  /S /r  >>       % A number tree containing
                                 4 <<  /S /D  >>       % three page label dictionaries
                                 7 <<  /S /D
                                       /P (A-)
                                       /St 8
                                    >>
                             ]
                     >>
      …
   >>
endobj
```

12.4.3 Articles

Some types of documents may contain sequences of content items that are logically connected but not physically sequential.

EXAMPLE 1 A news story may begin on the first page of a newsletter and run over onto one or more nonconsecutive interior pages.

To represent such sequences of physically discontiguous but logically related items, a PDF document may define one or more *articles (PDF 1.1)*. The sequential flow of an article shall be defined by an *article thread*; the individual content items that make up the article are called *beads* on the thread. Conforming readers may provide navigation facilities to allow the user to follow a thread from one bead to the next.

The optional **Threads** entry in the document catalogue (see 7.7.2, “Document Catalog”) holds an array of *thread dictionaries* (Table 160) defining the document’s articles. Each individual bead within a thread shall be represented by a *bead dictionary* (Table 161). The thread dictionary's **F** entry shall refer to the first bead in the thread; the beads shall be chained together sequentially in a doubly linked list through their **N** (next) and **V** (previous) entries. In addition, for each page on which article beads appear, the page object (see 7.7.3, "Page Tree") shall contain a **B** entry whose value is an array of indirect references to the beads on the page, in drawing order.

Table 160 – Entries in a thread dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Thread** for a thread dictionary.
F	dictionary	*(Required;* shall *be an indirect reference)* The first bead in the thread.
I	dictionary	*(Optional)* A thread information dictionary containing information about the thread, such as its title, author, and creation date. The contents of this dictionary shall conform to the syntax for the document information dictionary (see 14.3.3, "Document Information Dictionary").

Table 161 – Entries in a bead dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Bead** for a bead dictionary.
T	dictionary	*(Required for the first bead of a thread; optional for all others*; *shall be an indirect reference)* The thread to which this bead belongs. *(PDF 1.1)* This entry shall be permitted only for the first bead of a thread. *(PDF 1.2)* It shall be permitted for any bead but required only for the first.
N	dictionary	*(Required; shall be an indirect reference)* The next bead in the thread. In the last bead, this entry shall refer to the first bead.
V	dictionary	*(Required; shall be an indirect reference)* The previous bead in the thread. In the first bead, this entry shall refer to the last bead.
P	dictionary	*(Required; shall be an indirect reference)* The page object representing the page on which this bead appears.
R	rectangle	*(Required)* A rectangle specifying the location of this bead on the page.

EXAMPLE 2 The following example shows a thread with three beads.

```
22  0  obj
    <<  /F  23 0 R
        /I   <<  /Title  (Man Bites Dog)  >>
    >>
endobj

23  0  obj
    <<  /T  22 0 R
        /N  24 0 R
        /V  25 0 R
        /P  8 0 R
        /R  [158 247 318 905]
    >>
endobj

24  0  obj
    <<  /T  22 0 R
```

```
        /N  25 0 R
        /V  23 0 R
        /P  8 0 R
        /R  [322 246 486 904]
    >>
endobj

25  0  obj
    <<  /T  22 0 R
        /N  23 0 R
        /V  24 0 R
        /P  10 0 R
        /R  [157 254 319 903]
    >>
endobj
```

12.4.4 Presentations

12.4.4.1 General

Some conforming readers may allow a document to be displayed in the form of a *presentation* or slide show, advancing from one page to the next either automatically or under user control. In addition, PDF 1.5 introduces the ability to advance between different states of the same page (see 12.4.4.2, “Sub-page Navigation”).

NOTE 1 PDF 1.4 introduces a different mechanism, known as alternate presentations, for slide show displays, described in 13.5, “Alternate Presentations.”

A page object (see 7.7.3, “Page Tree”) may contain two optional entries, **Dur** and **Trans** *(PDF 1.1)*, to specify how to display that page in presentation mode. The **Trans** entry shall contain a *transition dictionary* describing the style and duration of the visual transition to use when moving from another page to the given page during a presentation. Table 162 shows the contents of the transition dictionary. (Some of the entries shown are needed only for certain transition styles, as indicated in the table.)

The **Dur** entry in the page object specifies the page’s *display duration* (also called its *advance timing*): the maximum length of time, in seconds, that the page shall be displayed before the presentation automatically advances to the next page.

NOTE 2 The user can advance the page manually before the specified time has expired.

If no **Dur** entry is specified in the page object, the page shall not advance automatically.

Table 162 – Entries in a transition dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Trans** for a transition dictionary.
S	name	*(Optional)* The *transition style* that shall be used when moving to this page from another during a presentation. Default value: R. Split Two lines sweep across the screen, revealing the new page. The lines may be either horizontal or vertical and may move inward from the edges of the page or outward from the center, as specified by the **Dm** and **M** entries, respectively. Blinds Multiple lines, evenly spaced across the screen, synchronously sweep in the same direction to reveal the new page. The lines may be either horizontal or vertical, as specified by the **Dm** entry. Horizontal lines move downward; vertical lines move to the right. Box A rectangular box sweeps inward from the edges of the page or outward from the center, as specified by the **M** entry, revealing the new page. Wipe A single line sweeps across the screen from one edge to the other in the direction specified by the **Di** entry, revealing the new page. Dissolve The old page dissolves gradually to reveal the new one. Glitter Similar to Dissolve, except that the effect sweeps across the page in a wide band moving from one side of the screen to the other in the direction specified by the **Di** entry. R The new page simply replaces the old one with no special transition effect; the **D** entry shall be ignored. Fly *(PDF 1.5)* Changes are flown out or in (as specified by **M**), in the direction specified by **Di**, to or from a location that is offscreen except when **Di** is None.
		Push *(PDF 1.5)* The old page slides off the screen while the new page slides in, pushing the old page out in the direction specified by **Di**. Cover *(PDF 1.5)* The new page slides on to the screen in the direction specified by **Di**, covering the old page. Uncover *(PDF 1.5)* The old page slides off the screen in the direction specified by **Di**, uncovering the new page in the direction specified by **Di**. Fade *(PDF 1.5)* The new page gradually becomes visible through the old one.
D	number	*(Optional)* The duration of the transition effect, in seconds. Default value: 1.
Dm	name	*(Optional; **Split** and **Blinds** transition styles only)* The dimension in which the specified transition effect shall occur: H Horizontal V Vertical Default value: H.
M	name	*(Optional; **Split**, **Box** and **Fly** transition styles only)* The direction of motion for the specified transition effect: I Inward from the edges of the page O Outward from the center of the page Default value: I.

Table 162 – Entries in a transition dictionary (continued)

Key	Type	Value
Di	number or name	*(Optional; Wipe*, *Glitter*, ***Fly***, ***Cover***, ***Uncover*** and ***Push*** *transition styles only)* The direction in which the specified transition effect shall moves, expressed in degrees counterclockwise starting from a left-to-right direction. (This differs from the page object's **Rotate** entry, which is measured clockwise from the top.) If the value is a number, it shall be one of: 0 Left to right 90 Bottom to top (Wipe only) 180 Right to left (Wipe only) 270 Top to bottom 315 Top-left to bottom-right (Glitter only) If the value is a name, it shall be None, which is relevant only for the Fly transition when the value of SS is not 1.0. Default value: 0.
SS	number	(*Optional; PDF 1.5; Fly transition style only)* The starting or ending scale at which the changes shall be drawn. If **M** specifies an inward transition, the scale of the changes drawn shall progress from **SS** to 1.0 over the course of the transition. If **M** specifies an outward transition, the scale of the changes drawn shall progress from 1.0 to **SS** over the course of the transition Default: 1.0.
B	boolean	(*Optional; PDF 1.5;* ***Fly*** *transition style only*) If **true**, the area that shall be flown in is rectangular and opaque. Default: **false**.

NOTE 3 Figure 56 illustrates the relationship between transition duration (**D** in the transition dictionary) and display duration (**Dur** in the page object). Note that the transition duration specified for a page (page 2 in the figure) governs the transition to that page from another page; the transition from the page is governed by the next page's transition duration.

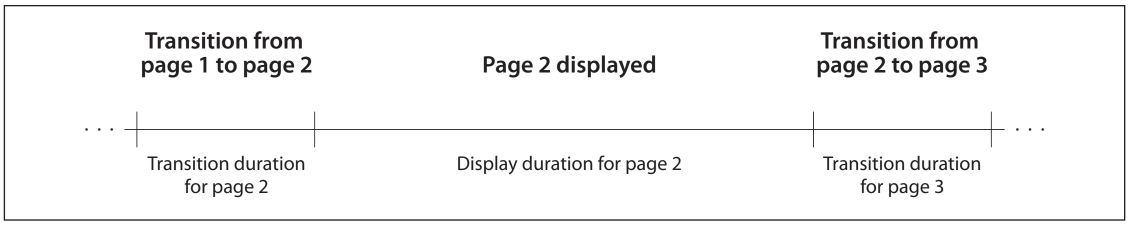

Transition from page 1 to page 2
Page 2 displayed
Transition from page 2 to page 3
Transition duration for page 2
Display duration for page 2
Transition duration for page 3

Figure 56 – Presentation timing

EXAMPLE The following example shows the presentation parameters for a page to be displayed for 5 seconds. Before the page is displayed, there is a 3.5-second transition in which two vertical lines sweep outward from the center to the edges of the page.

```
10 0 obj
   << /Type /Page
      /Parent 4 0 R
      /Contents 16 0 R
      /Dur 5
      /Trans  <<   /Type /Trans
                   /D 3.5
                   /S /Split
                   /Dm /V
                   /M /O
              >>
   >>
endobj
```

12.4.4.2 Sub-page Navigation

Sub-page navigation (PDF 1.5) provides the ability to navigate not only between pages but also between different states of the same page.

NOTE 1 A single page in a PDF presentation could have a series of bullet points that could be individually turned on and off. In such an example, the bullets would be represented by optional content (see 8.11.2, "Optional Content Groups"), and each state of the page would be represented as a *navigation node*.

NOTE 2 Conforming readers should save the state of optional content groups when a user enters presentation mode and restore it when presentation mode ends. This ensures, for example, that transient changes to bullets do not affect the printing of the document.

A navigation node dictionary (see Table 163) specifies actions to execute when the user makes a navigation request.

EXAMPLE Pressing an arrow key.

The navigation nodes on a page form a doubly linked list by means of their **Next** and **Prev** entries. The primary node on a page shall be determined by the optional **PresSteps** entry in a page dictionary (see Table 30).

NOTE 3 A conforming reader should respect navigation nodes only when in presentation mode (see 12.4.4, "Presentations").

Table 163 – Entries in a navigation node dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; shall be **NavNode** for a navigation node dictionary.
NA	dictionary	*(Optional)* An action (which may be the first in a sequence of actions) that shall be executed when a user navigates forward.
PA	dictionary	*(Optional)* An action (which may be the first in a sequence of actions) that shall be executed when a user navigates backward.
Next	dictionary	*(Optional)* The next navigation node, if any.
Prev	dictionary	*(Optional)* The previous navigation node, if any.
Dur	number	*(Optional)* The maximum number of seconds before the conforming reader shall automatically advance forward to the next navigation node. If this entry is not specified, no automatic advance shall occur.

A conforming reader shall maintain a *current* navigation node. When a user navigates to a page, if the page dictionary has a **PresSteps** entry, the node specified by that entry shall become the current node. (Otherwise, there is no current node.) If the user requests to navigate forward (such as an arrow key press) and there is a current navigation node, the following shall occur:

a) The sequence of actions specified by **NA** (if present) shall be executed.

   If **NA** specifies an action that navigates to another page, the following actions for navigating to another page take place, and **Next** should not be present.

b) The node specified by **Next** (if present) shall become the new current navigation node.

Similarly, if the user requests to navigate backward and there is a current navigation node, the following shall occur:

a) The sequence of actions specified by **PA** (if present) shall be executed.

If **PA** specifies an action that navigates to another page, the following actions for navigating to another page take place, and **Prev** should not be present.

b) The node specified by **Prev** (if present) shall become the new current navigation node.

Transition effects, similar to the page transitions described earlier, may be specified as transition actions that are part of the **NA** or **PA** sequence; see 12.6.4.14, “Transition Actions.”

If the user requests to navigate to another page (regardless of whether there is a current node) and that page’s dictionary contains a **PresSteps** entry, the following shall occur:

a) The navigation node represented by **PresSteps** shall become the current node.

b) If the navigation request was forward, or if the navigation request was for random access (such as by clicking on a link), the actions specified by **NA** shall be executed and the node specified by **Next** shall become the new current node, as described previously.

   If the navigation request was backward, the actions specified by **PA** shall be executed and the node specified by **Prev** shall become the new current node, as described previously.

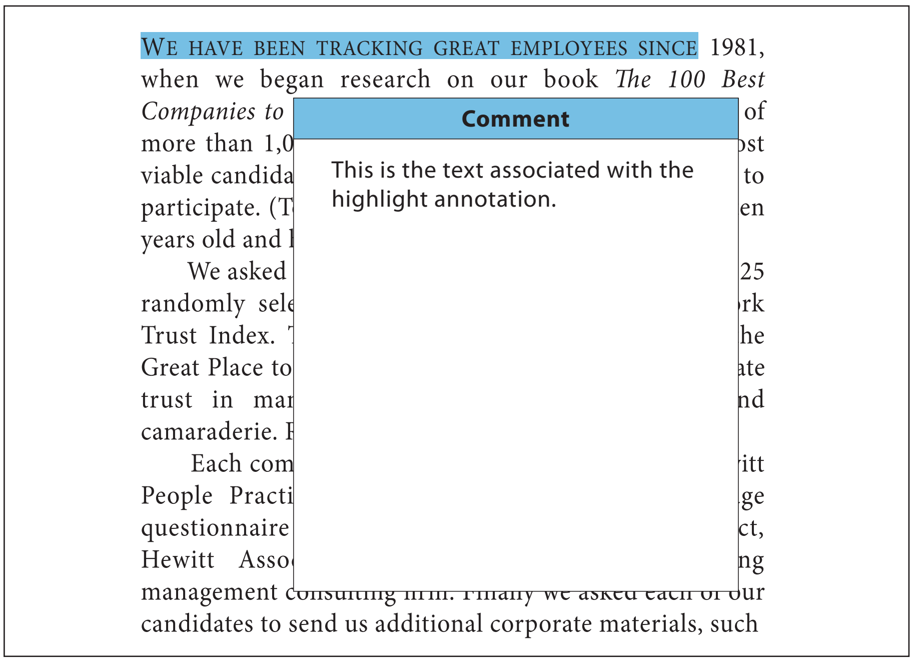
c) The conforming reader shall make the new page the current page and shall display it. Any page transitions specified by the **Trans** entry of the page dictionary shall be performed.

12.5 Annotations

12.5.1 General

An *annotation* associates an object such as a note, sound, or movie with a location on a page of a PDF document, or provides a way to interact with the user by means of the mouse and keyboard. PDF includes a wide variety of standard annotation types, described in detail in 12.5.6, “Annotation Types.”

Many of the standard annotation types may be displayed in either the *open* or the *closed* state. When closed, they appear on the page in some distinctive form, such as an icon, a box, or a rubber stamp, depending on the specific annotation type. When the user *activates* the annotation by clicking it, it exhibits its associated object, such as by opening a pop-up window displaying a text note (Figure 57) or by playing a sound or a movie.



WE HAVE BEEN TRACKING GREAT EMPLOYEES SINCE 1981,
when we began research on our book *The 100 Best*
Companies to of
more than 1,0 ost
viable candida to
participate. (T en
years old and l
We asked 25
randomly sel rk
Trust Index. he
Great Place to ate
trust in ma nd
camaraderie. I
Each com itt
People Practi ge
questionnaire ct,
Hewitt Asso ng
management consulting firm. Finally we asked each of our
candidates to send us additional corporate materials, such

Comment

This is the text associated with the highlight annotation.

Figure 57 – Open annotation

Conforming readers may permit the user to navigate through the annotations on a page by using the keyboard (in particular, the tab key). Beginning with PDF 1.5, PDF producers may make the navigation order explicit with the optional **Tabs** entry in a page object (see Table 30). The following are the possible values for this entry:

- R (row order): Annotations shall be visited in rows running horizontally across the page. The direction within a row shall be determined by the **Direction** entry in the viewer preferences dictionary (see 12.2, “Viewer Preferences”). The first annotation that shall be visited is the first annotation in the topmost row. When the end of a row is encountered, the first annotation in the next row shall be visited.

- C (column order): Annotations shall be visited in columns running vertically up and down the page. Columns shall be ordered by the **Direction** entry in the viewer preferences dictionary (see 12.2, “Viewer Preferences”). The first annotation that shall be visited is the one at the top of the first column. When the end of a column is encountered, the first annotation in the next column shall be visited.

- S (structure order): Annotations shall be visited in the order in which they appear in the structure tree (see 14.7, “Logical Structure”). The order for annotations that are not included in the structure tree shall be determined in a manner of the conforming reader's choosing.

These descriptions assume the page is being viewed in the orientation specified by the **Rotate** entry.

Conceptually, the behaviour of each annotation type may be implemented by a software module called an *annotation handler*. A conforming reader shall provide annotation handlers for all of the conforming annotation types. The set of annotation types is extensible. A conforming reader shall provide certain expected behaviour for all annotation types that it does not recognize, as documented in 12.5.2, “Annotation Dictionaries.”

12.5.2 Annotation Dictionaries

The optional **Annots** entry in a page object (see 7.7.3, “Page Tree”) holds an array of *annotation dictionaries*, each representing an annotation associated with the given page. Table 164 shows the required and optional entries that are common to all annotation dictionaries. The dictionary may contain additional entries specific to a particular annotation type; see the descriptions of individual annotation types in 12.5.6, “Annotation Types,” for details. A given annotation dictionary shall be referenced from the **Annots** array of only one page. This requirement applies only to the annotation dictionary itself, not to subsidiary objects, which may be shared among multiple annotations.

Table 164 – Entries common to all annotation dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Annot** for an annotation dictionary.
Subtype	name	*(Required)* The type of annotation that this dictionary describes; see Table 169 for specific values.
Rect	rectangle	*(Required)* The *annotation rectangle*, defining the location of the annotation on the page in default user space units.
Contents	text string	*(Optional)* Text that shall be displayed for the annotation or, if this type of annotation does not display text, an alternate description of the annotation's contents in human-readable form. In either case, this text is useful when extracting the document's contents in support of accessibility to users with disabilities or for other purposes (see 14.9.3, "Alternate Descriptions"). See 12.5.6, "Annotation Types" for more details on the meaning of this entry for each annotation type.
P	dictionary	*(Optional except as noted below; PDF 1.3; not used in FDF files)* An indirect reference to the page object with which this annotation is associated. This entry shall be present in screen annotations associated with rendition actions (*PDF 1.5*; see 12.5.6.18, "Screen Annotations" and 12.6.4.13, "Rendition Actions").
NM	text string	*(Optional; PDF 1.4)* The *annotation name*, a text string uniquely identifying it among all the annotations on its page.
M	date or text string	*(Optional; PDF 1.1)* The date and time when the annotation was most recently modified. The format should be a date string as described in 7.9.4, "Dates," but conforming readers shall accept and display a string in any format.
F	integer	*(Optional; PDF 1.1)* A set of flags specifying various characteristics of the annotation (see 12.5.3, "Annotation Flags"). Default value: 0.
AP	dictionary	*(Optional; PDF 1.2)* An *appearance dictionary* specifying how the annotation shall be presented visually on the page (see 12.5.5, "Appearance Streams"). Individual annotation handlers may ignore this entry and provide their own appearances.
AS	name	*(Required if the appearance dictionary* **AP** *contains one or more subdictionaries; PDF 1.2)* The annotation's *appearance state*, which selects the applicable appearance stream from an appearance subdictionary (see Section 12.5.5, "Appearance Streams").

Table 164 – Entries common to all annotation dictionaries (continued)

Key	Type	Value
Border	array	*(Optional)* An array specifying the characteristics of the annotation's border, which shall be drawn as a rounded rectangle. *(PDF 1.0)* The array consists of three numbers defining the horizontal corner radius, vertical corner radius, and border width, all in default user space units. If the corner radii are 0, the border has square (not rounded) corners; if the border width is 0, no border is drawn. *(PDF 1.1)* The array may have a fourth element, an optional *dash array* defining a pattern of dashes and gaps that shall be used in drawing the border. The dash array shall be specified in the same format as in the line dash pattern parameter of the graphics state (see 8.4.3.6, "Line Dash Pattern"). EXAMPLE A **Border** value of [0 0 1 [3 2]] specifies a border 1 unit wide, with square corners, drawn with 3-unit dashes alternating with 2-unit gaps. NOTE *(PDF 1.2)* The dictionaries for some annotation types (such as free text and polygon annotations) can include the **BS** entry. That entry specifies a border style dictionary that has more settings than the array specified for the **Border** entry. If an annotation dictionary includes the **BS** entry, then the **Border** entry is ignored. Default value: [0 0 1].
C	array	*(Optional; PDF 1.1)* An array of numbers in the range 0.0 to 1.0, representing a colour used for the following purposes: The background of the annotation's icon when closed The title bar of the annotation's pop-up window The border of a link annotation The number of array elements determines the colour space in which the colour shall be defined: 0 No colour; transparent 1 **DeviceGray** 3 **DeviceRGB** 4 **DeviceCMYK**
StructParent	integer	*(Required if the annotation is a structural content item; PDF 1.3)* The integer key of the annotation's entry in the structural parent tree (see 14.7.4.4, "Finding Structure Elements from Content Items").
OC	dictionary	*(Optional; PDF 1.5)* An optional content group or optional content membership dictionary (see 8.11, "Optional Content") specifying the optional content properties for the annotation. Before the annotation is drawn, its visibility shall be determined based on this entry as well as the annotation flags specified in the **F** entry (see 12.5.3, "Annotation Flags"). If it is determined to be invisible, the annotation shall be skipped, as if it were not in the document.

12.5.3 Annotation Flags

The value of the annotation dictionary's **F** entry is an integer interpreted as one-bit flags specifying various characteristics of the annotation. Bit positions within the flag word shall be numbered from low-order to high-order, with the lowest-order bit numbered 1. Table 165 shows the meanings of the flags; all other bits of the integer shall be set to 0.

Table 165 – Annotation flags

Bit position	Name	Meaning
1	Invisible	If set, do not display the annotation if it does not belong to one of the standard annotation types and no annotation handler is available. If clear, display such an unknown annotation using an appearance stream specified by its appearance dictionary, if any (see 12.5.5, “Appearance Streams”).
2	Hidden	*(PDF 1.2)* If set, do not display or print the annotation or allow it to interact with the user, regardless of its annotation type or whether an annotation handler is available. NOTE 1 In cases where screen space is limited, the ability to hide and show annotations selectively can be used in combination with appearance streams (see 12.5.5, “Appearance Streams”) to display auxiliary pop-up information similar in function to online help systems.
3	Print	*(PDF 1.2)* If set, print the annotation when the page is printed. If clear, never print the annotation, regardless of whether it is displayed on the screen. NOTE 2 This can be useful for annotations representing interactive pushbuttons, which would serve no meaningful purpose on the printed page.
4	NoZoom	*(PDF 1.3)* If set, do not scale the annotation’s appearance to match the magnification of the page. The location of the annotation on the page (defined by the upper-left corner of its annotation rectangle) shall remain fixed, regardless of the page magnification. See further discussion following this Table.
5	NoRotate	*(PDF 1.3)* If set, do not rotate the annotation’s appearance to match the rotation of the page. The upper-left corner of the annotation rectangle shall remain in a fixed location on the page, regardless of the page rotation. See further discussion following this Table.
6	NoView	*(PDF 1.3)* If set, do not display the annotation on the screen or allow it to interact with the user. The annotation may be printed (depending on the setting of the Print flag) but should be considered hidden for purposes of on-screen display and user interaction.
7	ReadOnly	*(PDF 1.3)* If set, do not allow the annotation to interact with the user. The annotation may be displayed or printed (depending on the settings of the NoView and Print flags) but should not respond to mouse clicks or change its appearance in response to mouse motions. This flag shall be ignored for widget annotations; its function is subsumed by the ReadOnly flag of the associated form field (see Table 221).
8	Locked	*(PDF 1.4)* If set, do not allow the annotation to be deleted or its properties (including position and size) to be modified by the user. However, this flag does not restrict changes to the annotation’s contents, such as the value of a form field.
9	ToggleNoView	*(PDF 1.5)* If set, invert the interpretation of the NoView flag for certain events. NOTE 3 A typical use is to have an annotation that appears only when a mouse cursor is held over it.
10	LockedContents	*(PDF 1.7)* If set, do not allow the contents of the annotation to be modified by the user. This flag does not restrict deletion of the annotation or changes to other annotation properties, such as position and size.

If the NoZoom flag is set, the annotation shall always maintain the same fixed size on the screen and shall be unaffected by the magnification level at which the page itself is displayed. Similarly, if the NoRotate flag is set, the annotation shall retain its original orientation on the screen when the page is rotated (by changing the Rotate entry in the page object; see 7.7.3, “Page Tree”).

In either case, the annotation’s position shall be determined by the coordinates of the upper-left corner of its annotation rectangle, as defined by the **Rect** entry in the annotation dictionary and interpreted in the default user space of the page. When the default user space is scaled or rotated, the positions of the other three corners of the annotation rectangle are different in the altered user space than they were in the original user space. The conforming reader shall perform this alteration automatically. However, it shall not actually change the annotation’s **Rect** entry, which continues to describe the annotation’s relationship with the unscaled, unrotated user space.

NOTE Figure 58 shows how an annotation whose NoRotate flag is set remains upright when the page it is on is rotated 90 degrees clockwise. The upper-left corner of the annotation remains at the same point in default user space; the annotation pivots around that point.

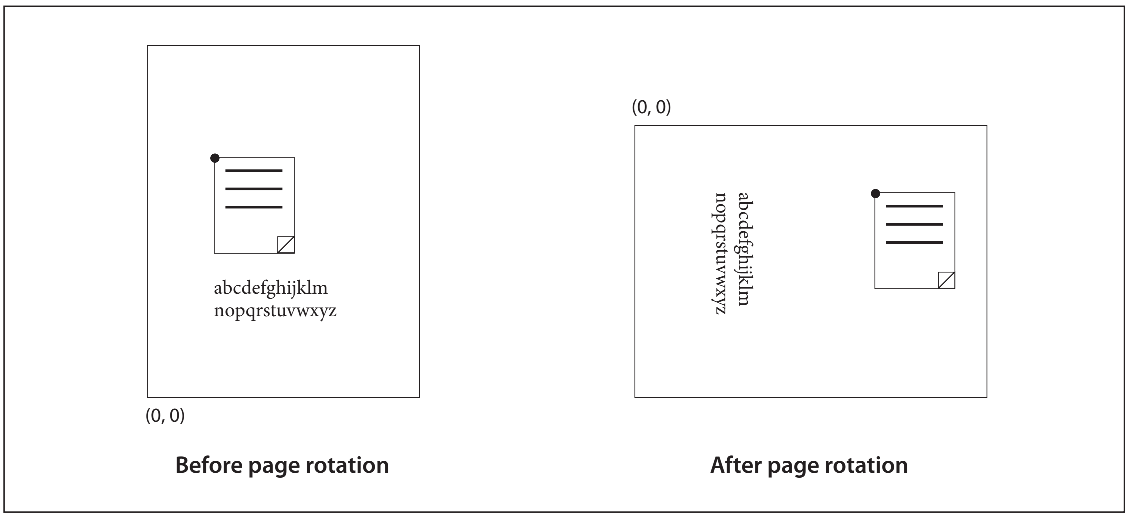


Figure 58 – Coordinate adjustment with the NoRotate flag

12.5.4 Border Styles

An annotation may optionally be surrounded by a border when displayed or printed. If present, the border shall be drawn completely inside the annotation rectangle. In PDF 1.1, the characteristics of the border shall be specified by the **Border** entry in the annotation dictionary (see Table 164). Beginning with PDF 1.2, the border characteristics for some types of annotations may instead be specified in a *border style dictionary* designated by the annotation’s **BS** entry. Such dictionaries may also be used to specify the width and dash pattern for the lines drawn by line, square, circle, and ink annotations. Table 166 summarizes the contents of the border style dictionary. If neither the **Border** nor the **BS** entry is present, the border shall be drawn as a solid line with a width of 1 point.

Table 166 – Entries in a border style dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Border** for a border style dictionary.
W	number	*(Optional)* The border width in points. If this value is 0, no border shall drawn. Default value: 1.

Table 166 – Entries in a border style dictionary (continued)

Key	Type	Value
S	name	*(Optional)* The border style: S (Solid) A solid rectangle surrounding the annotation. D (Dashed) A dashed rectangle surrounding the annotation. The dash pattern may be specified by the **D** entry. B (Beveled) A simulated embossed rectangle that appears to be raised above the surface of the page. I (Inset) A simulated engraved rectangle that appears to be recessed below the surface of the page. U (Underline) A single line along the bottom of the annotation rectangle. A conforming reader shall tolerate other border styles that it does not recognize and shall use the default value.
D	array	*(Optional)* A *dash array* defining a pattern of dashes and gaps that shall be used in drawing a dashed border (border style D in the **S** entry). The dash array shall be specified in the same format as in the line dash pattern parameter of the graphics state (see 8.4.3.6, “Line Dash Pattern”). The dash phase is not specified and shall be assumed to be 0. EXAMPLE A **D** entry of [3 2] specifies a border drawn with 3-point dashes alternating with 2-point gaps. Default value: [3].

Beginning with PDF 1.5, some annotations (square, circle, and polygon) may have a **BE** entry, which is a *border effect dictionary* that specifies an effect that shall be applied to the border of the annotations. Beginning with PDF 1.6, the free text annotation may also have a **BE** entry. Table 167 describes the entries in a border effect dictionary.

Table 167 – Entries in a border effect dictionary

Key	Type	Value
S	name	*(Optional)* A name representing the border effect to apply. Possible values are: S No effect: the border shall be as described by the annotation dictionary’s **BS** entry. C The border should appear “cloudy”. The width and dash array specified by **BS** shall be honored. Default value: S.
I	number	*(Optional; valid only if the value of **S** is C)* A number describing the intensity of the effect, in the range 0 to 2. Default value: 0.

12.5.5 Appearance Streams

Beginning with PDF 1.2, an annotation may specify one or more *appearance streams* as an alternative to the simple border and colour characteristics available in earlier versions. Appearance streams enable the annotation to be presented visually in different ways to reflect its interactions with the user. Each appearance stream is a form XObject (see 8.10, “Form XObjects”): a self-contained content stream that shall be rendered inside the annotation rectangle.

The algorithm outlined in this sub-clause shall be used to map from the coordinate system of the appearance XObject (as defined by its **Matrix** entry; see Table 97) to the annotation’s rectangle in default user space:

Algorithm: Appearance streams

a) The appearance's bounding box (specified by its **BBox** entry) shall be transformed, using **Matrix**, to produce a quadrilateral with arbitrary orientation. The *transformed appearance box* is the smallest upright rectangle that encompasses this quadrilateral.

b) A matrix *A* shall be computed that scales and translates the transformed appearance box to align with the edges of the annotation's rectangle (specified by the **Rect** entry). *A* maps the lower-left corner (the corner with the smallest *x* and *y* coordinates) and the upper-right corner (the corner with the greatest *x* and *y* coordinates) of the transformed appearance box to the corresponding corners of the annotation's rectangle.

c) **Matrix** shall be concatenated with *A* to form a matrix *AA* that maps from the appearance's coordinate system to the annotation's rectangle in default user space:

   AA = **Matrix** ¥ *A*

The annotation may be further scaled and rotated if either the NoZoom or NoRotate flag is set (see 12.5.3, "Annotation Flags"). Any transformation applied to the annotation as a whole shall also applied to the appearance within it.

Starting with PDF 1.4, an annotation appearance may include transparency. If the appearance's stream dictionary does not contain a **Group** entry, it shall be treated as a non-isolated, non-knockout transparency group. Otherwise, the isolated and knockout values specified in the group dictionary (see 11.6.6, "Transparency Group XObjects") shall be used.

The transparency group shall be composited with a backdrop consisting of the page content along with any previously painted annotations, using a blend mode of **Normal**, an alpha constant of 1.0, and a soft mask of **None**.

NOTE 1 If a transparent annotation appearance is painted over an annotation that is drawn without using an appearance stream, the effect is implementation-dependent. This is because such annotations are sometimes drawn by means that do not conform to the PDF imaging model. Also, the effect of highlighting a transparent annotation appearance is implementation-dependent.

An annotation may define as many as three separate appearances:

- The *normal appearance* shall be used when the annotation is not interacting with the user. This appearance is also used for printing the annotation.
- The *rollover appearance* shall be used when the user moves the cursor into the annotation's active area without pressing the mouse button.
- The *down appearance* shall be used when the mouse button is pressed or held down within the annotation's active area.

NOTE 2 As used here, the term mouse denotes a generic pointing device that controls the location of a cursor on the screen and has at least one button that can be pressed, held down, and released. See 12.6.3, "Trigger Events," for further discussion.

The normal, rollover, and down appearances shall be defined in an *appearance dictionary*, which in turn is the value of the AP entry in the annotation dictionary (see Table 164). Table 168 shows the contents of the appearance dictionary.

Table 168 – Entries in an appearance dictionary

Key	Type	Value
N	stream or dictionary	*(Required)* The annotation’s normal appearance.
R	stream or dictionary	*(Optional)* The annotation’s rollover appearance. Default value: the value of the **N** entry.
D	stream or dictionary	*(Optional)* The annotation’s down appearance. Default value: the value of the **N** entry.

Each entry in the appearance dictionary may contain either a single appearance stream or an *appearance subdictionary*. In the latter case, the subdictionary shall define multiple appearance streams corresponding to different *appearance states* of the annotation.

EXAMPLE An annotation representing an interactive check box may have two appearance states named On and Off. Its appearance dictionary may be defined as

```
/AP <<  /N  <<  /On  formXObject1
                /Off  formXObject2
             >>
        /D  <<  /On  formXObject3
                /Off  formXObject4
            >>
    >>
```

where *formXObject*$_1$ and *formXObject*$_2$ define the check box’s normal appearance in its checked and unchecked states, and *formXObject*$_3$ and *formXObject*$_4$ provide visual feedback, such as emboldening its outline, when the user clicks it. (No **R** entry is defined because no special appearance is needed when the user moves the cursor over the check box without pressing the mouse button.) The choice between the checked and unchecked appearance states is determined by the **AS** entry in the annotation dictionary (see Table 164).

NOTE 3 If a conforming reader does not have native support for a particular annotation type conforming readers shall display the annotation with its normal (**N**) appearance. Conforming readers shall also attempt to provide reasonable behavior (such as displaying nothing) if an annotation’s **AS** entry designates an appearance state for which no appearance is defined in the appearance dictionary.

For convenience in managing appearance streams that are used repeatedly, the **AP** entry in a PDF document’s name dictionary (see 7.7.4, “Name Dictionary”) may contain a name tree mapping name strings to appearance streams. The name strings have no standard meanings; no PDF objects may refer to appearance streams by name.

12.5.6 Annotation Types

12.5.6.1 General

PDF supports the standard annotation types listed in Table 169. The following sub-clauses describe each of these types in detail.

The values in the first column of Table 169 represent the value of the annotation dictionary’s **Subtype** entry. The third column indicates whether the annotation is a *markup annotation*, as described in 12.5.6.2, “Markup Annotations.” The sub-clause also provides more information about the value of the **Contents** entry for different annotation types.

Table 169 – Annotation types

Annotation type	Description	Markup	Discussed in sub-clause
Text	Text annotation	Yes	12.5.6.4, “Text Annotations”
Link	Link annotation	No	12.5.6.5, “Link Annotations”
FreeText	*(PDF 1.3)* Free text annotation	Yes	12.5.6.6, “Free Text Annotations”
Line	*(PDF 1.3)* Line annotation	Yes	12.5.6.7, “Line Annotations”
Square	*(PDF 1.3)* Square annotation	Yes	12.5.6.8, “Square and Circle Annotations”
Circle	*(PDF 1.3)* Circle annotation	Yes	12.5.6.8, “Square and Circle Annotations”
Polygon	*(PDF 1.5)* Polygon annotation	Yes	12.5.6.9, “Polygon and Polyline Annotations”
PolyLine	*(PDF 1.5)* Polyline annotation	Yes	12.5.6.9, “Polygon and Polyline Annotations”
Highlight	*(PDF 1.3)* Highlight annotation	Yes	12.5.6.10, “Text Markup Annotations”
Underline	*(PDF 1.3)* Underline annotation	Yes	12.5.6.10, “Text Markup Annotations”
Squiggly	*(PDF 1.4)* Squiggly-underline annotation	Yes	12.5.6.10, “Text Markup Annotations”
StrikeOut	*(PDF 1.3)* Strikeout annotation	Yes	12.5.6.10, “Text Markup Annotations”
Stamp	*(PDF 1.3)* Rubber stamp annotation	Yes	12.5.6.12, “Rubber Stamp Annotations”
Caret	*(PDF 1.5)* Caret annotation	Yes	12.5.6.11, “Caret Annotations”
Ink	*(PDF 1.3)* Ink annotation	Yes	12.5.6.13, “Ink Annotations”
Popup	*(PDF 1.3)* Pop-up annotation	No	12.5.6.14, “Pop-up Annotations”
FileAttachment	*(PDF 1.3)* File attachment annotation	Yes	12.5.6.15, “File Attachment Annotations”
Sound	*(PDF 1.2)* Sound annotation	Yes	12.5.6.16, “Sound Annotations”
Movie	*(PDF 1.2)* Movie annotation	No	12.5.6.17, “Movie Annotations”
Widget	*(PDF 1.2)* Widget annotation	No	12.5.6.19, “Widget Annotations”
Screen	*(PDF 1.5)* Screen annotation	No	12.5.6.18, “Screen Annotations”
PrinterMark	*(PDF 1.4)* Printer’s mark annotation	No	12.5.6.20, “Printer’s Mark Annotations”
TrapNet	*(PDF 1.3)* Trap network annotation	No	12.5.6.21, “Trap Network Annotations”
Watermark	*(PDF 1.6)* Watermark annotation	No	12.5.6.22, “Watermark Annotations”
3D	*(PDF 1.6)* 3D annotation	No	13.6.2, “3D Annotations”
Redact	*(PDF 1.7)* Redact annotation	Yes	12.5.6.23, “Redaction Annotations”

12.5.6.2 Markup Annotations

As mentioned in 12.5.2, “Annotation Dictionaries,” the meaning of an annotation’s **Contents** entry varies by annotation type. Typically, it is the text that shall be displayed for the annotation or, if the annotation does not display text, an alternate description of the annotation’s contents in human-readable form. In either case, the **Contents** entry is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes (see 14.9.3, “Alternate Descriptions”).

Many annotation types are defined as *markup annotations* because they are used primarily to mark up PDF documents (see Table 170). These annotations have text that appears as part of the annotation and may be displayed in other ways by a conforming reader, such as in a Comments pane.

Markup annotations may be divided into the following groups:

- Free text annotations display text directly on the page. The annotation’s **Contents** entry specifies the displayed text.
- Most other markup annotations have an associated pop-up window that may contain text. The annotation’s **Contents** entry specifies the text that shall be displayed when the pop-up window is opened. These include text, line, square, circle, polygon, polyline, highlight, underline, squiggly-underline, strikeout, rubber stamp, caret, ink, and file attachment annotations.
- Sound annotations do not have a pop-up window but may also have associated text specified by the **Contents** entry.

When separating text into paragraphs, a CARRIAGE RETURN (0Dh) shall be used and not, for example, a LINE FEED character (0Ah).

NOTE 1 A subset of markup annotations is called text markup annotations (see 12.5.6.10, “Text Markup Annotations”).

The remaining annotation types are not considered markup annotations:

- The pop-up annotation type shall not appear by itself; it shall be associated with a markup annotation that uses it to display text.

NOTE 2 If an annotation has no parent, the **Contents** entry shall represent the text of the annotation, otherwise it shall be ignored by a conforming reader.

- For all other annotation types (**Link**, **Movie**, **Widget**, **PrinterMark**, and **TrapNet**), the **Contents** entry shall provide an alternate representation of the annotation’s contents in human-readable form, which is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes (see 14.9.3, “Alternate Descriptions”).

Table 170 lists entries that apply to all markup annotations.

Table 170 – Additional entries specific to markup annotations

Key	Type	Value
T	text string	*(Optional; PDF 1.1)* The text label that shall be displayed in the title bar of the annotation’s pop-up window when open and active. This entry shall identify the user who added the annotation.
Popup	dictionary	*(Optional; PDF 1.3)* An indirect reference to a pop-up annotation for entering or editing the text associated with this annotation.

Table 170 – Additional entries specific to markup annotations (continued)

Key	Type	Value
CA	number	*(Optional; PDF 1.4)* The constant opacity value that shall be used in painting the annotation (see Sections 11.2, “Overview of Transparency,” and 11.3.7, “Shape and Opacity Computations”). This value shall apply to all visible elements of the annotation in its closed state (including its background and border) but not to the pop-up window that appears when the annotation is opened. The specified value shall not used if the annotation has an appearance stream (see 12.5.5, “Appearance Streams”); in that case, the appearance stream shall specify any transparency. (However, if the compliant viewer regenerates the annotation’s appearance stream, it may incorporate the **CA** value into the stream’s content.) The implicit blend mode (see 11.3.5, “Blend Mode””) is **Normal**. Default value: 1.0. If no explicit appearance stream is defined for the annotation, it may be painted by implementation-dependent means that do not necessarily conform to the PDF imaging model; in this case, the effect of this entry is implementation-dependent as well.
RC	text string or text stream	*(Optional; PDF 1.5)* A rich text string (see 12.7.3.4, “Rich Text Strings”) that shall be displayed in the pop-up window when the annotation is opened.
CreationDate	date	*(Optional; PDF 1.5)* The date and time (7.9.4, “Dates”) when the annotation was created.
IRT	dictionary	*(Required if an **RT** entry is present, otherwise optional; PDF 1.5)* A reference to the annotation that this annotation is “in reply to.” Both annotations shall be on the same page of the document. The relationship between the two annotations shall be specified by the **RT** entry. If this entry is present in an FDF file (see 12.7.7, “Forms Data Format”), its type shall not be a dictionary but a text string containing the contents of the **NM** entry of the annotation being replied to, to allow for a situation where the annotation being replied to is not in the same FDF file.
Subj	text string	*(Optional; PDF 1.5)* Text representing a short description of the subject being addressed by the annotation.
RT	name	*(Optional; meaningful only if **IRT** is present; PDF 1.6)* A name specifying the relationship (the “reply type”) between this annotation and one specified by **IRT**. Valid values are: **R** The annotation shall be considered a reply to the annotation specified by **IRT**. Conforming readers shall not display replies to an annotation individually but together in the form of threaded comments. **Group** The annotation shall be grouped with the annotation specified by **IRT**; see the discussion following this Table. Default value: **R**.
IT	name	*(Optional; PDF 1.6)* A name describing the *intent* of the markup annotation. Intents allow conforming readers to distinguish between different uses and behaviors of a single markup annotation type. If this entry is not present or its value is the same as the annotation type, the annotation shall have no explicit intent and should behave in a generic manner in a conforming reader. Free text annotations (Table 174), line annotations (Table 175), polygon annotations (Table 178), and *(PDF 1.7)* polyline annotations (Table 178) have defined intents, whose values are enumerated in the corresponding tables.

Table 170 – Additional entries specific to markup annotations (continued)

Key	Type	Value
ExData	dictionary	*(Optional; PDF 1.7)* An *external data dictionary* specifying data that shall be associated with the annotation. This dictionary contains the following entries: **Type** *(optional)* If present, shall be **ExData**. **Subtype** *(required)* a name specifying the type of data that the markup annotation shall be associated with. The only defined value is **Markup3D**. Table 298 lists the values that correspond to a subtype of **Markup3D**.

In PDF 1.6, a set of annotations may be grouped so that they function as a single unit when a user interacts with them. The group consists of a *primary annotation*, which shall not have an **IRT** entry, and one or more *subordinate annotations*, which shall have an **IRT** entry that refers to the primary annotation and an **RT** entry whose value is **Group**.

Some entries in the primary annotation are treated as "group attributes" that shall apply to the group as a whole; the corresponding entries in the subordinate annotations shall be ignored. These entries are **Contents** (or **RC** and **DS**), **M**, **C**, **T**, **Popup**, **CreationDate**, **Subj**, and **Open**. Operations that manipulate any annotation in a group, such as movement, cut, and copy, shall be treated by conforming readers as acting on the entire group.

NOTE 3 A primary annotation may have replies that are not subordinate annotations; that is, that do not have an *RT* value of **Group**.

12.5.6.3 Annotation States

Beginning with PDF 1.5, annotations may have an author-specific *state* associated with them. The state is not specified in the annotation itself but in a separate text annotation that refers to the original annotation by means of its **IRT** ("in reply to") entry (see Table 173). States shall be grouped into a number of *state models*, as shown in Table 171.

Table 171 – Annotation states

State model	State	Description
Marked	Marked	The annotation has been marked by the user.
	Unmarked	The annotation has not been marked by the user (the default).
Review	Accepted	The user agrees with the change.
	Rejected	The user disagrees with the change.
	Cancelled	The change has been cancelled.
	Completed	The change has been completed.
	None	The user has indicated nothing about the change (the default).

Annotations shall be thought of as initially being in the default state for each state model. State changes made by a user shall be indicated in a text annotation with the following entries:

- The **T** entry (see Table 170) shall specify the user.
- The **IRT** entry (see Table 173) shall refer to the original annotation.
- **State** and **StateModel** (see Table 172) shall update the state of the original annotation for the specified user.

Additional state changes shall be made by adding text annotations in reply to the previous reply for a given user.

12.5.6.4 Text Annotations

A *text annotation* represents a “sticky note” attached to a point in the PDF document. When closed, the annotation shall appear as an icon; when open, it shall display a pop-up window containing the text of the note in a font and size chosen by the conforming reader. Text annotations shall not scale and rotate with the page; they shall behave as if the NoZoom and NoRotate annotation flags (see Table 165) were always set. Table 172 shows the annotation dictionary entries specific to this type of annotation.

Table 172 – Additional entries specific to a text annotation

Key	**Type**	**Value**
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Text** for a text annotation.
Open	boolean	*(Optional)* A flag specifying whether the annotation shall initially be displayed open. Default value: **false** (closed).
Name	name	*(Optional)* The name of an icon that shall be used in displaying the annotation. Conforming readers shall provide predefined icon appearances for at least the following standard names: Comment, Key, Note, Help, NewParagraph, Paragraph, Insert Additional names may be supported as well. Default value: Note. The annotation dictionary’s ***AP*** entry, if present, shall take precedence over the ***Name*** entry; see Table 168 and 12.5.5, “Appearance Streams.”
State	text string	*(Optional; PDF 1.5)* The state to which the original annotation shall be set; see 12.5.6.3, “Annotation States.” Default: “Unmarked” if **StateModel** is “Marked”; “None” if **StateModel** is “Review”.
StateModel	text string	*(Required if **State** is present, otherwise optional; PDF 1.5)* The state model corresponding to **State**; see 12.5.6.3, “Annotation States.”

EXAMPLE The following example shows the definition of a text annotation.

```
22 0 obj
    << /Type /Annot
        /Subtype /Text
        /Rect [266 116 430 204]
        /Contents (The quick brown fox ate the lazy mouse.)
    >>
endobj
```

12.5.6.5 Link Annotations

A *link annotation* represents either a hypertext link to a destination elsewhere in the document (see 12.3.2, “Destinations”) or an action to be performed (12.6, “Actions”). Table 173 shows the annotation dictionary entries specific to this type of annotation.

Table 173 – Additional entries specific to a link annotation

Key	**Type**	**Value**
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Link** for a link annotation.
A	dictionary	*(Optional; PDF 1.1)* An action that shall be performed when the link annotation is activated (see 12.6, “Actions”).

Table 173 – Additional entries specific to a link annotation (continued)

Key	Type	Value
Dest	array, name or byte string	*(Optional; not permitted if an **A** entry is present)* A destination that shall be displayed when the annotation is activated (see 12.3.2, “Destinations”).
H	name	*(Optional; PDF 1.2)* The annotation’s *highlighting mode*, the visual effect that shall be used when the mouse button is pressed or held down inside its active area: N (None) No highlighting. I (Invert) Invert the contents of the annotation rectangle. O I(Outline) Invert the annotation’s border. P (Push) Display the annotation as if it were being pushed below the surface of the page. Default value: I.
PA	dictionary	*(Optional; PDF 1.3)* A URI action (see 12.6.4.7, “URI Actions”) formerly associated with this annotation. When Web Capture (14.10, “ Web Capture”) changes an annotation from a URI to a go-to action (12.6.4.2, “Go-To Actions”), it uses this entry to save the data from the original URI action so that it can be changed back in case the target page for the go-to action is subsequently deleted.
QuadPoints	array	*(Optional; PDF 1.6)* An array of $8 \times n$ numbers specifying the coordinates of *n* quadrilaterals in default user space that comprise the region in which the link should be activated. The coordinates for each quadrilateral are given in the order $x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4$ specifying the four vertices of the quadrilateral in counterclockwise order. For orientation purposes, such as when applying an underline border style, the bottom of a quadrilateral is the line formed by (x_1, y_1) and (x_2, y_2). If this entry is not present or the conforming reader does not recognize it, the region specified by the **Rect** entry should be used. **QuadPoints** shall be ignored if any coordinate in the array lies outside the region specified by **Rect**.
BS	dictionary	(*Optional; PDF 1.6*) A border style dictionary (see Table 166) specifying the line width and dash pattern to be used in drawing the annotation’s border. The annotation dictionary’s **AP** entry, if present, takes precedence over the **BS** entry; see Table 164 and 12.5.5, “Appearance Streams”.

EXAMPLE The following example shows a link annotation that jumps to a destination elsewhere in the document.

```
93  0  obj
    <<  /Type  /Annot
        /Subtype  /Link
        /Rect  [71  717  190  734]
        /Border  [16  16  1]
        /Dest  [3 0 R  /FitR  –4  399  199  533]
    >>
endobj
```

12.5.6.6 Free Text Annotations

A *free text annotation (PDF 1.3)* displays text directly on the page. Unlike an ordinary text annotation (see 12.5.6.4, “Text Annotations”), a free text annotation has no open or closed state; instead of being displayed in a pop-up window, the text shall be always visible. Table 174 shows the annotation dictionary entries specific to this type of annotation. 12.7.3.3, “Variable Text” describes the process of using these entries to generate the appearance of the text in these annotations.

Table 174 – Additional entries specific to a free text annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **FreeText** for a free text annotation.
DA	string	*(Required)* The default appearance string that shall be used in formatting the text (see 12.7.3.3, “Variable Text”). The annotation dictionary’s ***AP*** entry, if present, shall take precedence over the ***DA*** entry; see Table 168 and 12.5.5, “Appearance Streams.”
Q	integer	*(Optional; PDF 1.4)* A code specifying the form of *quadding* (justification) that shall be used in displaying the annotation’s text: 0 Left-justified 1 Centered 2 Right-justified Default value: 0 (left-justified).
RC	text string or text stream	*(Optional; PDF 1.5)* A rich text string (see 12.7.3.4, “Rich Text Strings”) that shall be used to generate the appearance of the annotation.
DS	text string	*(Optional; PDF 1.5)* A default style string, as described in 12.7.3.4, “Rich Text Strings.”
CL	array	*(Optional; meaningful only if **IT** is **FreeTextCallout**; PDF 1.6)* An array of four or six numbers specifying a callout line attached to the free text annotation. Six numbers [x_1 y_1 x_2 y_2 x_3 y_3] represent the starting, knee point, and ending coordinates of the line in default user space, as shown in Figure 8.4. Four numbers [x_1 y_1 x_2 y_2] represent the starting and ending coordinates of the line.
IT	name	(*Optional; PDF 1.6*) A name describing the intent of the free text annotation (see also the **IT** entry in Table 170). The following values shall be valid: **FreeText** The annotation is intended to function as a plain free-text annotation. A plain free-text annotation is also known as a text box comment. **FreeTextCallout** The annotation is intended to function as a callout. The callout is associated with an area on the page through the callout line specified in **CL**. **FreeTextTypeWriter** The annotation is intended to function as a click-to-type or typewriter object and no callout line is drawn. Default value: **FreeText**
BE	dictionary	(*Optional; PDF 1.6*) A border effect dictionary (see Table 167) used in conjunction with the border style dictionary specified by the **BS** entry.

Table 174 – Additional entries specific to a free text annotation (continued)

Key	Type	Value
RD	rectangle	*(Optional; PDF 1.6)* A set of four numbers describing the numerical differences between two rectangles: the **Rect** entry of the annotation and a rectangle contained within that rectangle. The inner rectangle is where the annotation's text should be displayed. Any border styles and/or border effects specified by **BS** and **BE** entries, respectively, shall be applied to the border of the inner rectangle. The four numbers correspond to the differences in default user space between the left, top, right, and bottom coordinates of **Rect** and those of the inner rectangle, respectively. Each value shall be greater than or equal to 0. The sum of the top and bottom differences shall be less than the height of **Rect**, and the sum of the left and right differences shall be less than the width of **Rect**.
BS	dictionary	(*Optional; PDF 1.6*) A border style dictionary (see Table 166) specifying the line width and dash pattern that shall be used in drawing the annotation's border. The annotation dictionary's **AP** entry, if present, takes precedence over the **BS** entry; see Table 164 and 12.5.5, "Appearance Streams".
LE	name	(*Optional; meaningful only if **CL** is present; PDF 1.6*) A name specifying the line ending style that shall be used in drawing the callout line specified in **CL**. The name shall specify the line ending style for the endpoint defined by the pairs of coordinates (x_1, y_1). Table 176 shows the possible line ending styles. Default value: **None**.

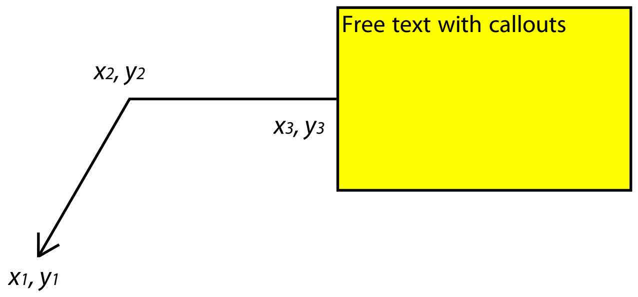


Figure 59 – Free text annotation with callout

12.5.6.7 Line Annotations

The purpose of a *line annotation (PDF 1.3)* is to display a single straight line on the page. When opened, it shall display a pop-up window containing the text of the associated note. Table 175 shows the annotation dictionary entries specific to this type of annotation.

Table 175 – Additional entries specific to a line annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Line** for a line annotation.
L	array	*(Required)* An array of four numbers, [x_1 y_1 x_2 y_2], specifying the starting and ending coordinates of the line in default user space. If the ***LL*** entry is present, this value shall represent the endpoints of the leader lines rather than the endpoints of the line itself; see Figure 60.
BS	dictionary	*(Optional)* A border style dictionary (see Table 166) specifying the width and dash pattern that shall be used in drawing the line. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***L*** and ***BS*** entries; see Table 168 and 12.5.5, "Appearance Streams."
LE	array	*(Optional; PDF 1.4)* An array of two names specifying the line ending styles that shall be used in drawing the line. The first and second elements of the array shall specify the line ending styles for the endpoints defined, respectively, by the first and second pairs of coordinates, (x_1, y_1) and (x_2, y_2), in the **L** array. Table 176 shows the possible values. Default value: [/None /None].
IC	array	*(Optional; PDF 1.4)* An array of numbers in the range 0.0 to 1.0 specifying the *interior color* that shall be used to fill the annotation's line endings (see Table 176). The number of array elements shall determine the colour space in which the colour is defined: 0 No colour; transparent 1 **DeviceGray** 3 **DeviceRGB** 4 **DeviceCMYK**
LL	number	*(Required if **LLE** is present, otherwise optional; PDF 1.6)* The length of *leader lines* in default user space that extend from each endpoint of the line perpendicular to the line itself, as shown in Figure 60. A positive value shall mean that the leader lines appear in the direction that is clockwise when traversing the line from its starting point to its ending point (as specified by **L**); a negative value shall indicate the opposite direction. Default value: 0 (no leader lines).
LLE	number	*(Optional; PDF 1.6)* A non-negative number that shall represents the length of *leader line extensions* that extend from the line proper 180 degrees from the leader lines, as shown in Figure 60. Default value: 0 (no leader line extensions).
Cap	boolean	*(Optional; PDF 1.6)* If **true**, the text specified by the **Contents** or **RC** entries shall be replicated as a caption in the appearance of the line, as shown in Figure 61 and Figure 62. The text shall be rendered in a manner appropriate to the content, taking into account factors such as writing direction. Default value: **false**.
IT	name	*(Optional; PDF 1.6)* A name describing the intent of the line annotation (see also Table 170). Valid values shall be **LineArrow**, which means that the annotation is intended to function as an arrow, and **LineDimension**, which means that the annotation is intended to function as a dimension line.

Table 175 – Additional entries specific to a line annotation (continued)

Key	Type	Value
LLO	number	(*Optional; PDF 1.7*) A non-negative number that shall represent the length of the leader line offset, which is the amount of empty space between the endpoints of the annotation and the beginning of the leader lines.
CP	name	(*Optional; meaningful only if* ***Cap*** *is* ***true****; PDF 1.7*) A name describing the annotation's caption positioning. Valid values are **Inline**, meaning the caption shall be centered inside the line, and **Top**, meaning the caption shall be on top of the line. Default value: **Inline**
Measure	dictionary	(*Optional; PDF 1.7*) A measure dictionary (see Table 261) that shall specify the scale and units that apply to the line annotation.
CO	array	(*Optional; meaningful only if* ***Cap*** *is* ***true****; PDF 1.7*) An array of two numbers that shall specify the offset of the caption text from its normal position. The first value shall be the horizontal offset along the annotation line from its midpoint, with a positive value indicating offset to the right and a negative value indicating offset to the left. The second value shall be the vertical offset perpendicular to the annotation line, with a positive value indicating a shift up and a negative value indicating a shift down. Default value: [0, 0] (no offset from normal positioning)

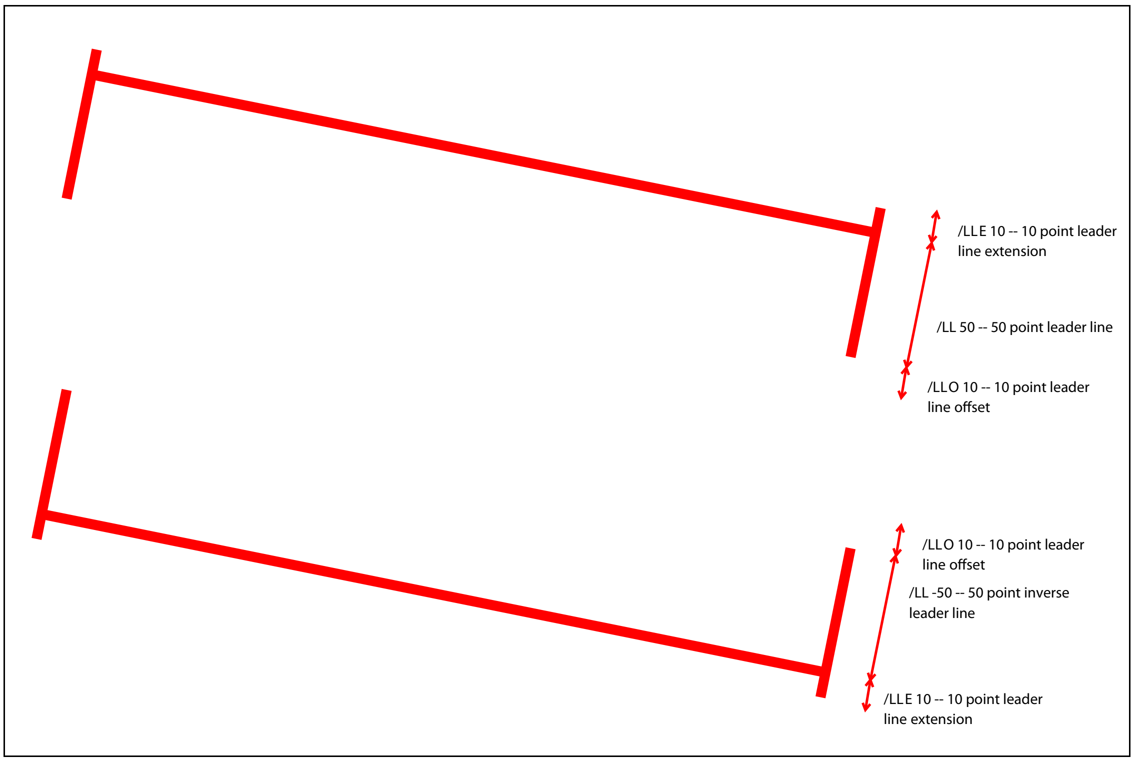

/LLE 10 -- 10 point leader line extension
/LL 50 -- 50 point leader line
/LLO 10 -- 10 point leader line offset
/LLO 10 -- 10 point leader line offset
/LL -50 -- 50 point inverse leader line
/LLE 10 -- 10 point leader line extension

Figure 60 – Leader lines

Figure 61 illustrates the effect of including a caption to a line annotation, which is specified by setting **Cap** to **true**.

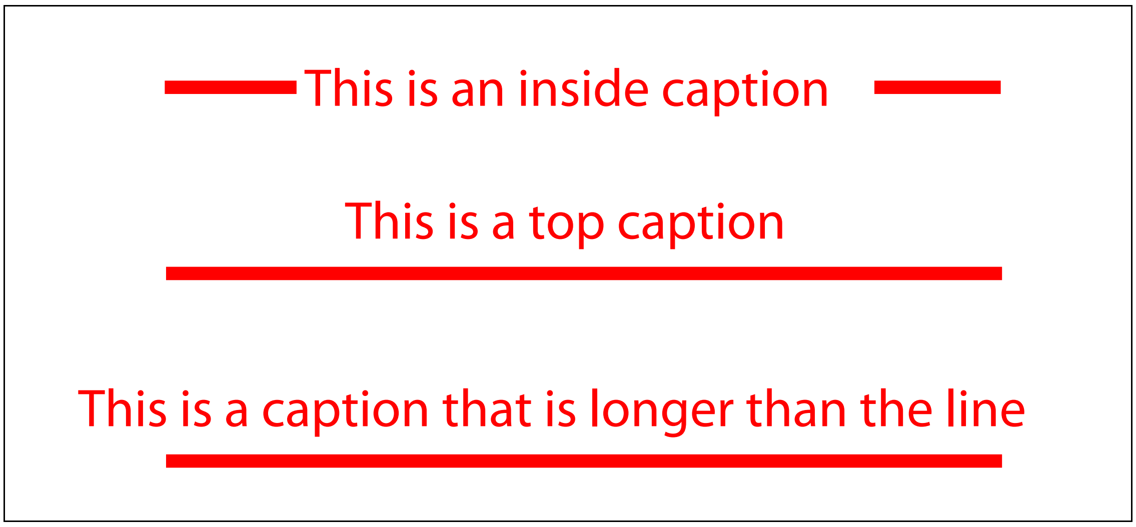

This is an inside caption
This is a top caption
This is a caption that is longer than the line

Figure 61 – Lines with captions appearing as part of the line

Figure 62 illustrates the effect of applying a caption to a line annotation that has a leader offset.

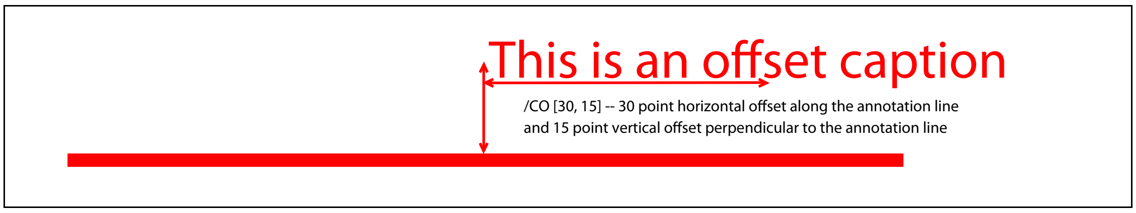

This is an offset caption
/CO [30, 15] -- 30 point horizontal offset along the annotation line and 15 point vertical offset perpendicular to the annotation line

Figure 62 – Line with a caption appearing as part of the offset

Table 176 – Line ending styles

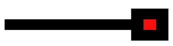
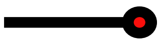
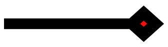
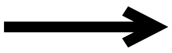
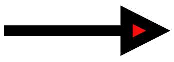
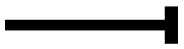
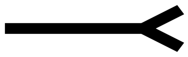
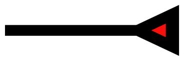
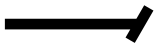
Name	Appearance	Description
Square		A square filled with the annotation’s interior color, if any
Circle		A circle filled with the annotation’s interior color, if any
Diamond		A diamond shape filled with the annotation’s interior color, if any
OpenArrow		Two short lines meeting in an acute angle to form an open arrowhead
ClosedArrow		Two short lines meeting in an acute angle as in the OpenArrow style and connected by a third line to form a triangular closed arrowhead filled with the annotation’s *interior color*, if any
None		No line ending

Table 176 – Line ending styles (continued)

Name	Appearance	Description
Butt		*(PDF 1.5)* A short line at the endpoint perpendicular to the line itself
ROpenArrow		*(PDF 1.5)* Two short lines in the reverse direction from OpenArrow
RClosedArrow		*(PDF 1.5)* A triangular closed arrowhead in the reverse direction from ClosedArrow
Slash		*(PDF 1.6)* A short line at the endpoint approximately 30 degrees clockwise from perpendicular to the line itself

12.5.6.8 Square and Circle Annotations

Square and *circle annotations (PDF 1.3)* shall display, respectively, a rectangle or an ellipse on the page. When opened, they shall display a pop-up window containing the text of the associated note. The rectangle or ellipse shall be inscribed within the annotation rectangle defined by the annotation dictionary's **Rect** entry (see Table 168).

Figure 63 shows two annotations, each with a border width of 18 points. Despite the names *square* and *circle*, the width and height of the annotation rectangle need not be equal. Table 177 shows the annotation dictionary entries specific to these types of annotations.

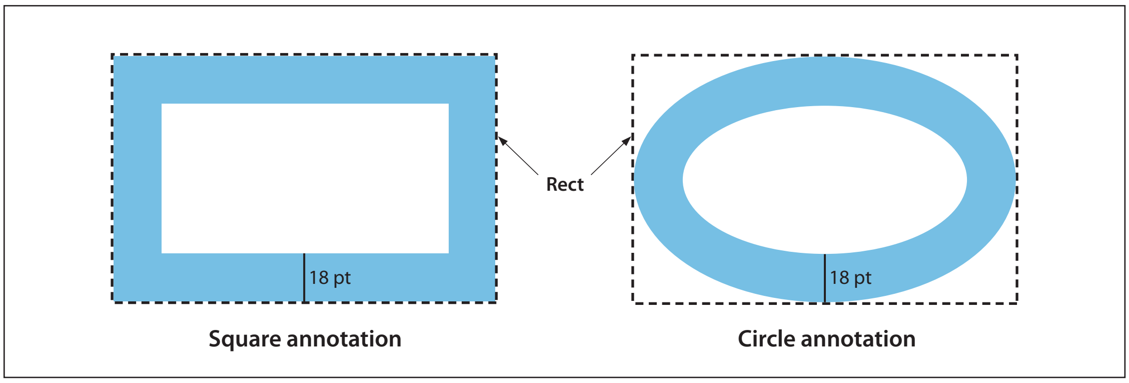


Figure 63 – Square and circle annotations

Table 177 – Additional entries specific to a square or circle annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Square** or **Circle** for a square or circle annotation, respectively.
BS	dictionary	*(Optional)* A border style dictionary (see Table 166) specifying the line width and dash pattern that shall be used in drawing the rectangle or ellipse. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***Rect*** and ***BS*** entries; see Table 168 and 12.5.5, "Appearance Streams."

Table 177 – Additional entries specific to a square or circle annotation (continued)

Key	Type	Value
IC	array	*(Optional; PDF 1.4)* An array of numbers that shall be in the range 0.0 to 1.0 and shall specify the *interior color* with which to fill the annotation's rectangle or ellipse. The number of array elements determines the colour space in which the colour shall be defined: 0 No colour; transparent 1 **DeviceGray** 3 **DeviceRGB** 4 **DeviceCMYK**
BE	dictionary	*(Optional; PDF 1.5)* A *border effect dictionary* describing an effect applied to the border described by the **BS** entry (see Table 167).
RD	rectangle	*(Optional; PDF 1.5)* A set of four numbers that shall describe the numerical differences between two rectangles: the **Rect** entry of the annotation and the actual boundaries of the underlying square or circle. Such a difference may occur in situations where a border effect (described by **BE**) causes the size of the **Rect** to increase beyond that of the square or circle. The four numbers shall correspond to the differences in default user space between the left, top, right, and bottom coordinates of **Rect** and those of the square or circle, respectively. Each value shall be greater than or equal to 0. The sum of the top and bottom differences shall be less than the height of **Rect**, and the sum of the left and right differences shall be less than the width of **Rect**.

12.5.6.9 Polygon and Polyline Annotations

Polygon annotations (PDF 1.5) display closed polygons on the page. Such polygons may have any number of vertices connected by straight lines. *Polyline annotations (PDF 1.5)* are similar to polygons, except that the first and last vertex are not implicitly connected.

Table 178 – Additional entries specific to a polygon or polyline annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Polygon** or **PolyLine** for a polygon or polyline annotation, respectively.
Vertices	array	*(Required)* An array of numbers (see Table 174) specifying the width and dash pattern that shall represent the alternating horizontal and vertical coordinates, respectively, of each vertex, in default user space.
LE	array	*(Optional; meaningful only for polyline annotations)* An array of two names that shall specify the line ending styles. The first and second elements of the array shall specify the line ending styles for the endpoints defined, respectively, by the first and last pairs of coordinates in the **Vertices** array. Table 176 shows the possible values. Default value: [/None /None].
BS	dictionary	*(Optional)* A border style dictionary (see Table 166) specifying the width and dash pattern that shall be used in drawing the line. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***Vertices*** and ***BS*** entries; see Table 168 and 12.5.5, "Appearance Streams."

Table 178 – Additional entries specific to a polygon or polyline annotation (continued)

Key	Type	Value
IC	array	*(Optional; PDF 1.4)* An array of numbers that shall be in the range 0.0 to 1.0 and shall specify the *interior color* with which to fill the annotation's line endings (see Table 176). The number of array elements determines the colour space in which the colour shall be defined: 0 No colour; transparent 1 **DeviceGray** 3 **DeviceRGB** 4 **DeviceCMYK**
BE	dictionary	*(Optional; meaningful only for polygon annotations)* A *border effect dictionary* that shall describe an effect applied to the border described by the **BS** entry (see Table 167).
IT	name	*(Optional; PDF 1.6)* A name that shall describe the intent of the polygon or polyline annotation (see also Table 170). The following values shall be valid: **PolygonCloud** The annotation is intended to function as a cloud object. **PolyLineDimension** (*PDF 1.7*) The polyline annotation is intended to function as a dimension. **PolygonDimension** (*PDF 1.7*) The polygon annotation is intended to function as a dimension.
Measure	dictionary	(*Optional; PDF 1.7*) A measure dictionary (see Table 261) that shall specify the scale and units that apply to the annotation.

12.5.6.10 Text Markup Annotations

Text markup annotations shall appear as highlights, underlines, strikeouts *(all PDF 1.3)*, or jagged ("squiggly") underlines *(PDF 1.4)* in the text of a document. When opened, they shall display a pop-up window containing the text of the associated note. Table 179 shows the annotation dictionary entries specific to these types of annotations.

Table 179 – Additional entries specific to text markup annotations

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Highlight**, **Underline**, **Squiggly**, or **StrikeOut** for a highlight, underline, squiggly-underline, or strikeout annotation, respectively.
QuadPoints	array	*(Required)* An array of 8 × *n* numbers specifying the coordinates of *n* quadrilaterals in default user space. Each quadrilateral shall encompasses a word or group of contiguous words in the text underlying the annotation. The coordinates for each quadrilateral shall be given in the order $x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4$ specifying the quadrilateral's four vertices in counterclockwise order (see Figure 64). The text shall be oriented with respect to the edge connecting points (x_1, y_1) and (x_2, y_2). The annotation dictionary's ***AP*** entry, if present, shall take precedence over ***QuadPoints***; see Table 168 and 12.5.5, "Appearance Streams."

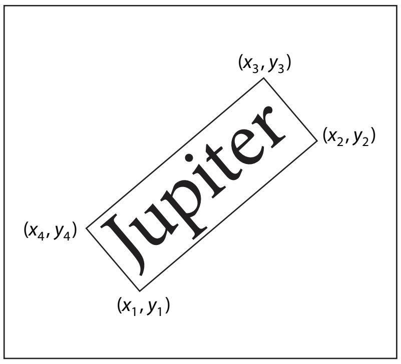


Figure 64 – QuadPoints specification

12.5.6.11 Caret Annotations

A caret annotation *(PDF 1.5)* is a visual symbol that indicates the presence of text edits. Table 180 lists the entries specific to caret annotations.

Table 180 – Additional entries specific to a caret annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Caret** for a caret annotation.
RD	rectangle	*(Optional; PDF 1.5)* A set of four numbers that shall describe the numerical differences between two rectangles: the **Rect** entry of the annotation and the actual boundaries of the underlying caret. Such a difference can occur. When a paragraph symbol specified by **Sy** is displayed along with the caret. The four numbers shall correspond to the differences in default user space between the left, top, right, and bottom coordinates of **Rect** and those of the caret, respectively. Each value shall be greater than or equal to 0. The sum of the top and bottom differences shall be less than the height of **Rect**, and the sum of the left and right differences shall be less than the width of **Rect**.
Sy	name	*(Optional)* A name specifying a symbol that shall be associated with the caret: P A new paragraph symbol (¶) should be associated with the caret. None No symbol should be associated with the caret. Default value: None.

12.5.6.12 Rubber Stamp Annotations

A *rubber stamp annotation (PDF 1.3)* displays text or graphics intended to look as if they were stamped on the page with a rubber stamp. When opened, it shall display a pop-up window containing the text of the associated note. Table 181 shows the annotation dictionary entries specific to this type of annotation.

Table 181 – Additional entries specific to a rubber stamp annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Stamp** for a rubber stamp annotation.
Name	name	*(Optional)* The name of an icon that shall be used in displaying the annotation. Conforming readers shall provide predefined icon appearances for at least the following standard names: Approved, Experimental, NotApproved, AsIs, Expired , NotForPublicRelease, Confidential, Final, Sold, Departmental, ForComment, TopSecret, Draft, ForPublicRelease Additional names may be supported as well. Default value: Draft. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***Name*** entry; see Table 168 and 12.5.5, "Appearance Streams."

12.5.6.13 Ink Annotations

An *ink annotation (PDF 1.3)* represents a freehand "scribble" composed of one or more disjoint paths. When opened, it shall display a pop-up window containing the text of the associated note. Table 182 shows the annotation dictionary entries specific to this type of annotation.

Table 182 – Additional entries specific to an ink annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Ink** for an ink annotation.
InkList	array	*(Required)* An array of *n* arrays, each representing a stroked path. Each array shall be a series of alternating horizontal and vertical coordinates in default user space, specifying points along the path. When drawn, the points shall be connected by straight lines or curves in an implementation-dependent way.
BS	dictionary	*(Optional)* A border style dictionary (see Table 166) specifying the line width and dash pattern that shall be used in drawing the paths. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***InkList*** and ***BS*** entries; see Table 168 and 12.5.5, "Appearance Streams."

12.5.6.14 Pop-up Annotations

A *pop-up annotation (PDF 1.3)* displays text in a pop-up window for entry and editing. It shall not appear alone but is associated with a markup annotation, its *parent annotation*, and shall be used for editing the parent's text. It shall have no appearance stream or associated actions of its own and shall be identified by the **Popup** entry in the parent's annotation dictionary (see Table 174). Table 183 shows the annotation dictionary entries specific to this type of annotation.

Table 183 – Additional entries specific to a pop-up annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Popup** for a pop-up annotation.
Parent	dictionary	*(Optional; shall be an indirect reference)* The parent annotation with which this pop-up annotation shall be associated. If this entry is present, the parent annotation's ***Contents***, ***M***, ***C***, and ***T*** entries (see Table 168) shall override those of the pop-up annotation itself.
Open	boolean	*(Optional)* A flag specifying whether the pop-up annotation shall initially be displayed open. Default value: **false** (closed).

12.5.6.15 File Attachment Annotations

A *file attachment annotation (PDF 1.3)* contains a reference to a file, which typically shall be embedded in the PDF file (see 7.11.4, "Embedded File Streams").

NOTE A table of data might use a file attachment annotation to link to a spreadsheet file based on that data; activating the annotation extracts the embedded file and gives the user an opportunity to view it or store it in the file system. Table 184 shows the annotation dictionary entries specific to this type of annotation.

The **Contents** entry of the annotation dictionary may specify descriptive text relating to the attached file. Conforming readers shall use this entry rather than the optional **Desc** entry *(PDF 1.6)* in the file specification dictionary (see Table 44) identified by the annotation's **FS** entry.

Table 184 – Additional entries specific to a file attachment annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **FileAttachment** for a file attachment annotation.
FS	file specification	*(Required)* The file associated with this annotation.
Name	name	*(Optional)* The name of an icon that shall be used in displaying the annotation. Conforming readers shall provide predefined icon appearances for at least the following standard names: GraphPushPin PaperclipTag Additional names may be supported as well. Default value: PushPin. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***Name*** entry; see Table 168 and 12.5.5, "Appearance Streams."

12.5.6.16 Sound Annotations

A *sound annotation (PDF 1.2)* shall analogous to a text annotation except that instead of a text note, it contains sound recorded from the computer's microphone or imported from a file. When the annotation is activated, the sound shall be played. The annotation shall behave like a text annotation in most ways, with a different icon (by default, a speaker) to indicate that it represents a sound. Table 185 shows the annotation dictionary entries specific to this type of annotation. Sound objects are discussed in 13.3, "Sounds."

Table 185 – Additional entries specific to a sound annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Sound** for a sound annotation.
Sound	stream	*(Required)* A sound object defining the sound that shall be played when the annotation is activated (see 13.3, “Sounds”).
Name	name	*(Optional)* The name of an icon that shall be used in displaying the annotation. Conforming readers shall provide predefined icon appearances for at least the standard names Speaker and Mic. Additional names may be supported as well. Default value: Speaker. The annotation dictionary’s ***AP*** entry, if present, shall take precedence over the ***Name*** entry; see Table 168 and 12.5.5, “Appearance Streams.”

12.5.6.17 Movie Annotations

A *movie annotation (PDF 1.2)* contains animated graphics and sound to be presented on the computer screen and through the speakers. When the annotation is activated, the movie shall be played. Table 186 shows the annotation dictionary entries specific to this type of annotation. Movies are discussed in 13.4, “Movies.”

Table 186 – Additional entries specific to a movie annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Movie** for a movie annotation.
T	text string	(*Optional*) The title of the movie annotation. Movie actions (12.6.4.9, “Movie Actions”) may use this title to reference the movie annotation.
Movie	dictionary	*(Required)* A movie dictionary that shall describe the movie’s static characteristics (see 13.4, “Movies”).
A	boolean or dictionary	*(Optional)* A flag or dictionary specifying whether and how to play the movie when the annotation is activated. If this value is a dictionary, it shall be a movie activation dictionary (see 13.4, “Movies”) specifying how to play the movie. If the value is the boolean **true**, the movie shall be played using default activation parameters. If the value is **false***,* the movie shall not be played. Default value: **true**.

12.5.6.18 Screen Annotations

A *screen annotation (PDF 1.5)* specifies a region of a page upon which media clips may be played. It also serves as an object from which actions can be triggered. 12.6.4.13, “Rendition Actions” discusses the relationship between screen annotations and rendition actions. Table 187 shows the annotation dictionary entries specific to this type of annotation.

Table 187 – Additional entries specific to a screen annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Screen** for a screen annotation.
T	text string	(*Optional*) The title of the screen annotation.
MK	dictionary	*(Optional)* An appearance characteristics dictionary (see Table 189). The **I** entry of this dictionary provides the icon used in generating the appearance referred to by the screen annotation’s **AP** entry.

Table 187 – Additional entries specific to a screen annotation (continued)

Key	Type	Value
A	dictionary	*(Optional; PDF 1.1)* An action that shall be performed when the annotation is activated (see 12.6, “Actions”).
AA	dictionary	*(Optional; PDF 1.2)* An additional-actions dictionary defining the screen annotation’s behaviour in response to various trigger events (see 12.6.3, “Trigger Events”).

In addition to the entries in Table 187, screen annotations may use the common entries in the annotation dictionary (see Table 164) in the following ways:

- The **P** entry shall be used for a screen annotation referenced by a rendition action. It shall reference a valid page object, and the annotation shall be present in the page’s **Annots** array for the action to be valid.
- The **AP** entry refers to an appearance dictionary (see Table 168) whose normal appearance provides the visual appearance for a screen annotation that shall be used for printing and default display when a media clip is not being played. If **AP** is not present, the screen annotation shall not have a default visual appearance and shall not be printed.

12.5.6.19 Widget Annotations

Interactive forms (see 12.7, “Interactive Forms”) use *widget annotations (PDF 1.2)* to represent the appearance of fields and to manage user interactions. As a convenience, when a field has only a single associated widget annotation, the contents of the field dictionary (12.7.3, “Field Dictionaries”) and the annotation dictionary may be merged into a single dictionary containing entries that pertain to both a field and an annotation.

NOTE This presents no ambiguity, since the contents of the two kinds of dictionaries do not conflict.

Table 188 shows the annotation dictionary entries specific to this type of annotation; interactive forms and fields are discussed at length in 12.7, “Interactive Forms.”

Table 188 – Additional entries specific to a widget annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Widget** for a widget annotation.
H	name	*(Optional)* The annotation’s *highlighting mode*, the visual effect that shall be used when the mouse button is pressed or held down inside its active area: N (None) No highlighting. I (Invert) Invert the contents of the annotation rectangle. O (Outline) Invert the annotation’s border. P (Push) Display the annotation’s down appearance, if any (see 12.5.5, “Appearance Streams”). If no down appearance is defined, the contents of the annotation rectangle shall be offset to appear as if it were being pushed below the surface of the page. T (Toggle) Same as P (which is preferred). A highlighting mode other than P shall override any down appearance defined for the annotation. Default value: I.
MK	dictionary	*(Optional)* An appearance characteristics dictionary (see Table 189) that shall be used in constructing a dynamic appearance stream specifying the annotation’s visual presentation on the page. The name ***MK*** for this entry is of historical significance only and has no direct meaning.

Table 188 – Additional entries specific to a widget annotation (continued)

Key	Type	Value
A	dictionary	*(Optional; PDF 1.1)* An action that shall be performed when the annotation is activated (see 12.6, "Actions").
AA	dictionary	*(Optional; PDF 1.2)* An additional-actions dictionary defining the annotation's behaviour in response to various trigger events (see 12.6.3, "Trigger Events").
BS	dictionary	*(Optional; PDF 1.2)* A border style dictionary (see Table 166) specifying the width and dash pattern that shall be used in drawing the annotation's border. The annotation dictionary's ***AP*** entry, if present, shall take precedence over the ***L*** and ***BS*** entries; see Table 168 and 12.5.5, "Appearance Streams."
Parent	dictionary	(*Required if this widget annotation is one of multiple children in a field; absent otherwise*) An indirect reference to the widget annotation's parent field. A widget annotation may have at most one parent; that is, it can be included in the **Kids** array of at most one field

The **MK** entry may be used to provide an *appearance characteristics dictionary* containing additional information for constructing the annotation's appearance stream. Table 189 shows the contents of this dictionary.

Table 189 – Entries in an appearance characteristics dictionary

Key	Type	Value
R	integer	*(Optional)* The number of degrees by which the widget annotation shall be rotated counterclockwise relative to the page. The value shall be a multiple of 90. Default value: 0.
BC	array	*(Optional)* An array of numbers that shall be in the range 0.0 to 1.0 specifying the colour of the widget annotation's border. The number of array elements determines the colour space in which the colour shall be defined: 0 No colour; transparent 1 **DeviceGray** 3 **DeviceRGB** 4 **DeviceCMYK**
BG	array	*(Optional)* An array of numbers that shall be in the range 0.0 to 1.0 specifying the colour of the widget annotation's background. The number of array elements shall determine the colour space, as described for **BC**.
CA	text string	*(Optional; button fields only)* The widget annotation's *normal caption*, which shall be displayed when it is not interacting with the user. Unlike the remaining entries listed in this Table, which apply only to widget annotations associated with pushbutton fields (see Pushbuttons in 12.7.4.2, "Button Fields"), the ***CA*** entry may be used with any type of button field, including check boxes (see Check Boxes in 12.7.4.2, "Button Fields") and radio buttons (Radio Buttons in 12.7.4.2, "Button Fields").
RC	text string	*(Optional; pushbutton fields only)* The widget annotation's *rollover caption*, which shall be displayed when the user rolls the cursor into its active area without pressing the mouse button.
AC	text string	*(Optional; pushbutton fields only)* The widget annotation's *alternate (down) caption*, which shall be displayed when the mouse button is pressed within its active area.

Table 189 – Entries in an appearance characteristics dictionary (continued)

Key	Type	Value
I	stream	*(Optional; pushbutton fields only; shall be an indirect reference)* A form XObject defining the widget annotation's *normal icon*, which shall be displayed when it is not interacting with the user.
RI	stream	*(Optional; pushbutton fields only; shall be an indirect reference)* A form XObject defining the widget annotation's *rollover icon*, which shall be displayed when the user rolls the cursor into its active area without pressing the mouse button.
IX	stream	*(Optional; pushbutton fields only; shall be an indirect reference)* A form XObject defining the widget annotation's *alternate (down) icon*, which shall be displayed when the mouse button is pressed within its active area.
IF	dictionary	*(Optional; pushbutton fields only)* An icon fit dictionary (see Table 247) specifying how the widget annotation's icon shall be displayed within its annotation rectangle. If present, the icon fit dictionary shall apply to all of the annotation's icons (normal, rollover, and alternate).
TP	integer	*(Optional; pushbutton fields only)* A code indicating where to position the text of the widget annotation's caption relative to its icon: 0 No icon; caption only 1 No caption; icon only 2 Caption below the icon 3 Caption above the icon 4 Caption to the right of the icon 5 Caption to the left of the icon 6 Caption overlaid directly on the icon Default value: 0.

12.5.6.20 Printer's Mark Annotations

A *printer's mark annotation (PDF 1.4)* represents a graphic symbol, such as a registration target, colour bar, or cut mark, that may be added to a page to assist production personnel in identifying components of a multiple-plate job and maintaining consistent output during production. See 14.11.3, "Printer's Marks," for further discussion.

12.5.6.21 Trap Network Annotations

A *trap network annotation (PDF 1.3)* may be used to define the trapping characteristics for a page of a PDF document.

NOTE *Trapping* is the process of adding marks to a page along colour boundaries to avoid unwanted visual artifacts resulting from misregistration of colorants when the page is printed.

A page shall have no more than one trap network annotation, whose **Subtype** entry has the value **TrapNet** and which shall always be the last element in the page object's **Annots** array (see 7.7.3.3, "Page Objects"). See 14.11.6, "Trapping Support," for further discussion.

12.5.6.22 Watermark Annotations

A *watermark annotation (PDF 1.6)* shall be used to represent graphics that shall be printed at a fixed size and position on a page, regardless of the dimensions of the printed page. The FixedPrint entry of a watermark annotation dictionary (see Table 190) shall be a dictionary that contains values for specifying the size and position of the annotation (see Table 191).

Watermark annotations shall have no pop-up window or other interactive elements. When displaying a watermark annotation on-screen, conforming readers shall use the dimensions of the media box as the page size so that the scroll and zoom behaviour is the same as for other annotations.

NOTE Since many printing devices have non printable margins, such margins should be taken into consideration when positioning watermark annotations near the edge of a page.

Table 190 – Additional entries specific to a watermark annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **Watermark** for a watermark annotation.
FixedPrint	dictionary	*(Optional)* A *fixed print dictionary* (see Table 191) that specifies how this annotation shall be drawn relative to the dimensions of the target media. If this entry is not present, the annotation shall be drawn without any special consideration for the dimensions of the target media. If the dimensions of the target media are not known at the time of drawing, drawing shall be done relative to the dimensions specified by the page's ***MediaBox*** entry (see Table 30).

Table 191 – Entries in a fixed print dictionary

Key	Type	Value
Type	name	*(Required)* Shall be **FixedPrint**.
Matrix	array	*(Optional)* The matrix used to transform the annotation's rectangle before rendering. Default value: the identity matrix [1 0 0 1 0 0]. When positioning content near the edge of a page, this entry should be used to provide a reasonable offset to allow for nonburnable margins.
H	number	*(Optional)* The amount to translate the associated content horizontally, as a percentage of the width of the target media (or if unknown, the width of the page's **MediaBox**). 1.0 represents 100% and 0.0 represents 0%. Negative values should not be used, since they may cause content to be drawn off the page. Default value: 0.
V	number	*(Optional)* The amount to translate the associated content vertically, as a percentage of the height of the target media (or if unknown, the height of the page's **MediaBox**). 1.0 represents 100% and 0.0 represents 0%. Negative values should not be used, since they may cause content to be drawn off the page. Default value: 0.

When rendering a watermark annotation with a **FixedPrint** entry, the following behaviour shall occur:

- The annotation's rectangle (as specified by its **Rect** entry) shall be translated to the origin and transformed by the **Matrix** entry of its **FixedPrint** dictionary to produce a quadrilateral with arbitrary orientation.
- The *transformed annotation rectangle* shall be defined as the smallest upright rectangle that encompasses this quadrilateral; it shall be used in place of the annotation rectangle referred to in steps 2 and 3 of "Algorithm: Appearance streams".

In addition, given a matrix *B* that maps a scaled and rotated page into the default user space, a new matrix shall be computed that cancels out *B* and translates the origin of the printed page to the origin of the default user space. This transformation shall be applied to ensure the correct scaling and alignment.

EXAMPLE The following example shows a watermark annotation that prints a text string one inch from the left and one inch from the top of the printed page.

```
8 0 obj                          % Watermark appearance
    <<
        /Length ...
        /Subtype /Form
        /Resources ...
        /BBox ...
    >>
    stream
    ...
        BT
        /F1 1 Tf
        36 0 0 36 0 -36 Tm
        (Do Not Build) Tx
        ET
        ...
    endstream
endobj

9 0 obj                          % Watermark annotation
    <<
        /Rect ...
        /Type /Annot
        /Subtype /Watermark
        /FixedPrint 10 0 R
        /AP <</N 8 0 R>>
    >>

% in the page dictionary
    /Annots [9 0 R]

10 0 obj                         % Fixed print dictionary
    <<
        /Type /FixedPrint
        /Matrix [1 0 0 1 72 -72]      % Translate one inch right and one inch down
        /H 0
        /V 1.0                        % Translate the full height of the page vertically
    >>
endobj
```

In situations other than the usual case where the PDF page size equals the printed page size, watermark annotations with a **FixedPrint** entry shall be printed in the following manner:

- When page tiling is selected in a conforming reader (that is, a single PDF page is printed on multiple pages), the annotations shall be printed at the specified size and position on each page to ensure that any enclosed content is present and legible on each printed page.

- When *n*-up printing is selected (that is, multiple PDF pages are printed on a single page), the annotations shall be printed at the specified size and shall be positioned as if the dimensions of the printed page were limited to a single portion of the page. This ensures that any enclosed content does not overlap content from other pages, thus rendering it illegible.

12.5.6.23 Redaction Annotations

A redaction annotation (*PDF 1.7*) identifies content that is intended to be removed from the document. The intent of redaction annotations is to enable the following process:

a) *Content identification*. A user applies redact annotations that specify the pieces or regions of content that should be removed. Up until the next step is performed, the user can see, move and redefine these annotations.

b) *Content removal*. The user instructs the viewer application to apply the redact annotations, after which the content in the area specified by the redact annotations is removed. In the removed content's place, some marking appears to indicate the area has been redacted. Also, the redact annotations are removed from the PDF document.

Redaction annotations provide a mechanism for the first step in the redaction process (content identification). This allows content to be marked for redaction in a non-destructive way, thus enabling a review process for evaluating potential redactions prior to removing the specified content.

Redaction annotations shall provide enough information to be used in the second phase of the redaction process (content removal). This phase is application-specific and requires the conforming reader to remove all content identified by the redaction annotation, as well as the annotation itself.

Conforming readers that support redaction annotations shall provide a mechanism for applying content removal, and they shall remove all traces of the specified content. If a portion of an image is contained in a redaction region, that portion of the image data shall be destroyed; clipping or image masks shall not be used to hide that data. Such conforming readers shall also be diligent in their consideration of all content that can exist in a PDF document, including XML Forms Architecture (XFA) content and Extensible Metadata Platform (XMP) content.

Table 192 – Additional entries specific to a redaction annotation

Key	Type	Value
Subtype	name	(*Required*) The type of annotation that this dictionary describes; shall be **Redact** for a redaction annotation.
QuadPoints	array	(*Optional*) An array of 8 x n numbers specifying the coordinates of n quadrilaterals in default user space, as described in Table 175 for text markup annotations. If present, these quadrilaterals denote the content region that is intended to be removed. If this entry is not present, the **Rect** entry denotes the content region that is intended to be removed.
IC	array	(*Optional*) An array of three numbers in the range 0.0 to 1.0 specifying the components, in the **DeviceRGB** colour space, of the interior colour with which to fill the redacted region after the affected content has been removed. If this entry is absent, the interior of the redaction region is left transparent. This entry is ignored if the **RO** entry is present.
RO	stream	(*Optional*) A form XObject specifying the overlay appearance for this redaction annotation. After this redaction is applied and the affected content has been removed, the overlay appearance should be drawn such that its origin lines up with the lower-left corner of the annotation rectangle. This form XObject is not necessarily related to other annotation appearances, and may or may not be present in the **AP** dictionary. This entry takes precedence over the **IC**, **OverlayText**, **DA**, and **Q** entries.
OverlayText	text string	(*Optional*) A text string specifying the overlay text that should be drawn over the redacted region after the affected content has been removed. This entry is ignored if the **RO** entry is present.
Repeat	boolean	(*Optional*) If **true**, then the text specified by **OverlayText** should be repeated to fill the redacted region after the affected content has been removed. This entry is ignored if the **RO** entry is present. Default value: **false**.
DA	byte string	(*Required if **OverlayText** is present, ignored otherwise*) The appearance string to be used in formatting the overlay text when it is drawn after the affected content has been removed (see 12.7.3.3, “Variable Text”). This entry is ignored if the **RO** entry is present.

Table 192 – Additional entries specific to a redaction annotation (continued)

Key	Type	Value
Q	integer	(*Optional*) A code specifying the form of quadding (justification) to be used in laying out the overlay text: 0 Left-justified 1 Centered 2 Right-justified This entry is ignored if the **RO** entry is present. Default value: 0 (left-justified).

12.6 Actions

12.6.1 General

In addition to jumping to a destination in the document, an annotation or outline item may specify an *action (PDF 1.1)* to perform, such as launching an application, playing a sound, changing an annotation's appearance state. The optional **A** entry in the annotation or outline item dictionary (see Tables 168 and 153) specifies an action performed when the annotation or outline item is activated; in PDF 1.2, a variety of other circumstances may trigger an action as well (see 12.6.3, "Trigger Events"). In addition, the optional OpenAction entry in a document's catalogue (7.7.2, "Document Catalog") may specify an action that shall be performed when the document is opened. PDF includes a wide variety of standard action types, described in detail in 12.6.4, "Action Types."

12.6.2 Action Dictionaries

An *action dictionary* defines the characteristics and behaviour of an action. Table 193 shows the required and optional entries that are common to all action dictionaries. The dictionary may contain additional entries specific to a particular action type; see the descriptions of individual action types in 12.6.4, "Action Types," for details.

Table 193 – Entries common to all action dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Action** for an action dictionary.
S	name	*(Required)* The type of action that this dictionary describes; see Table 194 for specific values.
Next	dictionary or array	*(Optional; PDF 1.2)* The next action or sequence of actions that shall be performed after the action represented by this dictionary. The value is either a single action dictionary or an array of action dictionaries that shall be performed in order; see the Note for further discussion.

NOTE 1 The action dictionary's **Next** entry *(PDF 1.2)* allows sequences of actions to be chained together. For example, the effect of clicking a link annotation with the mouse might be to play a sound, jump to a new page, and start up a movie. Note that the **Next** entry is not restricted to a single action but may contain an array of actions, each of which in turn may have a **Next** entry of its own. The actions may thus form a tree instead of a simple linked list. Actions within each **Next** array are executed in order, each followed in turn by any actions specified in *its* **Next** entry, and so on recursively. Conforming readers should attempt to provide reasonable behavior in anomalous situations. For example, self-referential actions should not be executed more than once, and actions that close the document or otherwise render the next action impossible should terminate the execution sequence. Applications should also provide some mechanism for the user to interrupt and manually terminate a sequence of actions.

PDF 1.5 introduces transition actions, which allow the control of drawing during a sequence of actions; see 12.6.4.14, "Transition Actions."

NOTE 2 No action should modify its own action dictionary or any other in the action tree in which it resides. The effect of such modification on subsequent execution of actions in the tree is undefined.

12.6.3 Trigger Events

An annotation, page object, or (beginning with PDF 1.3) interactive form field may include an entry named AA that specifies an *additional-actions dictionary (PDF 1.2)* that extends the set of events that can trigger the execution of an action. In PDF 1.4, the document catalogue dictionary (see 7.7.2, "Document Catalog") may also contain an **AA** entry for trigger events affecting the document as a whole. Tables 194 to 197 show the contents of this type of dictionary.

PDF 1.5 introduces four trigger events in annotation's additional-actions dictionary to support multimedia presentations:

- The PO and **PC** entries have a similar function to the **O** and **C** entries in the page object's additional-actions dictionary (see Table 194). However, associating these triggers with annotations allows annotation objects to be self-contained.

EXAMPLE Annotations containing such actions can be copied or moved between pages without requiring page open/close actions to be changed.

- The **PV** and **PI** entries allow a distinction between pages that are open and pages that are visible. At any one time, while more than one page may be visible, depending on the page layout.

NOTE 1 For these trigger events, the values of the flags specified by the annotation's **F** entry (see 12.5.3, "Annotation Flags") have no bearing on whether a given trigger event occurs.

Table 194 – Entries in an annotation's additional-actions dictionary

Key	Type	Value
E	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the cursor enters the annotation's active area.
X	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the cursor exits the annotation's active area.
D	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the mouse button is pressed inside the annotation's active area.
U	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the mouse button is released inside the annotation's active area. For backward compatibility, the ***A*** entry in an annotation dictionary, if present, takes precedence over this entry (see Table 168).
Fo	dictionary	*(Optional; PDF 1.2; widget annotations only)* An action that shall be performed when the annotation receives the input focus.
Bl	dictionary	*(Optional; PDF 1.2; widget annotations only)* (Uppercase B, lowercase L) An action that shall be performed when the annotation loses the input focus.
PO	dictionary	*(Optional; PDF 1.5)* An action that shall be performed when the page containing the annotation is opened. EXAMPLE 1 When the user navigates to it from the next or previous page or by means of a link annotation or outline item. The action shall be executed after the **O** action in the page's additional-actions dictionary (see Table 195) and the **OpenAction** entry in the document **Catalog** (see Table 28), if such actions are present.

Table 194 – Entries in an annotation's additional-actions dictionary (continued)

Key	Type	Value
PC	dictionary	*(Optional; PDF 1.5)* An action that shall be performed when the page containing the annotation is closed. EXAMPLE 2 When the user navigates to the next or previous page, or follows a link annotation or outline item. The action shall be executed before the **C** action in the page's additional-actions dictionary (see Table 195), if present.
PV	dictionary	*(Optional; PDF 1.5)* An action that shall be performed when the page containing the annotation becomes visible.
PI	dictionary	*(Optional; PDF 1.5)* An action that shall be performed when the page containing the annotation is no longer visible in the conforming reader's user interface.

Table 195 – Entries in a page object's additional-actions dictionary

Key	Type	Value
O	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the page is opened (for example, when the user navigates to it from the next or previous page or by means of a link annotation or outline item). This action is independent of any that may be defined by the **OpenAction** entry in the document **Catalog** (see 7.7.2, "Document Catalog") and shall be executed after such an action.
C	dictionary	*(Optional; PDF 1.2)* An action that shall be performed when the page is closed (for example, when the user navigates to the next or previous page or follows a link annotation or an outline item). This action applies to the page being closed and shall be executed before any other page is opened.

Table 196 – Entries in a form field's additional-actions dictionary

Key	Type	Value
K	dictionary	*(Optional; PDF 1.3)* A JavaScript action that shall be performed when the user modifies a character in a text field or combo box or modifies the selection in a scrollable list box. This action may check the added text for validity and reject or modify it.
F	dictionary	*(Optional; PDF 1.3)* A JavaScript action that shall be performed before the field is formatted to display its value. This action may modify the field's value before formatting.
V	dictionary	*(Optional; PDF 1.3)* A JavaScript action that shall be performed when the field's value is changed. This action may check the new value for validity. (The name V stands for "validate.")
C	dictionary	*(Optional; PDF 1.3)* A JavaScript action that shall be performed to recalculate the value of this field when that of another field changes. (The name **C** stands for "calculate.") The order in which the document's fields are recalculated shall be defined by the **CO** entry in the interactive form dictionary (see 12.7.2, "Interactive Form Dictionary").

Table 197 – Entries in the document catalog's additional-actions dictionary

Key	Type	Value
WC	dictionary	*(Optional; PDF 1.4)* A JavaScript action that shall be performed before closing a document. (The name **WC** stands for "will close.")
WS	dictionary	*(Optional; PDF 1.4)* A JavaScript action that shall be performed before saving a document. (The name **WS** stands for "will save.")
DS	dictionary	*(Optional; PDF 1.4)* A JavaScript action that shall be performed after saving a document. (The name **DS** stands for "did save.")
WP	dictionary	*(Optional; PDF 1.4)* A JavaScript action that shall be performed before printing a document. (The name **WP** stands for "will print.")
DP	dictionary	*(Optional; PDF 1.4)* A JavaScript action that shall be performed after printing a document. (The name **DP** stands for "did print.")

Conforming readers shall ensure the presence of such a device, or equivalent controls for simulating one, for the corresponding actions to be executed correctly. Mouse-related trigger events are subject to the following constraints:

- An **E** (enter) event may occur only when the mouse button is up.
- An **X** (exit) event may not occur without a preceding **E** event.
- A **U** (up) event may not occur without a preceding **E** and **D** event.
- In the case of overlapping or nested annotations, entering a second annotation's active area causes an **X** event to occur for the first annotation.

NOTE 2 The field-related trigger events ***K*** (keystroke), ***F*** (format), ***V*** (validate), and ***C*** (calculate) are not defined for button fields (see 12.7.4.2, "Button Fields"). The effects of an action triggered by one of these events are limited only by the action itself and can occur outside the described scope of the event. For example, even though the ***F*** event is used to trigger actions that format field values prior to display, it is possible for an action triggered by this event to perform a calculation or make any other modification to the document.

These field-related trigger events can occur either through user interaction or programmatically, such as in response to the **NeedAppearances** entry in the interactive form dictionary (see 12.7.2, "Interactive Form Dictionary"), importation of FDF data (12.7.7, "Forms Data Format"), or JavaScript actions (12.6.4.16, "JavaScript Actions"). For example, the user's modifying a field value can trigger a cascade of calculations and further formatting and validation for other fields in the document.

12.6.4 Action Types

12.6.4.1 General

PDF supports the standard action types listed in Table 198. The following sub-clauses describe each of these types in detail.

Table 198 – Action types

Action type	Description	Discussed in sub-clause
GoTo	Go to a destination in the current document.	12.6.4.2, "Go-To Actions"
GoToR	("Go-to remote") Go to a destination in another document.	12.6.4.3, "Remote Go-To Actions"
GoToE	("Go-to embedded"; *PDF 1.6*) Go to a destination in an embedded file.	12.6.4.4, "Embedded Go-To Actions"

Table 198 – Action types (continued)

Action type	Description	Discussed in sub-clause
Launch	Launch an application, usually to open a file.	12.6.4.5, “Launch Actions”
Thread	Begin reading an article thread.	12.6.4.6, “Thread Actions”
URI	Resolve a uniform resource identifier.	12.6.4.7, “URI Actions”
Sound	*(PDF 1.2)* Play a sound.	12.6.4.8, “Sound Actions”
Movie	*(PDF 1.2)* Play a movie.	12.6.4.9, “Movie Actions”
Hide	*(PDF 1.2)* Set an annotation’s Hidden flag.	12.6.4.10, “Hide Actions”
Named	*(PDF 1.2)* Execute an action predefined by the conforming reader.	12.6.4.11, “Named Actions”
SubmitForm	*(PDF 1.2)* Send data to a uniform resource locator.	12.7.5.2, “Submit-Form Action”
ResetForm	*(PDF 1.2)* Set fields to their default values.	12.7.5.3, “Reset-Form Action”
ImportData	*(PDF 1.2)* Import field values from a file.	12.7.5.4, “Import-Data Action”
JavaScript	*(PDF 1.3)* Execute a JavaScript script.	12.6.4.16, “JavaScript Actions”
SetOCGState	*(PDF 1.5)* Set the states of optional content groups.	12.6.4.12, “Set-OCG-State Actions”
Rendition	*(PDF 1.5)* Controls the playing of multimedia content.	12.6.4.13, “Rendition Actions”
Trans	*(PDF 1.5)* Updates the display of a document, using a transition dictionary.	12.6.4.14, “Transition Actions”
GoTo3DView	*(PDF 1.6)* Set the current view of a 3D annotation	12.6.4.15, “Go-To-3D-View Actions,”

NOTE The set-state action is considered obsolete and should not be used.

12.6.4.2 Go-To Actions

A *go-to action* changes the view to a specified destination (page, location, and magnification factor). Table 199 shows the action dictionary entries specific to this type of action.

Table 199 – Additional entries specific to a go-to action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **GoTo** for a go-to action.
D	name, byte string, or array	*(Required)* The destination to jump to (see 12.3.2, “Destinations”).

NOTE Specifying a go-to action in the **A** entry of a link annotation or outline item (see Table 173 and Table 153) has the same effect as specifying the destination directly with the **Dest** entry. For example, the link annotation shown in the Example in 12.6.4.12, “Set-OCG-State Actions,” which uses a go-to action, has the same effect as the one in the following Example, which specifies the destination directly. However, the go-to action is less compact and is not compatible with PDF 1.0; therefore, using a direct destination is preferable.

EXAMPLE
```
93 0 obj
   << /Type /Annot
      /Subtype /Link
      /Rect [71 717 190 734]
      /Border [16 16 1]
      /A << /Type /Action
           /S /GoTo
           /D [3 0 R /FitR –4 399 199 533]
         >>
   >>
endobj
```

12.6.4.3 Remote Go-To Actions

A *remote go-to action* is similar to an ordinary go-to action but jumps to a destination in another PDF file instead of the current file. Table 200 shows the action dictionary entries specific to this type of action.

NOTE Remote go-to actions cannot be used with embedded files; see 12.6.4.4, “Embedded Go-To Actions.”

Table 200 – Additional entries specific to a remote go-to action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **GoToR** for a remote go-to action.
F	file specification	*(Required)* The file in which the destination shall be located.
D	name, byte string, or array	*(Required)* The destination to jump to (see 12.3.2, “Destinations”). If the value is an array defining an explicit destination (as described under 12.3.2.2, “Explicit Destinations”), its first element shall be a page number within the remote document rather than an indirect reference to a page object in the current document. The first page shall be numbered 0.
NewWindow	boolean	*(Optional; PDF 1.2)* A flag specifying whether to open the destination document in a new window. If this flag is **false**, the destination document replaces the current document in the same window. If this entry is absent, the conforming reader should behave in accordance with its preference.

12.6.4.4 Embedded Go-To Actions

An *embedded go-to action (PDF 1.6)* is similar to a remote go-to action but allows jumping to or from a PDF file that is embedded in another PDF file (see 7.11.4, “Embedded File Streams”). Embedded files may be associated with file attachment annotations (see 12.5.6.15, “File Attachment Annotations”) or with entries in the **EmbeddedFiles** name tree (see 7.7.4, “Name Dictionary”). Embedded files may in turn contain embedded files. Table 201 shows the action dictionary entries specific to embedded go-to actions.

Embedded go-to actions provide a complete facility for linking between a file in a hierarchy of nested embedded files and another file in the same or different hierarchy. The following terminology shall be used:

- The *source* is the document containing the embedded go-to action.
- The *target* is the document in which the destination lives.
- The **T** entry in the action dictionary is a target dictionary that locates the target in relation to the source, in much the same way that a relative path describes the physical relationship between two files in a file system. Target dictionaries may be nested recursively to specify one or more intermediate targets before reaching the final one. As the hierarchy is navigated, each intermediate target shall be referred to as the *current document*. Initially, the source is the current document.

NOTE It is an error for a target dictionary to have an infinite cycle (for example, one where a target dictionary refers to itself). Conforming readers should attempt to detect such cases and refuse to execute the action if found.

- A *child* document shall be one that is embedded within another PDF file.
- The document in which a file is embedded shall be its *parent*.
- A *root document* is one that is not embedded in another PDF file. The target and source may be contained in root documents or embedded documents.

Table 201 – Additional entries specific to an embedded go-to action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **GoToE** for an embedded go-to action.
F	file specification	*(Optional)* The root document of the target relative to the root document of the source. If this entry is absent, the source and target share the same root document.
D	name, byte string, or array	*(Required)* The destination in the target to jump to (see 12.3.2, “Destinations”).
NewWindow	boolean	*(Optional)* If **true**, the destination document should be opened in a new window; if **false**, the destination document should replace the current document in the same window. If this entry is absent, the conforming reader should act according to its preference.
T	dictionary	*(Optional if **F** is present; otherwise required)* A *target dictionary* (see Table 202) specifying path information to the target document. Each target dictionary specifies one element in the full path to the target and may have nested target dictionaries specifying additional elements.

Table 202 – Entries specific to a target dictionary

Key	Type	Value
R	name	*(Required)* Specifies the relationship between the current document and the target (which may be an intermediate target). Valid values are **P** (the target is the parent of the current document) and **C** (the target is a child of the current document).
N	byte string	*(Required if the value of **R** is **C** and the target is located in the **EmbeddedFiles** name tree; otherwise, it shall be absent)* The name of the file in the **EmbeddedFiles** name tree.
P	integer or byte string	*(Required if the value of **R** is **C** and the target is associated with a file attachment annotation; otherwise, it shall be absent)* If the value is an integer, it specifies the page number (zero-based) in the current document containing the file attachment annotation. If the value is a string, it specifies a named destination in the current document that provides the page number of the file attachment annotation.
A	integer or text string	*(Required if the value of **R** is **C** and the target is associated with a file attachment annotation; otherwise, it shall be absent)* If the value is an integer, it specifies the index (zero-based) of the annotation in the **Annots** array (see Table 30) of the page specified by **P**. If the value is a text string, it specifies the value of **NM** in the annotation dictionary (see Table 164).
T	dictionary	*(Optional)* A target dictionary specifying additional path information to the target document. If this entry is absent, the current document is the target file containing the destination.

EXAMPLE The following example illustrates several possible relationships between source and target. Each object shown is an action dictionary for an embedded go-to action.

```
1 0 obj                    % Link to a child
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /T  << /R /C
                /N (Embedded document) >>
    >>
endobj


2 0 obj                    % Link to the parent
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /T << /R /P >>
    >>
endobj

3 0 obj                    % Link to a sibling
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /T <<   /R /P
                /T <<   /R /C
                        /N (Another embedded document) >>
            >>
    >>
endobj

4 0 obj                    % Link to an embedded file in an external document
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /F (someFile.pdf)
        /T <<    /R /C
                /N (Embedded document) >>
    >>
endobj

5 0 obj                    % Link from an embedded file to a normal file
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /F (someFile.pdf)
    >>
endobj

6 0 obj                    % Link to a grandchild
    << /Type /Action
        /S /GoToE
        /D (Chapter 1)
        /T <<   /R  /C
                /N (Embedded document)
                /T << /R /C
                        /P (A destination name)
                        /A (annotName)
                    >>
            >>
    >>
endobj

7 0 obj                    % Link to a niece/nephew through the source's parent
```

```
    << /Type /Action
         /S /GoToE
         /D (destination)
         /T <<   /R /P
                    /T << /R /C
                              /N (Embedded document)
                              /T <<   /R /C
                                         /P 3
                                         /A (annotName)
                                   >>
                         >>
               >>
    >>
endobj
```

12.6.4.5 Launch Actions

A *launch action* launches an application or opens or prints a document. Table 203 shows the action dictionary entries specific to this type of action.

The optional **Win**, **Mac**, and **Unix** entries allow the action dictionary to include platform-specific parameters for launching the designated application. If no such entry is present for the given platform, the **F** entry shall be used instead. Table 203 shows the platform-specific launch parameters for the Windows platform. Parameters for the Mac OS and UNIX platforms are not yet defined at the time of publication.

Table 203 – Additional entries specific to a launch action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Launch** for a launch action.
F	file specification	*(Required if none of the entries **Win**, **Mac**, or **Unix** is present)* The application that shall be launched or the document that shall be opened or printed. If this entry is absent and the conforming reader does not understand any of the alternative entries, it shall do nothing.
Win	dictionary	*(Optional)* A dictionary containing Windows-specific launch parameters (see Table 204).
Mac	(undefined)	*(Optional)* Mac OS–specific launch parameters; not yet defined.
Unix	(undefined)	*(Optional)* UNIX-specific launch parameters; not yet defined.
NewWindow	boolean	*(Optional; PDF 1.2)* A flag specifying whether to open the destination document in a new window. If this flag is **false**, the destination document replaces the current document in the same window. If this entry is absent, the conforming reader should behave in accordance with its current preference. This entry shall be ignored if the file designated by the **F** entry is not a PDF document.

Table 204 – Entries in a Windows launch parameter dictionary

Key	Type	Value
F	byte string	*(Required)* The file name of the application that shall be launched or the document that shall be opened or printed, in standard Windows pathname format. If the name string includes a backslash character (\), the backslash shall itself be preceded by a backslash. This value shall be a simple string; it is not a file specification.

Table 204 – Entries in a Windows launch parameter dictionary (continued)

Key	Type	Value
D	byte string	*(Optional)* A bye string specifying the default directory in standard DOS syntax.
O	ASCII string	*(Optional)* An ASCII string specifying the operation to perform: open Open a document. print Print a document. If the **F** entry designates an application instead of a document, this entry shall be ignored and the application shall be launched. Default value: open.
P	byte string	*(Optional)* A parameter string that shall be passed to the application designated by the **F** entry. This entry shall be omitted if **F** designates a document.

12.6.4.6 Thread Actions

A *thread action* jumps to a specified bead on an article thread (see 12.4.3, “Articles”), in either the current document or a different one. Table 205 shows the action dictionary entries specific to this type of action.

Table 205 – Additional entries specific to a thread action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Thread** for a thread action.
F	file specification	*(Optional)* The file containing the thread. If this entry is absent, the thread is in the current file.
D	dictionary, integer, or text string	*(Required)* The destination thread, specified in one of the following forms: An indirect reference to a thread dictionary (see 12.4.3, “Articles”). In this case, the thread shall be in the current file. The index of the thread within the **Threads** array of its document’s **Catalog** (see 7.7.2, “Document Catalog”). The first thread in the array has index 0. The title of the thread as specified in its thread information dictionary (see Table 160). If two or more threads have the same title, the one appearing first in the document **Catalog’s Threads** array shall be used.
B	dictionary or integer	*(Optional)* The bead in the destination thread, specified in one of the following forms: An indirect reference to a bead dictionary (see 12.4.3, “Articles”). In this case, the thread shall be in the current file. The index of the bead within its thread. The first bead in a thread has index 0.

12.6.4.7 URI Actions

A *uniform resource identifier* (URI) is a string that identifies (*resolves* to) a resource on the Internet—typically a file that is the destination of a hypertext link, although it may also resolve to a query or other entity. (URIs are described in Internet RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*; see the Bibliography.)

A *URI action* causes a URI to be resolved. Table 206 shows the action dictionary entries specific to this type of action.

Table 206 – Additional entries specific to a URI action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **URI** for a URI action.
URI	ASCII string	*(Required)* The uniform resource identifier to resolve, encoded in 7-bit ASCII.
IsMap	boolean	*(Optional)* A flag specifying whether to track the mouse position when the URI is resolved (see the discussion following this Table). Default value: **false**. This entry applies only to actions triggered by the user's clicking an annotation; it shall be ignored for actions associated with outline items or with a document's **OpenAction** entry.

If the **IsMap** flag is **true** and the user has triggered the URI action by clicking an annotation, the coordinates of the mouse position at the time the action has been triggered shall be transformed from device space to user space and then offset relative to the upper-left corner of the annotation rectangle (that is, the value of the **Rect** entry in the annotation with which the URI action is associated).

EXAMPLE 1 If the mouse coordinates in user space are (x_m, y_m) and the annotation rectangle extends from (ll_x, ll_y) at the lower-left to (ur_x, ur_y) at the upper-right, the final coordinates (x_f, y_f) are as follows:

$$(x_f = x_m - ll_x)$$

$$y_f = ur_y - y_m$$

If the resulting coordinates (x_f, y_f) are fractional, they shall be rounded to the nearest integer values. They shall then be appended to the URI to be resolved, separated by COMMAS (2Ch) and preceded by a QUESTION MARK (3Fh), as shown in this example:

EXAMPLE 2 http://www.adobe.com/intro?100,200

NOTE 1 To support URI actions, a PDF document's **Catalog** (see 7.7.2, "Document Catalog") may include a **URI** entry whose value is a *URI dictionary*. Only one entry shall be defined for such a dictionary (see Table 207).

Table 207 – Entry in a URI dictionary

Key	Type	Value
Base	ASCII string	*(Optional)* The *base URI* that shall be used in resolving relative URI references. URI actions within the document may specify URIs in partial form, to be interpreted relative to this base address. If no base URI is specified, such partial URIs shall be interpreted relative to the location of the document itself. The use of this entry is parallel to that of the body element <BASE>, as described in the *HTML 4.01 Specification* (see the Bibliography).

NOTE 2 The **Base** entry allows the URI of the document to be recorded in situations in which the document may be accessed out of context. For example, if a document has been moved to a new location but contains relative links to other documents that have not been moved, the **Base** entry could be used to refer such links to the true location of the other documents, rather than that of the moved document.

12.6.4.8 Sound Actions

A *sound action (PDF 1.2)* plays a sound through the computer's speakers. Table 208 shows the action dictionary entries specific to this type of action. Sounds are discussed in 13.3, "Sounds."

Table 208 – Additional entries specific to a sound action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Sound** for a sound action.
Sound	stream	*(Required)* A sound object defining the sound that shall be played (see 13.3, "Sounds").
Volume	number	*(Optional)* The volume at which to play the sound, in the range −1.0 to 1.0. Default value: 1.0.
Synchronous	boolean	*(Optional)* A flag specifying whether to play the sound synchronously or asynchronously. If this flag is **true**, the conforming reader retains control, allowing no further user interaction other than canceling the sound, until the sound has been completely played. Default value: **false**.
Repeat	boolean	*(Optional)* A flag specifying whether to repeat the sound indefinitely. If this entry is present, the **Synchronous** entry shall be ignored. Default value: **false**.
Mix	boolean	*(Optional)* A flag specifying whether to mix this sound with any other sound already playing. If this flag is **false**, any previously playing sound shall be stopped before starting this sound; this can be used to stop a repeating sound (see **Repeat**). Default value: **false**.

12.6.4.9 Movie Actions

A *movie action (PDF 1.2)* can be used to play a movie in a floating window or within the annotation rectangle of a movie annotation (see 12.5.6.17, "Movie Annotations" and 13.4, "Movies"). The movie annotation shall be associated with the page that is the destination of the link annotation or outline item containing the movie action, or with the page object with which the action is associated.

NOTE A movie action by itself does not guarantee that the page the movie is on will be displayed before attempting to play the movie; such page change actions shall be done explicitly.

The contents of a movie action dictionary are identical to those of a movie activation dictionary (see Table 296), with the additional entries shown in Table 209. The contents of the activation dictionary associated with the movie annotation provide the default values. Any information specified in the movie action dictionary overrides these values.

Table 209 – Additional entries specific to a movie action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Movie** for a movie action.
Annotation	dictionary	*(Optional)* An indirect reference to a movie annotation identifying the movie that shall be played.
T	text string	*(Optional)* The title of a movie annotation identifying the movie that shall be played. The dictionary shall include either an ***Annotation*** or a ***T*** entry but not both.

Table 209 – Additional entries specific to a movie action (continued)

Key	Type	Value
Operation	name	*(Optional)* The operation that shall be performed on the movie: PlayStart playing the movie, using the play mode specified by the dictionary's **Mode** entry (see Table 296). If the movie is currently paused, it shall be repositioned to the beginning before playing (or to the starting point specified by the dictionary's **Start** entry, if present). Stop Stop playing the movie. Pause Pause a playing movie. Resume Resume a paused movie. Default value: Play.

12.6.4.10 Hide Actions

A *hide action (PDF 1.2)* hides or shows one or more annotations on the screen by setting or clearing their Hidden flags (see 12.5.3, "Annotation Flags"). This type of action can be used in combination with appearance streams and trigger events (Sections 12.5.5, "Appearance Streams," and 12.6.3, "Trigger Events") to display pop-up help information on the screen.

NOTE The **E** (enter) and **X** (exit) trigger events in an annotation's additional-actions dictionary can be used to show and hide the annotation when the user rolls the cursor in and out of its active area on the page. This can be used to pop up a help label, or tool tip, describing the effect of clicking at that location on the page.

Table 210 shows the action dictionary entries specific to this type of action.

Table 210 – Additional entries specific to a hide action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Hide** for a hide action.
T	dictionary, text string, or array	*(Required)* The annotation or annotations to be hidden or shown, shall be specified in any of the following forms: An indirect reference to an annotation dictionary A text string giving the fully qualified field name of an interactive form field whose associated widget annotation or annotations are to be affected (see 12.7.3.2, "Field Names") An array of such dictionaries or text strings
H	boolean	*(Optional)* A flag indicating whether to hide the annotation (**true**) or show it (**false**). Default value: **true**.

12.6.4.11 Named Actions

Table 211 lists several *named actions (PDF 1.2)* that conforming readers shall support; further names may be added in the future.

Table 211 – Named actions

Name	Action
NextPage	Go to the next page of the document.
PrevPage	Go to the previous page of the document.
FirstPage	Go to the first page of the document.
LastPage	Go to the last page of the document.

NOTE Conforming readers may support additional, nonstandard named actions, but any document using them is not portable. If the viewer encounters a named action that is inappropriate for a viewing platform, or if the viewer does not recognize the name, it shall take no action.

Table 212 shows the action dictionary entries specific to named actions.

Table 212 – Additional entries specific to named actions

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Named** for a named action.
N	name	*(Required)* The name of the action that shall be performed (see Table 211).

12.6.4.12 Set-OCG-State Actions

A *set-OCG-state action (PDF 1.5)* sets the state of one or more optional content groups (see 8.11, “Optional Content”). Table 213 shows the action dictionary entries specific to this type of action.

Table 213 – Additional entries specific to a set-OCG-state action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **SetOCGState** for a set-OCG-state action.
State	array	*(Required)* An array consisting of any number of sequences beginning with a name object (**ON**, **OFF**, or **Toggle**) followed by one or more optional content group dictionaries. The array elements shall be processed from left to right; each name shall be applied to the subsequent groups until the next name is encountered: **ON** sets the state of subsequent groups to **ON** **OFF** sets the state of subsequent groups to **OFF** **Toggle** reverses the state of subsequent groups.
PreserveRB	boolean	*(Optional)* If **true**, indicates that radio-button state relationships between optional content groups (as specified by the **RBGroups** entry in the current configuration dictionary; see Table 101) should be preserved when the states in the **State** array are applied. That is, if a group is set to **ON** (either by **ON** or **Toggle**) during processing of the **State** array, any other groups belonging to the same radio-button group shall be turned **OFF**. If a group is set to **OFF**, there is no effect on other groups. If **PreserveRB** is **false**, radio-button state relationships, if any, shall be ignored. Default value: **true**.

When a set-OCG-state action is performed, the **State** array shall be processed from left to right. Each name shall be applied to subsequent groups in the array until the next name is encountered, as shown in the following example.

EXAMPLE 1
```
<< /S /SetOCGState
    /State [/OFF 2 0 R 3 0 R /Toggle 16 0 R 19 0 R /ON 5 0 R]
>>
```

A group may appear more than once in the **State** array; its state shall be set each time it is encountered, based on the most recent name. **ON**, **OFF** and **Toggle** sequences have no required order. More than one sequence in the array may contain the same name.

EXAMPLE 2 If the array contained [/OFF 1 0 R /Toggle 1 0 R], the group’s state would be **ON** after the action was performed.

NOTE While the specification allows a group to appear more than once in the ***State*** array, this is not intended to implement animation or any other sequential drawing operations. PDF processing applications are free to accumulate all state changes and apply only the net changes simultaneously to all affected groups before redrawing.

12.6.4.13 Rendition Actions

A *rendition action (PDF 1.5)* controls the playing of multimedia content (see 13.2, “Multimedia”). This action may be used in the following ways:

- To begin the playing of a rendition object (see 13.2.3, “Renditions”), associating it with a screen annotation (see 12.5.6.18, “Screen Annotations”). The screen annotation specifies where the rendition shall be played unless otherwise specified.
- To stop, pause, or resume a playing rendition.
- To trigger the execution of a JavaScript script that may perform custom operations.

Table 214 lists the entries in a rendition action dictionary.

Table 214 – Additional entries specific to a rendition action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Rendition** for a rendition action.
R	dictionary	(*Required when **OP** is present with a value of 0 or 4; otherwise optional*) A rendition object (see 13.2.3, “Renditions”).
AN	dictionary	(*Required if **OP** is present with a value of 0, 1, 2, 3 or 4; otherwise optional*) An indirect reference to a screen annotation (see 12.5.6.18, “Screen Annotations”).
OP	integer	(*Required if **JS** is not present; otherwise optional*) The operation to perform when the action is triggered. Valid values shall be: 0 If no rendition is associated with the annotation specified by **AN**, play the rendition specified by **R**, associating it with the annotation. If a rendition is already associated with the annotation, it shall be stopped, and the new rendition shall be associated with the annotation. 1 Stop any rendition being played in association with the annotation specified by **AN**, and remove the association. If no rendition is being played, there is no effect. 2 Pause any rendition being played in association with the annotation specified b y **AN**. If no rendition is being played, there is no effect. 3 Resume any rendition being played in association with the annotation specified by **AN**. If no rendition is being played or the rendition is not paused, there is no effect. 4 Play the rendition specified by **R**, associating it with the annotation specified by AN. If a rendition is already associated with the annotation, resume the rendition if it is paused; otherwise, do nothing.
JS	text string or stream	(*Required if **OP** is not present; otherwise optional*) A text string or stream containing a JavaScript script that shall be executed when the action is triggered.

Either the **JS** entry or the **OP** entry shall be present. If both are present, **OP** is considered a fallback that shall be executed if the conforming reader is unable to execute JavaScripts. If **OP** has an unrecognized value and there is no **JS** entry, the action is invalid.

In some situations, a pause (**OP** value of 2) or resume (**OP** value of 3) operation may not make sense or the player may not support it. In such cases, the user should be notified of the failure to perform the operation.

EXAMPLE A JPEG image

Before a rendition action is executed, the conforming reader shall make sure that the **P** entry of the screen annotation dictionary references a valid page object and that the annotation is present in the page object's **Annots** array (see Table 30).

A rendition may play in the rectangle occupied by a screen annotation, even if the annotation itself is not visible; for example, if its Hidden or NoView flags (see Table 165) are set. If a screen annotation is not visible because its location on the page is not being displayed by the viewer, the rendition is not visible. However, it may become visible if the view changes, such as by scrolling.

12.6.4.14 Transition Actions

A *transition action (PDF 1.5)* may be used to control drawing during a sequence of actions. As discussed in 12.6.2, "Action Dictionaries," the **Next** entry in an action dictionary may specify a sequence of actions. Conforming readers shall normally suspend drawing when such a sequence begins and resume drawing when it ends. If a transition action is present during a sequence, the conforming reader shall render the state of the page viewing area as it exists after completion of the previous action and display it using a transition specified in the action dictionary (see Table 215). Once this transition completes, drawing shall be suspended again.

Table 215 – Additional entries specific to a transition action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **Trans** for a transition action.
Trans	dictionary	*(Required)* The transition to use for the update of the display (see Table 162).

12.6.4.15 Go-To-3D-View Actions

A *go-to-3D-view action (PDF 1.6)* identifies a 3D annotation and specifies a view for the annotation to use (see 13.6, "3D Artwork"). Table 216 shows the entries in a go-to-3D-view action dictionary.

Table 216 – Additional entries specific to a go-to-3D-view action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **GoTo3DView** for a transition action.
TA	dictionary	*(Required)* The target annotation for which to set the view.
V	(various)	*(Required)* The view to use. It may be one of the following types: A 3D view dictionary (see 13.6.4, "3D Views"). An integer specifying an index into the **VA** array in the 3D stream (see Table 300). A text string matching the **IN** entry in one of the views in the **VA** array (see Table 304). A name that indicates the first (**F**), last (**L**), next (**N**), previous (**P**), or default (**D**) entries in the **VA** array; see discussion following this Table.

The **V** entry selects the view to apply to the annotation specified by **TA**. This view may be one of the predefined views specified by the **VA** entry of the 3D stream (see Table 300) or a unique view specified here.

If the predefined view is specified by the names **N** (next) or **P** (previous), it should be interpreted in the following way:

- When the last view applied was specified by means of the **VA** array, **N** and **P** indicate the next and previous entries, respectively, in the **VA** array (wrapping around if necessary).
- When the last view was not specified by means of **VA**, using **N** or **P** should result in reverting to the default view.

12.6.4.16 JavaScript Actions

Upon invocation of a JavaScript action, a conforming processor shall execute a script that is written in the JavaScript programming language. Depending on the nature of the script, various interactive form fields in the document may update their values or change their visual appearances. Mozilla Development Center's Client-Side JavaScript Reference and the Adobe JavaScript for Acrobat API Reference (see the Bibliography) give details on the contents and effects of JavaScript scripts. Table 217 shows the action dictionary entries specific to this type of action.

Table 217 – Additional entries specific to a JavaScript action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **JavaScript** for a JavaScript action.
JS	text string or text stream	*(Required)* A text string or text stream containing the JavaScript script to be executed. ***PDFDocEncoding*** or Unicode encoding (the latter identified by the Unicode prefix *U+FEFF*) shall be used to encode the contents of the string or stream.

To support the use of parameterized function calls in JavaScript scripts, the **JavaScript** entry in a PDF document's name dictionary (see 7.7.4, "Name Dictionary") may contain a name tree that maps name strings to document-level JavaScript actions. When the document is opened, all of the actions in this name tree shall be executed, defining JavaScript functions for use by other scripts in the document.

NOTE The name strings associated with individual JavaScript actions in the name dictionary serve merely as a convenient means for organizing and packaging scripts. The names are arbitrary and need not bear any relation to the JavaScript name space.

12.7 Interactive Forms

12.7.1 General

An *interactive form (PDF 1.2)*—sometimes referred to as an *AcroForm*—is a collection of *fields* for gathering information interactively from the user. A PDF document may contain any number of fields appearing on any combination of pages, all of which make up a single, global interactive form spanning the entire document. Arbitrary subsets of these fields can be imported or exported from the document; see 12.7.5, "Form Actions."

NOTE 1 Interactive forms should not be confused with form XObjects (see 8.10, "Form XObjects"). Despite the similarity of names, the two are different, unrelated types of objects.

Each field in a document's interactive form shall be defined by a *field dictionary* (see 12.7.3, "Field Dictionaries"). For purposes of definition and naming, the fields can be organized hierarchically and can inherit attributes from their ancestors in the field hierarchy. A field's children in the hierarchy may also include widget annotations (see 12.5.6.19, "Widget Annotations") that define its appearance on the page. A field that has children that are fields is called a *non-terminal field*. A field that does not have children that are fields is called a *terminal field*.

A terminal field may have children that are widget annotations (see 12.5.6.19, “Widget Annotations“) that define its appearance on the page. As a convenience, when a field has only a single associated widget annotation, the contents of the field dictionary and the annotation dictionary (12.5.2, “Annotation Dictionaries”) may be merged into a single dictionary containing entries that pertain to both a field and an annotation. (This presents no ambiguity, since the contents of the two kinds of dictionaries do not conflict.) If such an object defines an appearance stream, the appearance shall be consistent with the object’s current value as a field.

NOTE 2 Fields containing text whose contents are not known in advance may need to construct their appearance streams dynamically instead of defining them statically in an appearance dictionary; see 12.7.3.3, “Variable Text.”

12.7.2 Interactive Form Dictionary

The contents and properties of a document’s interactive form shall be defined by an *interactive form dictionary* that shall be referenced from the **AcroForm** entry in the document catalogue (see 7.7.2, “Document Catalog”). Table 218 shows the contents of this dictionary.

Table 218 – Entries in the interactive form dictionary

Key	Type	Value
Fields	array	*(Required)* An array of references to the document’s *root fields* (those with no ancestors in the field hierarchy).
NeedAppearances	boolean	*(Optional)* A flag specifying whether to construct appearance streams and appearance dictionaries for all widget annotations in the document (see 12.7.3.3, “Variable Text”). Default value: **false**.
SigFlags	integer	*(Optional; PDF 1.3)* A set of flags specifying various document-level characteristics related to signature fields (see Table 219, and 12.7.4.5, “Signature Fields”). Default value: 0.
CO	array	*(Required if any fields in the document have additional-actions dictionaries containing a* ***C*** *entry; PDF 1.3)* An array of indirect references to field dictionaries with calculation actions, defining the *calculation order* in which their values will be recalculated when the value of any field changes (see 12.6.3, “Trigger Events”).
DR	dictionary	*(Optional)* A resource dictionary (see 7.8.3, “Resource Dictionaries”) containing default resources (such as fonts, patterns, or colour spaces) that shall be used by form field appearance streams. At a minimum, this dictionary shall contain a **Font** entry specifying the resource name and font dictionary of the default font for displaying text.
DA	string	*(Optional)* A document-wide default value for the **DA** attribute of variable text fields (see 12.7.3.3, “Variable Text”).
Q	integer	*(Optional)* A document-wide default value for the **Q** attribute of variable text fields (see 12.7.3.3, “Variable Text”).
XFA	stream or array	*(Optional; PDF 1.5)* A stream or array containing an *XFA resource*, whose format shall be described by the *Data Package (XDP) Specification*. (see the Bibliography). The value of this entry shall be either a stream representing the entire contents of the XML Data Package or an array of text string and stream pairs representing the individual packets comprising the XML Data Package. See 12.7.8, “XFA Forms,” for more information.

The value of the interactive form dictionary's **SigFlags** entry is an unsigned 32-bit integer containing flags specifying various document-level characteristics related to signature fields (see 12.7.4.5, "Signature Fields"). Bit positions within the flag word shall be numbered from 1 (low-order) to 32 (high-order). Table 219 shows the meanings of the flags; all undefined flag bits shall be reserved and shall be set to 0.

Table 219 – Signature flags

Bit position	Name	Meaning
1	SignaturesExist	If set, the document contains at least one signature field. This flag allows a conforming reader to enable user interface items (such as menu items or pushbuttons) related to signature processing without having to scan the entire document for the presence of signature fields.
2	AppendOnly	If set, the document contains signatures that may be invalidated if the file is saved (written) in a way that alters its previous contents, as opposed to an incremental update. Merely updating the file by appending new information to the end of the previous version is safe (see H.7, "Updating Example"). Conforming readers may use this flag to inform a user requesting a full save that signatures will be invalidated and require explicit confirmation before continuing with the operation.

12.7.3 Field Dictionaries

12.7.3.1 General

Each field in a document's interactive form shall be defined by a *field dictionary*, which shall be an indirect object. The field dictionaries may be organized hierarchically into one or more tree structures. Many field attributes are *inheritable*, meaning that if they are not explicitly specified for a given field, their values are taken from those of its parent in the field hierarchy. Such inheritable attributes shall be designated as such in the Tables 220 and 221. The designation *(Required; inheritable)* means that an attribute shall be defined for every field, whether explicitly in its own field dictionary or by inheritance from an ancestor in the hierarchy. Table 220 shows those entries that are common to all field dictionaries, regardless of type. Entries that pertain only to a particular type of field are described in the relevant sub-clauses in Table 220.

Table 220 – Entries common to all field dictionaries

Key	Type	Value
FT	name	*(Required for terminal fields; inheritable)* The type of field that this dictionary describes: **Btn** Button (see 12.7.4.2, "Button Fields") **Tx** Text (see 12.7.4.3, "Text Fields") **Ch** Choice (see 12.7.4.4, "Choice Fields") **Sig** *(PDF 1.3)* Signature (see 12.7.4.5, "Signature Fields") This entry may be present in a non-terminal field (one whose descendants are fields) to provide an inheritable ***FT*** value. However, a non-terminal field does not logically have a type of its own; it is merely a container for inheritable attributes that are intended for descendant terminal fields of any type.
Parent	dictionary	*(Required if this field is the child of another in the field hierarchy; absent otherwise)* The field that is the immediate parent of this one (the field, if any, whose **Kids** array includes this field). A field can have at most one parent; that is, it can be included in the **Kids** array of at most one other field.

Table 220 – Entries common to all field dictionaries (continued)

Key	Type	Value
Kids	array	*(Sometimes required, as described below)* An array of indirect references to the immediate children of this field. In a non-terminal field, the **Kids** array shall refer to field dictionaries that are immediate descendants of this field. In a terminal field, the **Kids** array ordinarily shall refer to one or more separate widget annotations that are associated with this field. However, if there is only one associated widget annotation, and its contents have been merged into the field dictionary, **Kids** shall be omitted.
T	text string	*(Optional)* The partial field name (see 12.7.3.2, “Field Names”).
TU	text string	*(Optional; PDF 1.3)* An alternate field name that shall be used in place of the actual field name wherever the field shall be identified in the user interface (such as in error or status messages referring to the field). This text is also useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes (see 14.9.3, “Alternate Descriptions”).
TM	text string	*(Optional; PDF 1.3)* The *mapping name* that shall be used when exporting interactive form field data from the document.
Ff	integer	*(Optional; inheritable)* A set of flags specifying various characteristics of the field (see Table 221). Default value: 0.
V	(various)	*(Optional; inheritable)* The field’s value, whose format varies depending on the field type. See the descriptions of individual field types for further information.
DV	(various)	*(Optional; inheritable)* The default value to which the field reverts when a reset-form action is executed (see 12.7.5.3, “Reset-Form Action”). The format of this value is the same as that of **V**.
AA	dictionary	*(Optional; PDF 1.2)* An additional-actions dictionary defining the field’s behaviour in response to various trigger events (see 12.6.3, “Trigger Events”). This entry has exactly the same meaning as the **AA** entry in an annotation dictionary (see 12.5.2, “Annotation Dictionaries”).

The value of the field dictionary’s **Ff** entry is an unsigned 32-bit integer containing flags specifying various characteristics of the field. Bit positions within the flag word shall be numbered from 1 (low-order) to 32 (high-order). The flags shown in Table 221 are common to all types of fields. Flags that apply only to specific field types are discussed in the sub-clauses describing those types. All undefined flag bits shall be reserved and shall be set to 0.

Table 221 – Field flags common to all field types

Bit position	Name	Meaning
1	ReadOnly	If set, the user may not change the value of the field. Any associated widget annotations will not interact with the user; that is, they will not respond to mouse clicks or change their appearance in response to mouse motions. This flag is useful for fields whose values are computed or imported from a database.
2	Required	If set, the field shall have a value at the time it is exported by a submit-form action (see 12.7.5.2, “Submit-Form Action”).
3	NoExport	If set, the field shall not be exported by a submit-form action (see 12.7.5.2, “Submit-Form Action”).

12.7.3.2 Field Names

The **T** entry in the field dictionary (see Table 220) holds a text string defining the field's *partial field name*. The *fully qualified field name* is not explicitly defined but shall be constructed from the partial field names of the field and all of its ancestors. For a field with no parent, the partial and fully qualified names are the same. For a field that is the child of another field, the fully qualified name shall be formed by appending the child field's partial name to the parent's fully qualified name, separated by a PERIOD (2Eh) as shown:

parent's_full_name.*child's_partial_name*

EXAMPLE If a field with the partial field name PersonalData has a child whose partial name is Address, which in turn has a child with the partial name ZipCode, the fully qualified name of this last field is

PersonalData.Address.ZipCode

Because the PERIOD is used as a separator for fully qualified names, a partial name shall not contain a PERIOD character. Thus, all fields descended from a common ancestor share the ancestor's fully qualified field name as a common prefix in their own fully qualified names.

It is possible for different field dictionaries to have the same fully qualified field name if they are descendants of a common ancestor with that name and have no partial field names (**T** entries) of their own. Such field dictionaries are different representations of the same underlying field; they should differ only in properties that specify their visual appearance. In particular, field dictionaries with the same fully qualified field name shall have the same field type (**FT**), value (**V**), and default value (**DV**).

12.7.3.3 Variable Text

When the contents and properties of a field are known in advance, its visual appearance can be specified by an appearance stream defined in the PDF file (see 12.5.5, "Appearance Streams," and 12.5.6.19, "Widget Annotations"). In some cases, however, the field may contain text whose value is not known until viewing time.

NOTE Examples include text fields to be filled in with text typed by the user from the keyboard, scrollable list boxes whose contents are determined interactively at the time the document is displayed and fields containing current dates or values calculated by a JavaScript.

In such cases, the PDF document cannot provide a statically defined appearance stream for displaying the field. Instead, the conforming reader shall construct an appearance stream dynamically at viewing time. The dictionary entries shown in Table 222 provide general information about the field's appearance that can be combined with the specific text it contains to construct an appearance stream.

Table 222 – Additional entries common to all fields containing variable text

Key	Type	Value
DA	string	*(Required; inheritable)* The *default appearance string* containing a sequence of valid page-content graphics or text state operators that define such properties as the field's text size and colour.
Q	integer	*(Optional; inheritable)* A code specifying the form of *quadding* (justification) that shall be used in displaying the text: 0 Left-justified 1 Centered 2 Right-justified Default value: 0 (left-justified).
DS	text string	*(Optional; PDF 1.5)* A default style string, as described in 12.7.3.4, "Rich Text Strings."

Table 222 – Additional entries common to all fields containing variable text (continued)

Key	Type	Value
RV	text string or text stream	*(Optional; PDF 1.5)* A rich text string, as described in 12.7.3.4, “Rich Text Strings.”

The new appearance stream becomes the normal appearance (**N**) in the appearance dictionary associated with the field’s widget annotation (see Table 168). (If the widget annotation has no appearance dictionary, the conforming reader shall create one and store it in the annotation dictionary’s **AP** entry.)

In PDF 1.5, form fields that have the RichText flag set (see Table 226) specify formatting information as described in 12.7.3.4, “Rich Text Strings.” For these fields, the following conventions are not used, and the entire annotation appearance shall be regenerated each time the value is changed.

For non-rich text fields, the appearance stream—which, like all appearance streams, is a form XObject—has the contents of its form dictionary initialized as follows:

- The resource dictionary (**Resources**) shall be created using resources from the interactive form dictionary’s **DR** entry (see Table 218).
- The lower-left corner of the bounding box (**BBox**) is set to coordinates (0, 0) in the form coordinate system. The box’s top and right coordinates are taken from the dimensions of the annotation rectangle (the **Rect** entry in the widget annotation dictionary).
- All other entries in the appearance stream’s form dictionary are set to their default values (see 8.10, “Form XObjects”).

EXAMPLE The appearance stream includes the following section of marked content, which represents the portion of the stream that draws the text:

```
/Tx  BMC                                          % Begin marked content with tag Tx
    q                                             % Save graphics state
            …Any required graphics state changes, such as clipping…
        BT                                        % Begin text object
            …Default appearance string (DA)…
            …Text-positioning and text-showing operators to show the variable text…
        ET                                        % End text object
    Q                                                 % Restore graphics state
EMC                                               % End marked content
```

The **BMC** (begin marked content) and **EMC** (end marked content) operators are discussed in 14.6, “Marked Content.” **q** (save graphics state) and **Q** (restore graphics state) are discussed in 8.4.4, “Graphics State Operators.” **BT** (begin text object) and **ET** (end text object) are discussed in 9.4, “Text Objects.” See Example 1 in 12.7.8, “XFA Forms” for an example.

The default appearance string (**DA**) contains any graphics state or text state operators needed to establish the graphics state parameters, such as text size and colour, for displaying the field’s variable text. Only operators that are allowed within text objects shall occur in this string (see Figure 9). At a minimum, the string shall include a **Tf** (text font) operator along with its two operands, *font* and *size*. The specified *font* value shall match a resource name in the **Font** entry of the default resource dictionary (referenced from the **DR** entry of the interactive form dictionary; see Table 218). A zero value for *size* means that the font shall be *auto-sized*: its size shall be computed as a function of the height of the annotation rectangle.

The default appearance string shall contain at most one **Tm** (text matrix) operator. If this operator is present, the conforming reader shall replace the horizontal and vertical translation components with positioning values it determines to be appropriate, based on the field value, the quadding (**Q**) attribute, and any layout rules it employs. If the default appearance string contains no **Tm** operator, the viewer shall insert one in the appearance stream (with appropriate horizontal and vertical translation components) after the default appearance string and before the text-positioning and text-showing operators for the variable text.

To update an existing appearance stream to reflect a new field value, the conforming reader shall first copy any needed resources from the document's **DR** dictionary (see Table 218) into the stream's **Resources** dictionary. (If the **DR** and **Resources** dictionaries contain resources with the same name, the one already in the **Resources** dictionary shall be left intact, not replaced with the corresponding value from the **DR** dictionary.) The conforming reader shall then replace the existing contents of the appearance stream from /Tx BMC to the matching EMC with the corresponding new contents as shown in Example 1 in "Check Boxes," 12.7.4, "Field Types." (If the existing appearance stream contains no marked content with tag Tx, the new contents shall be appended to the end of the original stream.)

12.7.3.4 Rich Text Strings

Beginning with PDF 1.5, the text contents of variable text form fields, as well as markup annotations, may include formatting (style) information. These *rich text strings* are fully-formed XML documents that conform to the rich text conventions specified for the XML Forms Architecture (XFA) specification, which is itself a subset of the XHTML 1.0 specification, augmented with a restricted set of CSS2 style attributes (see the Bibliography for references to all these standards).

Table 223 lists the XHTML elements that may appear in rich text strings. The <body> element is the root element; its required attributes are listed in Table 224. Other elements (<p> and <span>) contain enclosed text that may take style attributes, which are listed in Table 225. These style attributes are CSS inline style property declarations of the form *name*:*value*, with each declaration separated by a SEMICOLON (3Bh), as illustrated in the Example in "Radio Buttons," 12.7.4, "Field Types."

In PDF 1.6, PDF supports the rich text elements and attributes specified in the *XML Forms Architecture (XFA) Specification, 2.2* (see Bibliography). These rich text elements and attributes are a superset of those described in Table 223, Table 224 and Table 225. In PDF 1.7, PDF supports the rich text elements and attributes specified in the *XML Forms Architecture (XFA) Specification, 2.4* (see Bibliography).

Table 223 – XHTML elements used in rich text strings

element	**Description**
<body>	The element at the root of the XML document. Table 224 lists the required attributes for this element.
<p>	Encloses text that shall be interpreted as a paragraph. It may take the style attributes listed in Table 225.
<i>	Encloses text that shall be displayed in an italic font.
<b>	Encloses text that shall be displayed in a bold font.
<span>	Groups text solely for the purpose of applying styles (using the attributes in Table 225).

Table 224 – Attributes of the <body> element

Attribute	Description
xmlns	The default namespaces for elements within the rich text string. Shall be xmlns="http://www.w3.org/1999/xhtml" xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0".
xfa:contentType	Shall be "text/html".

Table 224 – Attributes of the <body> element (continued)

Attribute	Description
xfa:APIVersion	A string that identifies the software used to generate the rich text string. It shall be of the form software_name:software_version, where software_name identifies the software by name. It shall not contain spaces. software_version identifies the version of the software. It consists of a series of integers separated by decimal points. Each integer is a version number, the leftmost value being a major version number, with values to the right increasingly minor. When comparing strings, the versions shall be compared in order. NOTE “5.2” is less than “5.13” because 2 is less than 13; the string is not treated as a decimal number. When comparing strings with different numbers of sections, the string with fewer sections is implicitly padded on the right with sections containing “0” to make the number of sections equivalent.
xfa:spec	The version of the *XML Forms Architecture (XFA)* specification to which the rich text string complies. If the file being written conforms to PDF 1.5, then the rich text string shall conform to XFA 2.0, and this attribute shall be XFA 2.0; if the file being written conforms to PDF 1.6, then the rich text string shall conform to XFA 2.2, and this attribute shall be XFA 2.2; and if the file being written conforms to PDF 1.7, then the rich text string shall conform to XFA 2.4, and this attribute shall be XFA 2.4.

Table 225 – CSS2 style attributes used in rich text strings

Attribute	**Value**	**Description**
text-align	keyword	Horizontal alignment. Possible values: left, right, and center.
vertical-align	decimal	An amount by which to adjust the baseline of the enclosed text. A positive value indicates a superscript; a negative value indicates a subscript. The value is of the form <*decimal number*>pt, optionally preceded by a sign, and followed by “pt”. EXAMPLE -3pt, 4pt.
font-size	decimal	The font size of the enclosed text. The value is of the form <*decimal number*>pt.
font-style	keyword	Specifies the font style of the enclosed text. Possible values: normal, italic.
font-weight	keyword	The weight of the font for the enclosed text. Possible values: normal, bold, 100, 200, 300, 400, 500, 600, 700, 800, 900. *normal* is equivalent to 400, and *bold* is equivalent to 700.
font-family	list	A font name or list of font names that shall be used to display the enclosed text. (If a list is provided, the first one containing glyphs for the specified text shall be used.)
font	list	A shorthand CSS font property of the form font:<*font-style*> <*font-weight*> <*font-size*> <*font-family*>
color	RGB value	The colour of the enclosed text. It can be in one of two forms: #*rrggbb* with a 2-digit hexadecimal value for each component rgb*(rrr,ggg,bbb)* with a decimal value for each component. Although the values specified by the color property are interpreted as sRGB values, they shall be transformed into values in a non-ICC based colour space when used to generate the annotation’s appearance.

Table 225 – CSS2 style attributes used in rich text strings (continued)

Attribute	Value	Description
text-decoration	keyword	One of the following keywords: underline: The enclosed text shall be underlined. line-through: The enclosed text shall have a line drawn through it.
font-stretch	keyword	Specifies a normal, condensed or extended face from a font family. Supported values from narrowest to widest are ultra-condensed, extra-condensed, condensed, semi-condensed, normal, semi-expanded, expanded, extra-expanded, and ultra-expanded.

Rich text strings shall be specified by the **RV** entry of variable text form field dictionaries (see Table 222) and the **RC** entry of markup annotation dictionaries (see Table 170). Rich text strings may be packaged as *text streams* (see 7.9.3, “Text Streams”). Form fields using rich text streams should also have the RichText flag set (see Table 228).

A *default style string* shall be specified by the **DS** entry for free text annotations (see Table 174) or variable text form fields (see Table 222). This string specifies the default values for style attributes, which shall be used for any style attributes that are not explicitly specified for the annotation or field. All attributes listed in Table 225 are legal in the default style string. This string, in addition to the **RV** or **RC** entry, shall be used to generate the appearance.

NOTE 1 Markup annotations other than free text annotations (see 12.5.6.2, “Markup Annotations”) do not use a default style string because their appearances are implemented using platform controls requiring the conforming reader to pick an appropriate system font for display.

When a form field or annotation contains rich text strings, the *flat text* (character data) of the string should also be preserved (in the **V** entry for form fields and the **Contents** entry for annotations). This enables older readers to read and edit the data (although with loss of formatting information). The **DA** entry should be written out as well when the file is saved.

When a rich text string specifies font attributes, the conforming reader shall use font name selection as described in Section 15.3 of the CSS2 specification (see the Bibliography). Precedence should be given to the fonts in the default resources dictionary, as specified by the **DR** entry in Table 218.

EXAMPLE The following example illustrates the entries in a widget annotation dictionary for rich text. The **DS** entry specifies the default font. The **RV** entry contains two paragraphs of rich text: the first paragraph specifies bold and italic text in the default font; the second paragraph changes the font size.

```
/DS (font: 18pt Arial)              % Default style string using an abbreviated font
                                    % descriptor to specify 18pt text using an Arial font

/RV (<?xml version="1.0"?><body xmlns="http://www.w3.org/1999/xtml"
        xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/"
        xfa:contentType="text/html" xfa:APIVersion="Acrobat:8.0.0" xfa:spec="2.4">
        <p style="text-align:left">
            <b>
                <i>
                    Here is some bold italic text
                </i>
            </b>
        </p>
        <p style= "font-size:16pt">
        This text uses default text state parameters but changes the font size to 16.
    </p>
</body> )
```

12.7.4 Field Types

12.7.4.1 General

Interactive forms support the following field types:

- *Button fields* represent interactive controls on the screen that the user can manipulate with the mouse. They include *pushbuttons*, *check boxes*, and *radio buttons*.
- *Text fields* are boxes or spaces in which the user can enter text from the keyboard.
- *Choice fields* contain several text items, at most one of which may be selected as the field value. They include *scrollable list boxes* and *combo boxes*.
- *Signature fields* represent digital signatures and optional data for authenticating the name of the signer and the document's contents.

The following sub-clauses describe each of these field types in detail. Further types may be added in the future.

12.7.4.2 Button Fields

12.7.4.2.1 General

A *button field* (field type **Btn**) represents an interactive control on the screen that the user can manipulate with the mouse. There are three types of button fields:

- A *pushbutton* is a purely interactive control that responds immediately to user input without retaining a permanent value (see 12.7.4.2.2, "Pushbuttons").
- A *check box* toggles between two states, on and off (see 12.7.4.2.3, "Check Boxes").
- *Radio button fields* contain a set of related buttons that can each be on or off. Typically, at most one radio button in a set may be on at any given time, and selecting any one of the buttons automatically deselects all the others. (There are exceptions to this rule, as noted in "Radio Buttons.")

For button fields, bits 15, 16, 17, and 26 shall indicate the intended behaviour of the button field. A conforming reader shall follow the intended behaviour, as defined in Table 226 and clauses 12.7.4.2.2, "Pushbuttons", 12.7.4.2.3, "Check Boxes" and 12.7.4.2.4, "Radio Buttons".

Table 226 – Field flags specific to button fields

Bit position	Name	Meaning
15	NoToggleToOff	*(Radio buttons only)* If set, exactly one radio button shall be selected at all times; selecting the currently selected button has no effect. If clear, clicking the selected button deselects it, leaving no button selected.
16	Radio	If set, the field is a set of radio buttons; if clear, the field is a check box. This flag may be set only if the Pushbutton flag is clear.
17	Pushbutton	If set, the field is a pushbutton that does not retain a permanent value.
26	RadiosInUnison	*(PDF 1.5)* If set, a group of radio buttons within a radio button field that use the same value for the on state will turn on and off in unison; that is if one is checked, they are all checked. If clear, the buttons are mutually exclusive (the same behavior as HTML radio buttons).

12.7.4.2.2 Pushbuttons

A *pushbutton field* shall have a field type of **Btn** and the Pushbutton flag (see Table 226) set to one. Because this type of button retains no permanent value, it shall not use the **V** and **DV** entries in the field dictionary (see Table 220).

12.7.4.2.3 Check Boxes

A *check box field* represents one or more check boxes that toggle between two states, on and off, when manipulated by the user with the mouse or keyboard. Its field type shall be **Btn** and its Pushbutton and Radio flags (see Table 226) shall both be clear. Each state can have a separate appearance, which shall be defined by an appearance stream in the appearance dictionary of the field's widget annotation (see 12.5.5, "Appearance Streams"). The appearance for the off state is optional but, if present, shall be stored in the appearance dictionary under the name **Off**. **Yes** should be used as the name for the on state.

The **V** entry in the field dictionary (see Table 220) holds a name object representing the check box's appearance state, which shall be used to select the appropriate appearance from the appearance dictionary.

EXAMPLE 1 This example shows a typical check box definition.

```
1 0 obj
    << /FT /Btn
        /T (Urgent)
        /V /Yes
        /AS /Yes
        /AP << /N << /Yes 2 0 R /Off  3 0 R>>
    >>
endobj

2 0 obj
    << /Resources 20 0 R
        /Length 104
    >>

stream
    q
        0 0 1 rg
        BT
            /ZaDb 12 Tf
            0 0 Td
            (8) Tj
        ET
    Q
endstream
endobj

3 0 obj
    << /Resources 20 0 R
        /Length 104
    >>

stream
    q
        0 0 1 rg
        BT
            /ZaDb 12 Tf
            0 0 Td
            (8) Tj
        ET
    Q
endstream
endobj
```

Beginning with PDF 1.4, the field dictionary for check boxes and radio buttons may contain an optional **Opt** entry (see Table 227). If present, the **Opt** entry shall be an array of text strings representing the export value of each annotation in the field. It may be used for the following purposes:

- To represent the export values of check box and radio button fields in non-Latin writing systems. Because name objects in the appearance dictionary are limited to **PDFDocEncoding**, they cannot represent non-Latin text.
- To allow radio buttons or check boxes to be checked independently, even if they have the same export value.

EXAMPLE 2 A group of check boxes may be duplicated on more than one page such that the desired behavior is that when a user checks a box, the corresponding boxes on each of the other pages are also checked. In this case, each of the corresponding check boxes is a widget in the **Kids** array of a check box field.

NOTE For radio buttons, the same behavior shall occur only if the *RadiosInUnison* flag is set. If it is not set, at most one radio button in a field shall be set at a time.

Table 227 – Additional entry specific to check box and radio button fields

Key	Type	Value
Opt	array of text strings	*(Optional; inheritable; PDF 1.4)* An array containing one entry for each widget annotation in the **Kids** array of the radio button or check box field. Each entry shall be a text string representing the on state of the corresponding widget annotation. When this entry is present, the names used to represent the on state in the **AP** dictionary of each annotation (for example, /1, /2) numerical position (starting with **0**) of the annotation in the **Kids** array, encoded as a name object. This allows distinguishing between the annotations even if two or more of them have the same value in the **Opt** array.

12.7.4.2.4 Radio Buttons

A radio button field is a set of related buttons. Like check boxes, individual radio buttons have two states, on and off. A single radio button may not be turned off directly but only as a result of another button being turned on. Typically, a set of radio buttons (annotations that are children of a single radio button field) have at most one button in the on state at any given time; selecting any of the buttons automatically deselects all the others.

NOTE An exception occurs when multiple radio buttons in a field have the same on state and the RadiosInUnison flag is set. In that case, turning on one of the buttons turns on all of them.

The field type is **Btn**, the Pushbutton flag (see Table 226) is clear, and the Radio flag is set. This type of button field has an additional flag, NoToggleToOff, which specifies, if set, that exactly one of the radio buttons shall be selected at all times. In this case, clicking the currently selected button has no effect; if the NoToggleToOff flag is clear, clicking the selected button deselects it, leaving no button selected.

The **Kids** entry in the radio button field's field dictionary (see Table 220) holds an array of widget annotations representing the individual buttons in the set. The parent field's **V** entry holds a name object corresponding to the appearance state of whichever child field is currently in the on state; the default value for this entry is Off.

EXAMPLE This example shows the object definitions for a set of radio buttons.

```
10 0 obj                                       % Radio button field
   << /FT /Btn
      /Ff …                                    % …Radio flag = 1, Pushbutton = 0…
      /T (Credit card)
      /V /cardbrand1
      /Kids  [ 11 0 R
               12 0 R
             ]
```

```
    >>
endobj

11  0  obj                                        % First radio button
    <<  /Parent  10 0 R
        /AS  /cardbrand1
        /AP  <<  /N    <<  /cardbrand1  8 0 R
                           /Off  9 0 R
                       >>
            >>
    >>
endobj

12  0  obj                                        % Second radio button
    <<  /Parent  10 0 R
        /AS  /Off
        /AP  <<  /N    <<  /cardbrand2  8 0 R
                            /Off  9 0 R
                       >>
            >>
    >>
endobj

8  0  obj                                         % Appearance stream for “on” state
    <<  /Resources  20 0 R
        /Length  104
    >>

stream
    q
        0  0  1  rg
        BT
            /ZaDb  12  Tf
            0  0  Td
            (8)  Tj
        ET
    Q
endstream
endobj

9  0  obj                                         % Appearance stream for “off” state
    <<  /Resources  20 0 R
        /Length  104
    >>
stream
    q
        0  0  1  rg
        BT
            /ZaDb  12  Tf
            0  0  Td
            (4)  Tj
        ET
    Q
endstream
endobj
```

Like a check box field, a radio button field may use the optional **Opt** entry in the field dictionary *(PDF 1.4)* to define export values for its constituent radio buttons, using Unicode (UTF-16BE) encoding for non-Latin characters (see Table 227).

12.7.4.3 Text Fields

A *text field* (field type **Tx**) is a box or space for text fill-in data typically entered from a keyboard. The text may be restricted to a single line or may be permitted to span multiple lines, depending on the setting of the Multiline flag in the field dictionary’s **Ff** entry. Table 228 shows the flags pertaining to this type of field. A text field shall have a field type of **Tx**. A conforming PDF file, and a conforming processor shall obey the usage guidelines in Table 228.

Table 228 – Field flags specific to text fields

Bit position	Name	Meaning
13	Multiline	If set, the field may contain multiple lines of text; if clear, the field’s text shall be restricted to a single line.
14	Password	If set, the field is intended for entering a secure password that should not be echoed visibly to the screen. Characters typed from the keyboard shall instead be echoed in some unreadable form, such as asterisks or bullet characters. NOTE To protect password confidentiality, readers should never store the value of the text field in the PDF file if this flag is set.
21	FileSelect	*(PDF 1.4)* If set, the text entered in the field represents the pathname of a file whose contents shall be submitted as the value of the field.
23	DoNotSpellCheck	*(PDF 1.4)* If set, text entered in the field shall not be spell-checked.
24	DoNotScroll	*(PDF 1.4)* If set, the field shall not scroll (horizontally for single-line fields, vertically for multiple-line fields) to accommodate more text than fits within its annotation rectangle. Once the field is full, no further text shall be accepted for interactive form filling; for non-interactive form filling, the filler should take care not to add more character than will visibly fit in the defined area.
25	Comb	*(PDF 1.5)* May be set only if the **MaxLen** entry is present in the text field dictionary (see Table 229) and if the Multiline, Password, and FileSelect flags are clear. If set, the field shall be automatically divided into as many equally spaced positions, or *combs*, as the value of **MaxLen**, and the text is laid out into those combs.
26	RichText	*(PDF 1.5)* If set, the value of this field shall be a rich text string (see 12.7.3.4, “Rich Text Strings”). If the field has a value, the **RV** entry of the field dictionary (Table 222) shall specify the rich text string.

The field’s text shall be held in a text string (or, beginning with PDF 1.5, a stream) in the **V** (value) entry of the field dictionary. The contents of this text string or stream shall be used to construct an appearance stream for displaying the field, as described under 12.7.3.3, “Variable Text.” The text shall be presented in a single style (font, size, colour, and so forth), as specified by the **DA** (default appearance) string.

If the FileSelect flag *(PDF 1.4)* is set, the field shall function as a *file-select control*. In this case, the field’s text represents the pathname of a file whose contents shall be submitted as the field’s value:

- For fields submitted in HTML Form format, the submission shall use the MIME content type multipart/form-data, as described in Internet RFC 2045, *Multipurpose Internet Mail Extensions (MIME), Part One: Format of Internet Message Bodies* (see the Bibliography).
- For Forms Data Format (FDF) submission, the value of the **V** entry in the FDF field dictionary (see FDF Fields in 12.7.7.3, “FDF Catalog”) shall be a file specification (7.11, “File Specifications”) identifying the selected file.
- XML format is not supported for file-select controls; therefore, no value shall be submitted in this case.

Besides the usual entries common to all fields (see Table 220) and to fields containing variable text (see Table 222), the field dictionary for a text field may contain the additional entry shown in Table 229.

Table 229 – Additional entry specific to a text field

Key	Type	Value
MaxLen	integer	*(Optional; inheritable)* The maximum length of the field's text, in characters.

EXAMPLE The following example shows the object definitions for a typical text field.

```
6 0 obj
    << /FT /Tx
        /Ff …                                         % Set Multiline flag
        /T (Silly prose)
        /DA (0 0 1 rg /Ti 12 Tf)
        /V (The quick brown fox ate the lazy mouse)
        /AP << /N 5 0 R >>
    >>
endobj

5 0 obj
    << /Resources 21 0 R
        /Length 172
    >>

stream
    /Tx BMC
        q
            BT
                0 0 1 rg
                /Ti 12 Tf
                1 0 0 1 100 100 Tm
                0 0 Td
                (The quick brown fox ) Tj
                0 -13 Td
                (ate the lazy mouse.) Tj
            ET
        Q
    EMC
endstream
endobj
```

12.7.4.4 Choice Fields

A *choice field* shall have a field type of **Ch** that contains several text items, one or more of which shall be selected as the field value. The items may be presented to the user in one of the following two forms:

- A scrollable *list box*
- A *combo box* consisting of a drop-down list. The combo box may be accompanied by an editable text box in which the user can type a value other than the predefined choices, as directed by the value of the Edit bit in the **Ff** entry.

Table 230 – Field flags specific to choice fields

Bit position	Name	Meaning
18	Combo	If set, the field is a combo box; if clear, the field is a list box.
19	Edit	If set, the combo box shall include an editable text box as well as a drop-down list; if clear, it shall include only a drop-down list. This flag shall be used only if the Combo flag is set.

Table 230 – Field flags specific to choice fields (continued)

Bit position	Name	Meaning
20	Sort	If set, the field's option items shall be sorted alphabetically. This flag is intended for use by writers, not by readers. Conforming readers shall display the options in the order in which they occur in the **Opt** array (see Table 231).
22	MultiSelect	*(PDF 1.4)* If set, more than one of the field's option items may be selected simultaneously; if clear, at most one item shall be selected.
23	DoNotSpellCheck	*(PDF 1.4)* If set, text entered in the field shall not be spell-checked. This flag shall not be used unless the Combo and Edit flags are both set.
27	CommitOnSelChange	*(PDF 1.5)* If set, the new value shall be committed as soon as a selection is made (commonly with the pointing device). In this case, supplying a value for a field involves three actions: selecting the field for fill-in, selecting a choice for the fill-in value, and leaving that field, which finalizes or "commits" the data choice and triggers any actions associated with the entry or changing of this data. If this flag is on, then processing does not wait for leaving the field action to occur, but immediately proceeds to the third step. This option enables applications to perform an action once a selection is made, without requiring the user to exit the field. If clear, the new value is not committed until the user exits the field.

The various types of choice fields are distinguished by flags in the **Ff** entry, as shown in Table 230. Table 231 shows the field dictionary entries specific to choice fields.

Table 231 – Additional entries specific to a choice field

Key	Type	Value
Opt	array	*(Optional)* An array of options that shall be presented to the user. Each element of the array is either a text string representing one of the available options or an array consisting of two text strings: the option's export value and the text that shall be displayed as the name of the option. If this entry is not present, no choices should be presented to the user.
TI	integer	*(Optional)* For scrollable list boxes, the *top index* (the index in the **Opt** array of the first option visible in the list). Default value: 0.
I	array	*(Sometimes required, otherwise optional; PDF 1.4)* For choice fields that allow multiple selection (MultiSelect flag set), an array of integers, sorted in ascending order, representing the zero-based indices in the **Opt** array of the currently selected option items. This entry shall be used when two or more elements in the **Opt** array have different names but the same export value or when the value of the choice field is an array. This entry should not be used for choice fields that do not allow multiple selection. If the items identified by this entry differ from those in the **V** entry of the field dictionary (see discussion following this Table), the **V** entry shall be used.

The **Opt** array specifies the list of options in the choice field, each of which shall be represented by a text string that shall be displayed on the screen. Each element of the **Opt** array contains either this text string by itself or a two-element array, whose second element is the text string and whose first element is a text string representing the export value that shall be used when exporting interactive form field data from the document.

The field dictionary's **V** (value) entry (see Table 220) identifies the item or items currently selected in the choice field. If the field does not allow multiple selection—that is, if the MultiSelect flag *(PDF 1.4)* is not set—or if multiple selection is supported but only one item is currently selected, **V** is a text string representing the name of the selected item, as given in the field dictionary's **Opt** array. If multiple items are selected, **V** is an array of such strings. (For items represented in the **Opt** array by a two-element array, the name string is the second of the two array elements.) The default value of **V** is **null**, indicating that no item is currently selected.

EXAMPLE The following example shows a typical choice field definition.

```
<< /FT /Ch
   /Ff …
   /T (Body Color)
   /V (Blue)
   /Opt  [ (Red)
           (My favorite color)
           (Blue)
           ]
>>
```

12.7.4.5 Signature Fields

A *signature field (PDF 1.3)* is a form field that contains a digital signature (see 12.8, "Digital Signatures"). The field dictionary representing a signature field may contain the additional entries listed in Table 232, as well as the standard entries described in Table 220. The field type (**FT**) shall be **Sig**, and the field value (**V**), if present, shall be a *signature dictionary* containing the signature and specifying various attributes of the signature field (see Table 252).

NOTE 1 This signature form field serves two primary purposes. The first is to define the form field that will provide the visual signing properties for display but it also may hold information needed later when the actual signing takes place, such as the signature technology to use. This carries information from the author of the document to the software that later does the signing.

NOTE 2 Filling in (signing) the signature field entails updating at least the **V** entry and usually also the **AP** entry of the associated widget annotation. Exporting a signature field typically exports the **T**, **V**, and **AP** entries.

Like any other field, a signature field may be described by a widget annotation dictionary containing entries pertaining to an annotation as well as a field (see 12.5.6.19, "Widget Annotations"). The annotation rectangle (**Rect**) in such a dictionary shall give the position of the field on its page. Signature fields that are not intended to be visible shall have an annotation rectangle that has zero height and width. Conforming readers shall treat such signatures as not visible. Conforming readers shall also treat signatures as not visible if either the **Hidden** bit or the **NoView** bit of the **F** entry is **true**. The **F** entry is described in Table 164, and annotation flags are described in Table 165.

The appearance dictionary (**AP**) of a signature field's widget annotation defines the field's visual appearance on the page (see 12.5.5, "Appearance Streams").

Table 232 – Additional entries specific to a signature field

Key	**Type**	**Value**
Lock	dictionary	*(Optional; shall be an indirect reference; PDF 1.5)* A *signature field lock dictionary* that specifies a set of form fields that shall be locked when this signature field is signed. Table 233 lists the entries in this dictionary.
SV	dictionary	*(Optional; shall be an indirect reference; PDF 1.5)* A *seed value dictionary* (see Table 234) containing information that constrains the properties of a signature that is applied to this field.

The signature field lock dictionary (described in Table 233) contains field names from the signature seed value dictionary (described in Table 234) that cannot be changed through the user interface of a conforming reader.

Table 233 – Entries in a signature field lock dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SigFieldLock** for a signature field lock dictionary.
Action	name	*(Required)* A name which, in conjunction with **Fields**, indicates the set of fields that should be locked. The value shall be one of the following: All — All fields in the document Include — All fields specified in **Fields** Exclude — All fields except those specified in **Fields**
Fields	array	*(Required if the value of* ***Action*** *is Include or Exclude)* An array of text strings containing field names.

The value of the **SV** entry in the field dictionary is a seed value dictionary whose entries (see Table 234) provide constraining information that shall be used at the time the signature is applied. Its **Ff** entry specifies whether the other entries in the dictionary shall be honoured or whether they are merely recommendations.

The seed value dictionary may include seed values for private entries belonging to multiple handlers. A given handler shall use only those entries that are pertinent to itself and ignore the others.

Table 234 – Entries in a signature field seed value dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SV** for a seed value dictionary.
Ff	integer	*(Optional)* A set of bit flags specifying the interpretation of specific entries in this dictionary. A value of 1 for the flag indicates that the associated entry is a required constraint. A value of 0 indicates that the associated entry is an optional constraint. Bit positions are 1 (**Filter**); 2 (**SubFilter**); 3 (**V**); 4 (**Reasons**); 5 (**LegalAttestation**); 6 (**AddRevInfo**); and 7 (**DigestMethod**). Default value: 0.
Filter	name	*(Optional)* The signature handler that shall be used to sign the signature field. Beginning with PDF 1.7, if **Filter** is specified and the **Ff** entry indicates this entry is a required constraint, then the signature handler specified by this entry shall be used when signing; otherwise, signing shall not take place. If **Ff** indicates that this is an optional constraint, this handler may be used if it is available. If it is not available, a different handler may be used instead.
SubFilter	array	*(Optional)* An array of names indicating encodings to use when signing. The first name in the array that matches an encoding supported by the signature handler shall be the encoding that is actually used for signing. If **SubFilter** is specified and the **Ff** entry indicates that this entry is a required constraint, then the first matching encodings shall be used when signing; otherwise, signing shall not take place. If **Ff** indicates that this is an optional constraint, then the first matching encoding shall be used if it is available. If none is available, a different encoding may be used.
DigestMethod	array	(*Optional; PDF 1.7*) An array of names indicating acceptable digest algorithms to use while signing. The value shall be one of **SHA1**, **SHA256**, **SHA384**, **SHA512** and **RIPEMD160**. The default value is implementation-specific. This property is only applicable if the digital credential signing contains RSA public/private keys. If it contains DSA public/ private keys, the digest algorithm is always ***SHA1*** and this attribute shall be ignored.

Table 234 – Entries in a signature field seed value dictionary (continued)

Key	Type	Value
V	real	*(Optional)* The minimum required capability of the signature field seed value dictionary parser. A value of 1 specifies that the parser shall be able to recognize all seed value dictionary entries in a PDF 1.5 file. A value of 2 specifies that it shall be able to recognize all seed value dictionary entries specified. The **Ff** entry indicates whether this shall be treated as a required constraint.
Cert	dictionary	*(Optional)* A *certificate seed value dictionary* (see Table 235) containing information about the characteristics of the certificate that shall be used when signing.
Reasons	array	*(Optional)* An array of text strings that specifying possible reasons for signing a document. If specified, the reasons supplied in this entry replace those used by conforming products. If the **Reasons** array is provided and the **Ff** entry indicates that **Reasons** is a required constraint, one of the reasons in the array shall be used for the signature dictionary; otherwise, signing shall not take place. If the **Ff** entry indicates **Reasons** is an optional constraint, one of the reasons in the array may be chosen or a custom reason can be provided. If the **Reasons** array is omitted or contains a single entry with the value PERIOD (2Eh) and the **Ff** entry indicates that **Reasons** is a required constraint, the **Reason** entry shall be omitted from the signature dictionary (see Table 252).
MDP	dictionary	*(Optional; PDF 1.6)* A dictionary containing a single entry whose key is **P** and whose value is an integer between 0 and 3. A value of 0 defines the signature as an author signature (see 12.8, “Digital Signatures”). The values 1 through 3 shall be used for certification signatures and correspond to the value of **P** in a **DocMDP** transform parameters dictionary (see Table 254). If this **MDP** key is not present or the **MDP** dictionary does not contain a **P** entry, no rules shall be defined regarding the type of signature or its permissions.
TimeStamp	dictionary	*(Optional; PDF 1.6)* A time stamp dictionary containing two entries: **URL** An ASCII string specifying the URL of a time-stamping server, providing a time stamp that is compliant with RFC 3161, *Internet X.509 Public Key Infrastructure Time-Stamp Protocol* (see the Bibliography). **Ff** An integer whose value is 1 (the signature shall have a time stamp) or 0 (the signature need not have a time stamp). Default value: 0. NOTE Please see 12.8.3.3, "PKCS#7 Signatures as used in ISO 32000" for more details about hashing.
LegalAttestation	array	*(Optional; PDF 1.6)* An array of text strings specifying possible legal attestations (see 12.8.5, “Legal Content Attestations”). The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.

Table 234 – Entries in a signature field seed value dictionary (continued)

Key	Type	Value
AddRevInfo	boolean	(*Optional; PDF 1.7*) A flag indicating whether revocation checking shall be carried out. If **AddRevInfo** is **true**, the conforming processor shall perform the following additional tasks when signing the signature field: Perform revocation checking of the certificate (and the corresponding issuing certificates) used to sign. Include the revocation information within the signature value. Three **SubFilter** values have been defined for ISO 32000. For those values the **AddRevInfo** value shall be **true** only if **SubFilter** is **adbe.pkcs7.detached** or **adbe.pkcs7.sha1**. If **SubFilter** is **x509.rsa_sha1**, this entry shall be omitted or set to **false**. Additional SubFilters may be defined that also use **AddRevInfo** values. If **AddRevInfo** is **true** and the **Ff** entry indicates this is a required constraint, then the preceding tasks shall be performed. If they cannot be performed, then signing shall fail. Default value: **false** NOTE 1 Revocation information is carried in the signature data as specified by PCKS#7. See 12.8.3.3, "PKCS#7 Signatures as used in ISO 32000". NOTE 2 The trust anchors used are determined by the signature handlers for both creation and validation.

For optional keys that are not present, no constraint shall be placed upon the signature handler for that property when signing.

Table 235 – Entries in a certificate seed value dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SVCert** for a certificate seed value dictionary.
Ff	integer	*(Optional)* A set of bit flags specifying the interpretation of specific entries in this dictionary. A value of 1 for the flag means that a signer shall be required to use only the specified values for the entry. A value of 0 means that other values are permissible. Bit positions are 1 (**Subject**); 2 (**Issuer**); 3 (**OID**); 4 (**SubjectDN**); 5 (**Reserved**); 6 (**KeyUsage**); 7 (**URL**). Default value: 0.
Subject	array	*(Optional)* An array of byte strings containing DER-encoded X.509v3 certificates that are acceptable for signing. X.509v3 certificates are described in RFC 3280, *Internet X.509 Public Key Infrastructure, Certificate and Certificate Revocation List (CRL) Profile* (see the Bibliography). The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.

Table 235 – Entries in a certificate seed value dictionary (continued)

Key	Type	Value
SubjectDN	array of dictionaries	(*Optional; PDF 1.7*) An array of dictionaries, each specifying a Subject Distinguished Name (DN) that shall be present within the certificate for it to be acceptable for signing. The certificate ultimately used for the digital signature shall contain all the attributes specified in each of the dictionaries in this array. (PDF keys and values are mapped to certificate attributes and values.) The certificate is not constrained to use only attribute entries from these dictionaries but may contain additional attributes.The Subject Distinguished Name is described in RFC 3280 (see the Bibliography). The key can be any legal attribute identifier (OID). Attribute names shall contain characters in the set a-z A-Z 0-9 and PERIOD. Certificate attribute names are used as key names in the dictionaries in this array. Values of the attributes are used as values of the keys. Values shall be text strings. The value of the corresponding flag in the **Ff** entry indicates whether this entry is a required constraint.
KeyUsage	array of ASCII strings	(*Optional; PDF 1.7*) An array of ASCII strings, where each string specifies an acceptable key-usage extension that shall be present in the signing certificate. Multiple strings specify a range of acceptable key-usage extensions. The key-usage extension is described in RFC 3280. Each character in a string represents a key-usage type, where the order of the characters indicates the key-usage extension it represents. The first through ninth characters in the string, from left to right, represent the required value for the following key-usage extensions: 1 digitalSignature 4 dataEncipherment 7 cRLSign 2 non-Repudiation 5 keyAgreement 8 encipherOnly 3 keyEncipherment 6 keyCertSign 9 decipherOnly Any additional characters shall be ignored. Any missing characters or characters that are not one of the following values, shall be treated as 'X'. The following character values shall be supported: 0 Corresponding key-usage shall not be set. 1 Corresponding key-usage shall be set. X State of the corresponding key-usage does not matter. EXAMPLE 1 The string values '1' and '1XXXXXXXX' represent settings where the key-usage type digitalSignature is set and the state of all other key-usage types do not matter. The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.
Issuer	array	*(Optional)* An array of byte strings containing DER-encoded X.509v3 certificates of acceptable issuers. If the signer's certificate refers to any of the specified issuers (either directly or indirectly), the certificate shall be considered acceptable for signing. The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint. This array may contain self-signed certificates.
OID	array	*(Optional)* An array of byte strings that contain Object Identifiers (OIDs) of the certificate policies that shall be present in the signing certificate. EXAMPLE 2 An example of such a string is: (2.16.840.1.113733.1.7.1.1). This field shall only be used if the value of **Issuer** is not empty. The certificate policies extension is described in RFC 3280 (see the Bibliography). The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.

Table 235 – Entries in a certificate seed value dictionary (continued)

Key	Type	Value
URL	ASCII string	*(Optional)* A URL, the use for which shall be defined by the **URLType** entry.
URLType	Name	(*Optional; PDF 1.7*) A name indicating the usage of the **URL** entry. There are standard uses and there can be implementation-specific uses for this URL. The following value specifies a valid standard usage: **Browser** – The URL references content that shall be displayed in a web browser to allow enrolling for a new credential if a matching credential is not found. The **Ff** attribute's URL bit shall be ignored for this usage. Third parties may extend the use of this attribute with their own attribute values, which shall conform to the guidelines described in Annex E. The default value is **Browser**.

12.7.5 Form Actions

12.7.5.1 General

Interactive forms also support special types of actions in addition to those described in 12.6.4, "Action Types":

- *submit-form action*
- *reset-form action*
- *import-data action*

12.7.5.2 Submit-Form Action

Upon invocation of a *submit-form action*, a conforming processor shall transmit the names and values of selected interactive form fields to a specified uniform resource locator (URL),

NOTE Presumably, the URL is the address of a Web server that will process them and send back a response.

The value of the action dictionary's **Flags** entry shall be an non-negative integer containing flags specifying various characteristics of the action. Bit positions within the flag word shall be numbered starting with 1 (low-order). Table 237 shows the meanings of the flags; all undefined flag bits shall be reserved and shall be set to 0.

Table 236 – Additional entries specific to a submit-form action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **SubmitForm** for a submit-form action.
F	file specification	*(Required)* A URL file specification (see 7.11.5, "URL Specifications") giving the uniform resource locator (URL) of the script at the Web server that will process the submission.

Table 236 – Additional entries specific to a submit-form action (continued)

Key	Type	Value
Fields	array	*(Optional)* An array identifying which fields to include in the submission or which to exclude, depending on the setting of the Include/Exclude flag in the **Flags** entry (see Table 237). Each element of the array shall be either an indirect reference to a field dictionary or *(PDF 1.3)* a text string representing the fully qualified name of a field. Elements of both kinds may be mixed in the same array. If this entry is omitted, the Include/Exclude flag shall be ignored, and all fields in the document's interactive form shall be submitted except those whose NoExport flag (see Table 221) is set. Fields with no values may also be excluded, as dictated by the value of the IncludeNoValueFields flag; see Table 237.
Flags	integer	*(Optional; inheritable)* A set of flags specifying various characteristics of the action (see Table 237). Default value: 0.

Table 237 – Flags for submit-form actions

Bit position	Name	Meaning
1	Include/Exclude	If clear, the **Fields** array (see Table 236) specifies which fields to include in the submission. (All descendants of the specified fields in the field hierarchy shall be submitted as well.) If set, the **Fields** array tells which fields to exclude. All fields in the document's interactive form shall be submitted *except* those listed in the **Fields** array and those whose NoExport flag (see Table 221) is set and fields with no values if the IncludeNoValueFields flag is clear.
2	IncludeNoValueFields	If set, all fields designated by the **Fields** array and the Include/Exclude flag shall be submitted, regardless of whether they have a value (**V** entry in the field dictionary). For fields without a value, only the field name shall be transmitted. If clear, fields without a value shall not be submitted.
3	ExportFormat	Meaningful only if the SubmitPDF and XFDF flags are clear. If set, field names and values shall be submitted in HTML Form format. If clear, they shall be submitted in Forms Data Format (FDF); see 12.7.7, "Forms Data Format."
4	GetMethod	If set, field names and values shall be submitted using an HTTP GET request. If clear, they shall be submitted using a POST request. This flag is meaningful only when the ExportFormat flag is set; if ExportFormat is clear, this flag shall also be clear.

Table 237 – Flags for submit-form actions (continued)

Bit position	Name	Meaning
5	SubmitCoordinates	If set, the coordinates of the mouse click that caused the submit-form action shall be transmitted as part of the form data. The coordinate values are relative to the upper-left corner of the field's widget annotation rectangle. They shall be represented in the data in the format *name*.x=*xval*&*name*.y=*yval* where *name* is the field's mapping name (**TM** in the field dictionary) if present; otherwise, *name* is the field name. If the value of the **TM** entry is a single ASCII SPACE (20h) character, both the name and the ASCII PERIOD (2Eh) following it shall be suppressed, resulting in the format x=*xval*&y=*yval* This flag shall be used only when the ExportFormat flag is set. If ExportFormat is clear, this flag shall also be clear.
6	XFDF	*(PDF 1.4)* shall be used only if the SubmitPDF flags are clear. If set, field names and values shall be submitted as XFDF.
7	IncludeAppendSaves	*(PDF 1.4)* shall be used only when the form is being submitted in Forms Data Format (that is, when both the XFDF and ExportFormat flags are clear). If set, the submitted FDF file shall include the contents of all incremental updates to the underlying PDF document, as contained in the **Differences** entry in the FDF dictionary (see Table 243). If clear, the incremental updates shall not be included.
8	IncludeAnnotations	*(PDF 1.4)* shall be used only when the form is being submitted in Forms Data Format (that is, when both the XFDF and ExportFormat flags are clear). If set, the submitted FDF file shall include includes all markup annotations in the underlying PDF document (see 12.5.6.2, "Markup Annotations"). If clear, markup annotations shall not be included.
9	SubmitPDF	*(PDF 1.4)* If set, the document shall be submitted as PDF, using the MIME content type application/pdf (described in Internet RFC 2045, *Multipurpose Internet Mail Extensions (MIME), Part One: Format of Internet Message Bodies*; see the Bibliography). If set, all other flags shall be ignored except GetMethod.
10	CanonicalFormat	*(PDF 1.4)* If set, any submitted field values representing dates shall be converted to the standard format described in 7.9.4, "Dates." NOTE 1 The interpretation of a form field as a date is not specified explicitly in the field itself but only in the JavaScript code that processes it.

Table 237 – Flags for submit-form actions (continued)

Bit position	Name	Meaning
11	ExclNonUserAnnots	*(PDF 1.4)* shall be used only when the form is being submitted in Forms Data Format (that is, when both the XFDF and ExportFormat flags are clear) and the IncludeAnnotations flag is set. If set, it shall include only those markup annotations whose **T** entry (see Table 170) matches the name of the current user, as determined by the remote server to which the form is being submitted. NOTE 2 The **T** entry for markup annotations specifies the text label that is displayed in the title bar of the annotation's pop-up window and is assumed to represent the name of the user authoring the annotation. NOTE 3 This allows multiple users to collaborate in annotating a single remote PDF document without affecting one another's annotations.
12	ExclFKey	*(PDF 1.4)* shall be used only when the form is being submitted in Forms Data Format (that is, when both the XFDF and ExportFormat flags are clear). If set, the submitted FDF shall exclude the **F** entry.
14	EmbedForm	*(PDF 1.5)* shall be used only when the form is being submitted in Forms Data Format (that is, when both the XFDF and ExportFormat flags are clear). If set, the **F** entry of the submitted FDF shall be a file specification containing an embedded file stream representing the PDF file from which the FDF is being submitted.

The set of fields whose names and values are to be submitted shall be defined by the **Fields** array in the action dictionary (Table 236) together with the Include/Exclude and IncludeNoValueFields flags in the **Flags** entry (Table 237). Each element of the **Fields** array shall identify an interactive form field, either by an indirect reference to its field dictionary or *(PDF 1.3)* by its fully qualified field name (see 12.7.3.2, "Field Names"). If the Include/Exclude flag is clear, the submission consists of all fields listed in the **Fields** array, along with any descendants of those fields in the field hierarchy. If the Include/Exclude flag is set, the submission shall consist of all fields in the document's interactive form *except* those listed in the **Fields** array.

The NoExport flag in the field dictionary's **Ff** entry (see Table 220 and Table 221) takes precedence over the action's **Fields** array and Include/Exclude flag. Fields whose NoExport flag is set shall not be included in a submit-form action.

Field names and values may be submitted in any of the following formats, depending on the settings of the action's ExportFormat, SubmitPDF, and XFDF flags (see the Bibliography for references):

- HTML Form format (described in the *HTML 4.01 Specification*)
- Forms Data Format (FDF), which is described in 12.7.7, "Forms Data Format."
- XFDF, a version of FDF based on XML. XFDF is described in the Adobe technical note *XML Forms Data Format Specification, Version 2.0.* XML is described in the W3C document *Extensible Markup Language (XML) 1.1*)
- PDF (in this case, the entire document shall be submitted rather than individual fields and values).

The name submitted for each field shall be its fully qualified name (see 12.7.3.2, "Field Names"), and the value shall be specified by the **V** entry in its field dictionary.

For pushbutton fields submitted in FDF, the value submitted shall be that of the **AP** entry in the field's widget annotation dictionary. If the submit-form action dictionary contains no **Fields** entry, such pushbutton fields shall not be submitted.

Fields with no value (that is, whose field dictionary does not contain a **V** entry) are ordinarily not included in the submission. The submit-form action's IncludeNoValueFields flag may override this behaviour. If this flag is set, such valueless fields shall be included in the submission by name only, with no associated value.

12.7.5.3 Reset-Form Action

Upon invocation of a *reset-form action*, a conforming processor shall reset selected interactive form fields to their default values; that is, it shall set the value of the **V** entry in the field dictionary to that of the **DV** entry (see Table 220). If no default value is defined for a field, its **V** entry shall be removed. For fields that can have no value (such as pushbuttons), the action has no effect. Table 238 shows the action dictionary entries specific to this type of action.

The value of the action dictionary's **Flags** entry is a non-negative containing flags specifying various characteristics of the action. Bit positions within the flag word shall be numbered starting from 1 (low-order). Only one flag is defined for this type of action. All undefined flag bits shall be reserved and shall be set to 0.

Table 238 – Additional entries specific to a reset-form action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **ResetForm** for a reset-form action.
Fields	array	*(Optional)* An array identifying which fields to reset or which to exclude from resetting, depending on the setting of the Include/Exclude flag in the **Flags** entry (see Table 239). Each element of the array shall be either an indirect reference to a field dictionary or *(PDF 1.3)* a text string representing the fully qualified name of a field. Elements of both kinds may be mixed in the same array. If this entry is omitted, the Include/Exclude flag shall be ignored; all fields in the document's interactive form are reset.
Flags	integer	*(Optional; inheritable)* A set of flags specifying various characteristics of the action (see Table 239). Default value: 0.

Table 239 – Flag for reset-form actions

Bit position	Name	Meaning
1	Include/Exclude	If clear, the **Fields** array (see Table 238) specifies which fields to reset. (All descendants of the specified fields in the field hierarchy are reset as well.) If set, the **Fields** array indicates which fields to exclude from resetting; that is, all fields in the document's interactive form shall be reset *except* those listed in the **Fields** array.

12.7.5.4 Import-Data Action

Upon invocation of an *import-data action*, a conforming processor shall import Forms Data Format (FDF) data into the document's interactive form from a specified file.

Table 240 – Additional entries specific to an import-data action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be **ImportData** for an import-data action.
F	file specification	*(Required)* The FDF file from which to import the data.

12.7.6 Named Pages

The optional **Pages** entry *(PDF 1.3)* in a document's name dictionary (see 7.7.4, "Name Dictionary") contains a name tree that maps name strings to individual pages within the document. Naming a page allows it to be referenced in two different ways:

- An import-data action can add the named page to the document into which FDF is being imported, either as a page or as a button appearance.
- A script executed by a JavaScript action can add the named page to the current document as a regular page.

A named page that is intended to be visible to a user shall be left in the page tree (see 7.7.3, "Page Tree"), and there shall be a reference to it in the appropriate leaf node of the name dictionary's **Pages** tree. If the page is not intended to be displayed by the reader, it shall be referenced from the name dictionary's **Templates** tree instead. Such invisible pages shall have an object type of **Template** rather than **Page** and shall have no **Parent** or **B** entry (see Table 30).

12.7.7 Forms Data Format

12.7.7.1 General

This sub-clause describes Forms Data Format (FDF), the file format used for interactive form data *(PDF 1.2)*. FDF can be used when submitting form data to a server, receiving the response, and incorporating it into the interactive form. It can also be used to export form data to stand-alone files that can be stored, transmitted electronically, and imported back into the corresponding PDF interactive form. In addition, beginning in PDF 1.3, FDF can be used to define a container for annotations that are separate from the PDF document to which they apply.

FDF is based on PDF; it uses the same syntax and has essentially the same file structure (7.5, "File Structure"). However, it differs from PDF in the following ways:

- The cross-reference table (7.5.4, "Cross-Reference Table") is optional.
- FDF files shall not be updated (see 7.5.6, "Incremental Updates"). Objects shall only be of generation 0, and no two objects within an FDF file shall have the same object number.
- The document structure is much simpler than PDF, since the body of an FDF document consists of only one required object.
- The length of a stream shall not be specified by an indirect object.

FDF uses the MIME content type application/vnd.fdf. On the Windows and UNIX platforms, FDF files have the extension .fdf; on Mac OS, they have file type 'FDF '.

12.7.7.2 FDF File Structure

12.7.7.2.1 General

An FDF file shall be structured in essentially the same way as a PDF file but contains only those elements required for the export and import of interactive form and annotation data. It consists of three required elements and one optional element (see Figure 65):

- A one-line *header* identifying the version number of the PDF specification to which the file conforms
- A *body* containing the objects that make up the content of the file
- An optional *cross-reference table* containing information about the indirect objects in the file
- An optional *trailer* giving the location of the cross-reference table and of certain special objects within the body of the file

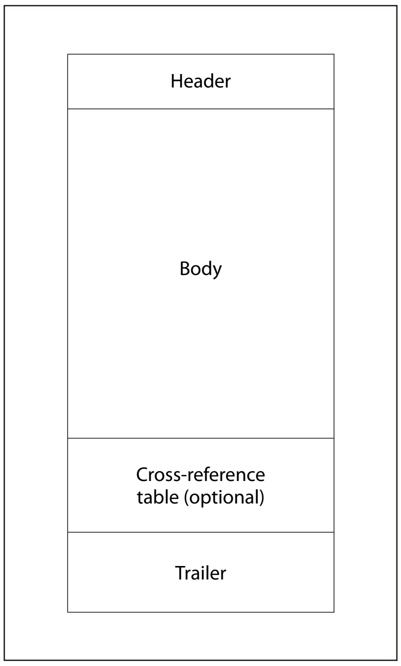


Figure 65 – FDF file structure

12.7.7.2.2 FDF Header

The first line of an FDF file shall be a *header*, which shall contain

```
%FDF-1.2
```

The version number is given by the **Version** entry in the FDF catalogue dictionary (see 12.7.7.3, “FDF Catalog”).

12.7.7.2.3 FDF Body

The *body* of an FDF file shall consist of a sequence of indirect objects representing the file’s catalogue (see 12.7.7.3, “FDF Catalog”) and any additional objects that the catalogue references. The objects are of the same basic types described in 7.5, “File Structure” (other than the %PDF–n.m and %%EOF comments described in 7.5, “File Structure”) have no semantics. They are not necessarily preserved by applications that edit PDF files.” Just as in PDF, objects in FDF can be direct or indirect.

12.7.7.2.4 FDF Trailer

The *trailer* of an FDF file enables a reader to find significant objects quickly within the body of the file. The last line of the file contains only the end-of-file marker, %%EOF. This marker shall be preceded by the *FDF trailer dictionary*, consisting of the keyword **trailer** followed by a series of one or more key-value pairs enclosed in double angle brackets (<<…>>) (using LESS-THAN SIGNs (3Ch) and GREATER-THAN SIGNs (3Eh)). The only required key is **Root**, whose value is an indirect reference to the file's catalogue dictionary (see Table 242). The trailer may optionally contain additional entries for objects that are referenced from within the catalogue.

Table 241 – Entry in the FDF trailer dictionary

Key	Type	Value
Root	dictionary	*(Required; shall be an indirect reference)* The **Catalog** object for this FDF file (see 12.7.7.3, "FDF Catalog").

Thus, the trailer has the overall structure

trailer
<< /Root c 0 R
key_2 $value_2$
…
key_n $value_n$
>>
%%EOF

where *c* is the object number of the file's catalogue dictionary.

12.7.7.3 FDF Catalog

12.7.7.3.1 General

The root node of an FDF file's object hierarchy is the **Catalog** dictionary, located by means of the **Root** entry in the file's trailer dictionary (see FDF Trailer in 12.7.7.2, "FDF File Structure"). As shown in Table 241, the only required entry in the catalogue is **FDF**; its value shall be an *FDF dictionary* (Table 243), which in turn shall contain references to other objects describing the file's contents. The catalogue may also contain an optional **Version** entry identifying the version of the PDF specification to which this FDF file conforms.

Table 242 – Entries in the FDF catalog dictionary

Key	Type	Value
Version	name	*(Optional; PDF 1.4)* The version of the FDF specification to which this FDF file conforms (for example, 1.4) if later than the version specified in the file's header (see FDF Header in 12.7.7.2, "FDF File Structure"). If the header specifies a later version, or if this entry is absent, the document conforms to the version specified in the header. The value of this entry is a name object, not a number, and therefore shall be preceded by a slash character (/) when written in the FDF file (for example, */1.4*).
FDF	dictionary	*(Required)* The FDF dictionary for this file (see Table 243).

Table 243 – Entries in the FDF dictionary

Key	Type	Value
F	file specification	*(Optional)* The *source file* or *target file*: the PDF document file that this FDF file was exported from or is intended to be imported into.
ID	array	*(Optional)* An array of two byte strings constituting a file identifier (see 14.4, “File Identifiers”) for the source or target file designated by **F**, taken from the **ID** entry in the file’s trailer dictionary (see 7.5.5, “File Trailer”).
Fields	array	*(Optional)* An array of FDF field dictionaries (see FDF Fields in 12.7.7.3, “FDF Catalog”) describing the root fields (those with no ancestors in the field hierarchy) that shall be exported or imported. This entry and the **Pages** entry shall not both be present.
Status	PDFDocEncoded string	*(Optional)* A status string that shall be displayed indicating the result of an action, typically a submit-form action (see 12.7.5.2, “Submit-Form Action”). The string shall be encoded with **PDFDocEncoding**. This entry and the **Pages** entry shall not both be present.
Pages	array	*(Optional; PDF 1.3)* An array of FDF page dictionaries (see FDF Pages in 12.7.7.3, “FDF Catalog”) describing pages that shall be added to a PDF target document. The **Fields** and **Status** entries shall not be present together with this entry.
Encoding	name	*(Optional; PDF 1.3)* The encoding that shall be used for any FDF field value or option (**V** or **Opt** in the field dictionary; see Table 246) or field name that is a string and does not begin with the Unicode prefix U+FEFF. Default value: **PDFDocEncoding**. Other allowed values include **Shift_JIS**, **BigFive**, **GBK**, **UHC**, **utf_8**, **utf_16**
Annots	array	*(Optional; PDF 1.3)* An array of FDF annotation dictionaries (see FDF Annotation Dictionaries in 12.7.7.3, “FDF Catalog”). The array may include annotations of any of the standard types listed in Table 169 except **Link**, **Movie**, **Widget**, **PrinterMark**, **Screen**, and **TrapNet**.

Table 243 – Entries in the FDF dictionary (continued)

Key	Type	Value
Differences	stream	*(Optional; PDF 1.4)* A stream containing all the bytes in all incremental updates made to the underlying PDF document since it was opened (see 7.5.6, “Incremental Updates”). If a submit-form action submitting the document to a remote server as FDF has its IncludeAppendSaves flag set (see 12.7.5.2, “Submit-Form Action”), the contents of this stream shall be included in the submission. This allows any digital signatures (see 12.8, “Digital Signatures”) to be transmitted to the server. An incremental update shall be automatically performed just before the submission takes place, in order to capture all changes made to the document. The submission shall include the full set of incremental updates back to the time the document was first opened, even if some of them may already have been included in intervening submissions. Although a ***Fields*** or ***Annots*** entry (or both) may be present along with ***Differences***, there is no requirement that their contents will be consistent with each other. In particular, if ***Differences*** contains a digital signature, only the values of the form fields given in the ***Differences*** stream shall be considered trustworthy under that signature.
Target	string	*(Optional; PDF 1.4)* The name of a browser frame in which the underlying PDF document shall be opened. This mimics the behaviour of the target attribute in HTML <href> tags.
EmbeddedFDFs	array	*(Optional; PDF 1.4)* An array of file specifications (see 7.11, “File Specifications”) representing other FDF files embedded within this one (7.11.4, “Embedded File Streams”).
JavaScript	dictionary	*(Optional; PDF 1.4)* A *JavaScript dictionary* (see Table 245) defining document-level JavaScript scripts.

Embedded FDF files specified in the FDF dictionary’s **EmbeddedFDFs** entry may be encrypted. Besides the usual entries for an embedded file stream, the stream dictionary representing such an encrypted FDF file shall contain the additional entry shown in Table 244 to identify the revision number of the FDF encryption algorithm used to encrypt the file. Although the FDF encryption mechanism is separate from the one for PDF file encryption described in 7.6, “Encryption,” revision 1 (the only one defined) uses a similar RC4 encryption algorithm based on a 40-bit encryption key. The key shall be computed by means of an MD5 hash, using a padded user-supplied password as input. The computation shall be identical to steps (a) and (b) of the "Algorithm 2: Computing an encryption key" in 7.6.3.3, "Encryption Key Algorithm"; the first 5 bytes of the result shall be the encryption key for the embedded FDF file.

Table 244 – Additional entry in an embedded file stream dictionary for an encrypted FDF file

Key	Type	Value
EncryptionRevision	integer	*(Required if the FDF file is encrypted; PDF 1.4)* The revision number of the FDF encryption algorithm used to encrypt the file. This value shall be defined at the time of publication is 1.

The **JavaScript** entry in the FDF dictionary holds a *JavaScript dictionary* containing JavaScript scripts that shall be defined globally at the document level, rather than associated with individual fields. The dictionary may contain scripts defining JavaScript functions for use by other scripts in the document, as well as scripts that shall be executed immediately before and after the FDF file is imported. Table 245 shows the contents of this dictionary.

Table 245 – Entries in the JavaScript dictionary

Key	Type	Value
Before	text string or text stream	*(Optional)* A text string or text stream containing a JavaScript script that shall be executed just before the FDF file is imported.
After	text string or text stream	*(Optional)* A text string or text stream containing a JavaScript script that shall be executed just after the FDF file is imported.
AfterPermsReady	text string or text stream	*(Optional; PDF 1.6)* A text string or text stream containing a JavaScript script that shall be executed after the FDF file is imported and the usage rights in the PDF document have been determined (see 12.8.2.3, "UR"). Verification of usage rights requires the entire file to be present, in which case execution of this script shall be deferred until that requirement is met.
Doc	array	*(Optional)* An array defining additional JavaScript scripts that shall be added to those defined in the **JavaScript** entry of the document's name dictionary (see 7.7.4, "Name Dictionary"). The array shall contain an even number of elements, organized in pairs. The first element of each pair shall be a name and the second shall be a text string or text stream defining the script corresponding to that name. Each of the defined scripts shall be added to those already defined in the name dictionary and shall then be executed before the script defined in the **Before** entry is executed. NOTE As described in 12.6.4.16, "JavaScript Actions," these scripts are used to define JavaScript functions for use by other scripts in the document.

12.7.7.3.2 FDF Fields

Each field in an FDF file shall be described by an *FDF field dictionary*. Table 246 shows the contents of this type of dictionary. Most of the entries have the same form and meaning as the corresponding entries in a field dictionary (Table 220, Table 222, Table 229, and Table 231) or a widget annotation dictionary (Table 168 and Table 188). Unless otherwise indicated in the table, importing a field causes the values of the entries in the FDF field dictionary to replace those of the corresponding entries in the field with the same fully qualified name in the target document.

Table 246 – Entries in an FDF field dictionary

Key	Type	Value
Kids	array	*(Optional)* An array containing the immediate children of this field. Unlike the children of fields in a PDF file, which shall be specified as indirect object references, those of an FDF field may be either direct or indirect objects.
T	text string	*(Required)* The partial field name (see 12.7.3.2, "Field Names").
V	(various)	*(Optional)* The field's value, whose format varies depending on the field type; see the descriptions of individual field types in 12.7.4, "Field Types" for further information.
Ff	integer	*(Optional)* A set of flags specifying various characteristics of the field (see Table 221, Table 226, Table 228, and Table 230). When imported into an interactive form, the value of this entry shall replace that of the **Ff** entry in the form's corresponding field dictionary. If this field is present, the **SetFf** and **ClrFf** entries, if any, shall be ignored.
SetFf	integer	*(Optional)* A set of flags to be set (turned on) in the **Ff** entry of the form's corresponding field dictionary. Bits equal to 1 in **SetFf** shall cause the corresponding bits in **Ff** to be set to 1. This entry shall be ignored if an **Ff** entry is present in the FDF field dictionary.

Table 246 – Entries in an FDF field dictionary (continued)

Key	Type	Value
ClrFf	integer	*(Optional)* A set of flags to be cleared (turned off) in the **Ff** entry of the form's corresponding field dictionary. Bits equal to 1 in **ClrFf** shall cause the corresponding bits in **Ff** to be set to 0. If a **SetFf** entry is also present in the FDF field dictionary, it shall be applied before this entry. This entry shall be ignored if an **Ff** entry is present in the FDF field dictionary.
F	integer	*(Optional)* A set of flags specifying various characteristics of the field's widget annotation (see 12.5.3, "Annotation Flags"). When imported into an interactive form, the value of this entry shall replace that of the **F** entry in the form's corresponding annotation dictionary. If this field is present, the **SetF** and **ClrF** entries, if any, shall be ignored.
SetF	integer	*(Optional)* A set of flags to be set (turned on) in the **F** entry of the form's corresponding widget annotation dictionary. Bits equal to 1 in **SetF** shall cause the corresponding bits in **F** to be set to 1. This entry shall be ignored if an **F** entry is present in the FDF field dictionary.
ClrF	integer	*(Optional)* A set of flags to be cleared (turned off) in the **F** entry of the form's corresponding widget annotation dictionary. Bits equal to 1 in **ClrF** shall cause the corresponding bits in **F** to be set to 0. If a **SetF** entry is also present in the FDF field dictionary, it shall be applied before this entry. This entry shall be ignored if an **F** entry is present in the FDF field dictionary.
AP	dictionary	*(Optional)* An appearance dictionary specifying the appearance of a pushbutton field (see Pushbuttons in 12.7.4.2, "Button Fields"). The appearance dictionary's contents are as shown in Table 168, except that the values of the **N**, **R**, and **D** entries shall all be streams.
APRef	dictionary	*(Optional; PDF 1.3)* A dictionary holding references to external PDF files containing the pages to use for the appearances of a pushbutton field. This dictionary is similar to an appearance dictionary (see Table 168), except that the values of the **N**, **R**, and **D** entries shall all be named page reference dictionaries (Table 250). This entry shall be ignored if an **AP** entry is present.
IF	dictionary	*(Optional; PDF 1.3; button fields only)* An icon fit dictionary (see Table 247) specifying how to display a button field's icon within the annotation rectangle of its widget annotation.
Opt	array	*(Required; choice fields only)* An array of options that shall be presented to the user. Each element of the array shall take one of two forms: A text string representing one of the available options A two-element array consisting of a text string representing one of the available options and a default appearance string for constructing the item's appearance dynamically at viewing time (see 12.7.3.3, "Variable Text").
A	dictionary	*(Optional)* An action that shall be performed when this field's widget annotation is activated (see 12.6, "Actions").
AA	dictionary	*(Optional)* An additional-actions dictionary defining the field's behaviour in response to various trigger events (see 12.6.3, "Trigger Events").
RV	text string or text stream	*(Optional; PDF 1.5)* A rich text string, as described in 12.7.3.4, "Rich Text Strings."

In an FDF field dictionary representing a button field, the optional **IF** entry holds an *icon fit dictionary (PDF 1.3)* specifying how to display the button's icon within the annotation rectangle of its widget annotation. Table 247 shows the contents of this type of dictionary.

Table 247 – Entries in an icon fit dictionary

Key	Type	Value
SW	name	*(Optional)* The circumstances under which the icon shall be scaled inside the annotation rectangle: A Always scale. B Scale only when the icon is bigger than the annotation rectangle. S Scale only when the icon is smaller than the annotation rectangle. N Never scale. Default value: A.
S	name	*(Optional)* The type of scaling that shall be used: A *Anamorphic scaling*: Scale the icon to fill the annotation rectangle exactly, without regard to its original aspect ratio (ratio of width to height). P *Proportional scaling*: Scale the icon to fit the width or height of the annotation rectangle while maintaining the icon's original aspect ratio. If the required horizontal and vertical scaling factors are different, use the smaller of the two, centering the icon within the annotation rectangle in the other dimension. Default value: P.
A	array	*(Optional)* An array of two numbers that shall be between 0.0 and 1.0 indicating the fraction of leftover space to allocate at the left and bottom of the icon. A value of [0.0 0.0] shall position the icon at the bottom-left corner of the annotation rectangle. A value of [0.5 0.5] shall center it within the rectangle. This entry shall be used only if the icon is scaled proportionally. Default value: [0.5 0.5].
FB	boolean	*(Optional; PDF 1.5)* If **true**, indicates that the button appearance shall be scaled to fit fully within the bounds of the annotation without taking into consideration the line width of the border. Default value: **false**.

12.7.7.3.3 FDF Pages

The optional **Pages** field in an FDF dictionary (see Table 243) shall contain an array of *FDF page dictionaries (PDF 1.3)* describing new pages that shall be added to the target document. Table 248 shows the contents of this type of dictionary.

Table 248 – Entries in an FDF page dictionary

Key	Type	Value
Templates	array	*(Required)* An array of *FDF template dictionaries* (see Table 249) that shall describe the named pages that serve as templates on the page.
Info	dictionary	*(Optional)* An *FDF page information dictionary* that shall contain additional information about the page.

An *FDF template dictionary* shall contain information describing a named page that serves as a template. Table 249 shows the contents of this type of dictionary.

Table 249 – Entries in an FDF template dictionary

Key	Type	Value
TRef	dictionary	*(Required)* A named page reference dictionary (see Table 250) that shall specify the location of the template.
Fields	array	*(Optional)* An array of references to FDF field dictionaries (see Table 246) describing the root fields that shall be imported (those with no ancestors in the field hierarchy).
Rename	boolean	*(Optional)* A flag that shall specify whether fields imported from the template shall be renamed in the event of name conflicts with existing fields; see the Note in this sub-clause for further discussion. Default value: **true**.

NOTE The names of fields imported from a template can sometimes conflict with those of existing fields in the target document. This can occur, for example, if the same template page is imported more than once or if two different templates have fields with the same names.

The **Rename** flag does not define a renaming algorithm (see Annex J).

The **TRef** entry in an FDF template dictionary shall hold a *named page reference dictionary* that shall describe the location of external templates or page elements. Table 250 shows the contents of this type of dictionary.

Table 250 – Entries in an FDF named page reference dictionary

Key	Type	Value
Name	string	*(Required)* The name of the referenced page.
F	file specification	*(Optional)* The file containing the named page. If this entry is absent, it shall be assumed that the page resides in the associated PDF file.

12.7.7.3.4 FDF Annotation Dictionaries

Each annotation dictionary in an FDF file shall have a **Page** entry (see Table 251) that shall indicate the page of the source document to which the annotation is attached.

Table 251 – Additional entry for annotation dictionaries in an FDF file

Key	Type	Value
Page	integer	*(Required for annotations in FDF files)* The ordinal page number on which this annotation shall appear, where page 0 is the first page.

12.7.8 XFA Forms

PDF 1.5 introduces support for interactive forms based on the Adobe XML Forms Architecture (XFA). The **XFA** entry in the interactive forms dictionary (see Table 218) specifies an *XFA resource*, which shall be an XML stream that contains the form information. The format of an XFA resource is described in the *XML Data Package (XDP) Specification* (see the Bibliography).

The **XFA** entry shall be either a stream containing the entire XFA resource or an array specifying individual *packets* that together make up the XFA resource. The resource includes but is not limited to the following information:

- The *form template* (specified in the template packet), which describes the characteristics of the form, including its fields, calculations, validations, and formatting. The *XML Template Specification* describes the architecture of a form template (see Bibliography).

- The *data* (specified in the datasets packet), which represents the state of the form
- The *configuration information* (specified in the config packet), which shall be used to properly process the form template and associated data. Configuration information shall be formatted as described in the *XML Configuration Specification* (see Bibliography).

A *packet* is a pair of a string and stream. The string contains the name of the XML element and the stream contains the complete text of this XML element. Each packet represents a complete XML element, with the exception of the first and last packet, which specify begin and end tags for the xdp:xdp element (see EXAMPLE 1 in this sub-clause).

EXAMPLE 1 This example shows the **XFA** entry consisting of an array of packets.

```
1 0 obj                              XFA entry in interactive form dictionary
   << /XFA [(xdp:xdp) 10 0 R         XFA resource specified as individual packets
         (template) 11 0 R
         (datasets) 12 0 R
         (config) 13 0 R
         (/xdp:xdp) 14 0 R ]
   >>
endobj
10 0 obj
   stream
      <xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
   endstream

11 0 obj
   stream
      <template xmlns="http://www.xfa.org/schema/xfa-template/2.4/">
      ...remaining contents of template packet...
      </template>
   endstream

12 0 obj
   stream
      <xfa:datasets xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
      ...contents of datasets packet...
      </xfa:datasets>
   endstream

13 0 obj
   stream
      <config xmlns="http://www.xfa.org/schema/xci/1.0/">
      ...contents of config node of XFA Data Package...
      <config>
   endstream

14 0 obj
   stream
      </xdp:xdp>
   endstream
```

EXAMPLE 2 The following example shows the same entry specified as a stream.

```
1 0 obj                 XFA entry in interactive form dictionary
   << /XFA 10 0 R >>
endobj

10 0 obj
   stream
      <xdp:xdp xmlns:xdp="http://ns.adobe.com/xdp/">
      <template xmlns="http://www.xfa.org/schema/xfa-template/2.4/">
      ...remaining contents of template packet...
      </template>
```

```
                <xfa:datasets xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
                ...contents of datasets packet...
                </xfa:datasets>
                <config xmlns="http://www.xfa.org/schema/xci/1.0/">
                ...contents of config node of XFA Data Package...
                <config>
                </xdp:xdp>
            endstream
        endobj
```

When an **XFA** entry is present in an interactive form dictionary, the XFA resource provides information about the form; in particular, all form-related events such as calculations and validations. The other entries in the interactive form dictionary shall be consistent with the information in the XFA resource. When creating or modifying a PDF file with an XFA resource, a conforming writer shall follow these rules:

- PDF interactive form field objects shall be present for each field specified in the XFA resource. The XFA field values shall be consistent with the corresponding **V** entries of the PDF field objects.
- The *XFA Scripting Object Model (SOM)* specifies a naming convention that shall be used to connect interactive form field names with field names in the XFA resource. Information about this model is available in the *XFA Specification, version 2.5* (see the Bibliography).
- No **A** or **AA** entries (see Table 164) shall be present in the annotation dictionaries of fields that also have actions specified by the XFA resource.

12.8 Digital Signatures

12.8.1 General

A digital signature *(PDF 1.3)* may be used to authenticate the identity of a user and the document’s contents. It stores information about the signer and the state of the document when it was signed. The signature may be purely mathematical, such as a public/private-key encrypted document digest, or it may be a biometric form of identification, such as a handwritten signature, fingerprint, or retinal scan. The specific form of authentication used shall be implemented by a special software module called a *signature handler*. Signature handlers shall be identified in accordance with the rules defined in Annex E.

Digital signatures in ISO 32000 currently support two activities: adding a digital signature to a document and later checking that signature for validity. Revocation information is a signed attribute, which means that the signing software must capture the revocation information before signing. A similar requirement applies to the chain of certificates. The signing software must capture and validate the certificate's chain *before* signing.

Signature information shall be contained in a *signature dictionary*, whose entries are listed in Table 252. Signature handlers may use or omit those entries that are marked optional in the table but should use them in a standard way if they are used at all. In addition, signature handlers may add private entries of their own. To avoid name duplication, the keys for all such private entries shall be prefixed with the registered handler name followed by a PERIOD (2Eh).

Signatures shall be created by computing a *digest* of the data (or part of the data) in a document, and storing the digest in the document. To verify the signature, the digest shall be re-computed and compared with the one stored in the document. Differences in the digest values indicate that modifications have been made since the document was signed.

There are two defined techniques for computing a digital signature of the contents of all or part of a PDF file:

- A *byte range digest* shall be computed over a range of bytes in the file, that shall be indicated by the **ByteRange** entry in the signature dictionary. This range should be the entire file, including the signature dictionary but excluding the signature value itself (the **Contents** entry). Other ranges may be used but since they do not check for all changes to the document, their use is not recommended. When a byte range digest is present, all values in the signature dictionary shall be direct objects.

- Additionally, modification detection may be specified by a *signature reference dictionary*. The **TransformMethod** entry shall specify the general method for modification detection, and the **TransformParams** entry shall specify the variable portions of the method.

A PDF document may contain the following standard types of signatures:

- One or more approval signatures. These signatures appear in signature form fields (see 12.7.4.5, “Signature Fields”). The signature dictionary corresponding to each signature shall be the value of the form field (as specified by its **V** entry). The signature dictionary shall contain a **ByteRange** entry representing a byte range digest, as described previously. A signature shall be validated by recomputing the digest and comparing it with the one stored in the signature.

NOTE 1 If a signed document is modified and saved by incremental update (see 7.5.6, “Incremental Updates”), the data corresponding to the byte range of the original signature is preserved. Therefore, if the signature is valid, it is possible to recreate the state of the document as it existed at the time of signing.

- At most one certification signature (*PDF 1.5*). The signature dictionary of a certification signature shall be the value of a signature field and shall contain a **ByteRange** entry. It may also be referenced from the **DocMDP** entry in the permissions dictionary (see 12.8.4, “Permissions”). The signature dictionary shall contain a signature reference dictionary (see Table 253) that has a **DocMDP** transform method. See 12.8.2.2, “DocMDP” for information on how these signatures shall be created and validated.

  A signature dictionary for a certification or approval signature may also have a signature reference dictionary with a **FieldMDP** transform method; see 12.8.2.4, “FieldMDP.”

- At most two *usage rights* signatures (*PDF 1.5*). Its signature dictionary shall be referenced from the **UR3** (*PDF 1.6*) entry in the permissions dictionary, whose entries are listed in Table 258, (not from a signature field). The signature dictionary shall contain a **Reference** entry whose value is a signature reference dictionary that has a **UR** transform method. See 12.8.2.3, “UR” for information on how these signatures shall be created and validated.

Table 252 – Entries in a signature dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Sig** for a signature dictionary.
Filter	name	*(Required; inheritable)* The name of the preferred signature handler to use when validating this signature. If the **Prop_Build** entry is not present, it shall be also the name of the signature handler that was used to create the signature. If **Prop_Build** is present, it may be used to determine the name of the handler that created the signature (which is typically the same as **Filter** but is not needed to be). A conforming reader may substitute a different handler when verifying the signature, as long as it supports the specified **SubFilter** format. Example signature handlers are **Adobe.PPKLite**, **Entrust.PPKEF**, **CICI.SignIt**, and **VeriSign.PPKVS**. The name of the filter (i.e. signature handler) shall be identified in accordance with the rules defined in Annex E.
SubFilter	name	*(Optional)* A name that describes the encoding of the signature value and key information in the signature dictionary. A conforming reader may use any handler that supports this format to validate the signature. *(PDF 1.6)* The following values for public-key cryptographic signatures shall be used: **adbe.x509.rsa_sha1**, **adbe.pkcs7.detached**, and **adbe.pkcs7.sha1** (see 12.8.3, “Signature Interoperability”). Other values may be defined by developers, and when used, shall be prefixed with the registered developer identification. All prefix names shall be registered (see Annex E). The prefix “adbe” has been registered by Adobe Systems and the three subfilter names listed above and defined in 12.8.3, “Signature Interoperability“ may be used by any developer.

Table 252 – Entries in a signature dictionary (continued)

Key	Type	Value
Contents	byte string	*(Required)* The signature value. When **ByteRange** is present, the value shall be a hexadecimal string (see 7.3.4.3, “Hexadecimal Strings”) representing the value of the byte range digest. For public-key signatures, **Contents** should be either a DER-encoded PKCS#1 binary data object or a DER-encoded PKCS#7 binary data object. Space for the Contents value must be allocated before the message digest is computed. (See 7.3.4, “String Objects“)
Cert	array or byte string	*(Required when **SubFilter** is **adbe.x509.rsa_sha1**)* An array of byte strings that shall represent the X.509 certificate chain used when signing and verifying signatures that use public-key cryptography, or a byte string if the chain has only one entry. The signing certificate shall appear first in the array; it shall be used to verify the signature value in **Contents**, and the other certificates shall be used to verify the authenticity of the signing certificate. If **SubFilter** is **adbe.pkcs7.detached** or **adbe.pkcs7.sha1**, this entry shall not be used, and the certificate chain shall be put in the PKCS#7 envelope in **Contents**.
ByteRange	array	*(Required for all signatures that are part of a signature field and usage rights signatures referenced from the **UR3** entry in the permissions dictionary)* An array of pairs of integers (starting byte offset, length in bytes) that shall describe the exact byte range for the digest calculation. Multiple discontiguous byte ranges shall be used to describe a digest that does not include the signature value (the **Contents** entry) itself.
Reference	array	*(Optional; PDF 1.5)* An array of signature reference dictionaries (see Table 253).
Changes	array	*(Optional)* An array of three integers that shall specify changes to the document that have been made between the previous signature and this signature: in this order, the number of pages altered, the number of fields altered, and the number of fields filled in. The ordering of signatures shall be determined by the value of ***ByteRange***. Since each signature results in an incremental save, later signatures have a greater length value.
Name	text string	*(Optional)* The name of the person or authority signing the document. This value should be used only when it is not possible to extract the name from the signature. EXAMPLE 1 From the certificate of the signer.
M	date	*(Optional)* The time of signing. Depending on the signature handler, this may be a normal unverified computer time or a time generated in a verifiable way from a secure time server. This value should be used only when the time of signing is not available in the signature. EXAMPLE 2 A time stamp can be embedded in a PKCS#7 binary data object (see 12.8.3.3, “PKCS#7 Signatures as used in ISO 32000”).
Location	text string	*(Optional)* The CPU host name or physical location of the signing.
Reason	text string	*(Optional)* The reason for the signing, such as (I agree…).
ContactInfo	text string	*(Optional)* Information provided by the signer to enable a recipient to contact the signer to verify the signature. EXAMPLE 3 A phone number.

Table 252 – Entries in a signature dictionary (continued)

Key	Type	Value
R	integer	*(Optional)* The version of the signature handler that was used to create the signature. *(PDF 1.5)* This entry shall not be used, and the information shall be stored in the Prop_Build dictionary.
V	integer	*(Optional; PDF 1.5)* The version of the signature dictionary format. It corresponds to the usage of the signature dictionary in the context of the value of **SubFilter**. The value is 1 if the **Reference** dictionary shall be considered critical to the validation of the signature. Default value: 0.
Prop_Build	dictionary	*(Optional; PDF 1.5)* A dictionary that may be used by a signature handler to record information that captures the state of the computer environment used for signing, such as the name of the handler used to create the signature, software build date, version, and operating system. he PDF Signature Build Dictionary Specification, provides implementation guidelines for the use of this dictionary.
Prop_AuthTime	integer	*(Optional; PDF 1.5)* The number of seconds since the signer was last authenticated, used in claims of signature repudiation. It should be omitted if the value is unknown.
Prop_AuthType	name	*(Optional; PDF 1.5)* The method that shall be used to authenticate the signer, used in claims of signature repudiation. Valid values shall be PIN, Password, and Fingerprint.

NOTE 2 The entries in the signature dictionary can be conceptualized as being in different dictionaries; they are in one dictionary for historical and cryptographic reasons. The categories are signature properties (**R**, **M**, **Name**, **Reason**, **Location**, **Prop_Build**, **Prop_AuthTime**, and **Prop_AuthType**); key information (**Cert** and portions of **Contents** when the signature value is a PKCS#7 object); reference (Reference and **ByteRange**); and signature value (**Contents** when the signature value is a PKCS#1 object).

Table 253 – Entries in a signature reference dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SigRef** for a signature reference dictionary.
TransformMethod	name	*(Required)* The name of the transform method (see Section 12.8.2, “Transform Methods”) that shall guide the modification analysis that takes place when the signature is validated. Valid values shall be: **DocMDP** Used to detect modifications to a document relative to a signature field that is signed by the originator of a document; see 12.8.2.2, “DocMDP.” **UR** Used to detect modifications to a document that would invalidate a signature in a rights-enabled document; see 12.8.2.3, “UR.” **FieldMDP** Used to detect modifications to a list of form fields specified in **TransformParams**; see 12.8.2.4, “FieldMDP.”
TransformParams	dictionary	*(Optional)* A dictionary specifying transform parameters (variable data) for the transform method specified by **TransformMethod**. Each method takes its own set of parameters. See each of the sub-clauses specified previously for details on the individual transform parameter dictionaries

Table 253 – Entries in a signature reference dictionary (continued)

Key	Type	Value
Data	(various)	*(Required when **TransformMethod** is **FieldMDP**)* An indirect reference to the object in the document upon which the object modification analysis should be performed. For transform methods other than **FieldMDP**, this object is implicitly defined.
DigestMethod	name	*(Optional; PDF 1.5 required)* A name identifying the algorithm that shall be used when computing the digest. Valid values are *MD5* and *SHA1*. Default value: *MD5*. For security reasons, MD5 should not be used. It is mentioned for backwards compatibility, since it remains the default value.

12.8.2 Transform Methods

12.8.2.1 General

Transform methods, along with transform parameters, shall determine which objects are included and excluded in revision comparison. The following sub-clauses discuss the types of transform methods, their transform parameters, and when they shall be used.

12.8.2.2 DocMDP

12.8.2.2.1 General

The **DocMDP** transform method shall be used to detect modifications relative to a signature field that is signed by the author of a document (the person applying the first signature). A document can contain only one signature field that contains a **DocMDP** transform method; it shall be the first signed field in the document. It enables the author to specify what changes shall be permitted to be made the document and what changes invalidate the author's signature.

NOTE As discussed earlier, "MDP" stands for *modification detection and prevention*. Certification signatures that use the DocMDP transform method enable *detection* of disallowed changes specified by the author. In addition, disallowed changes can also be *prevented* when the signature dictionary is referred to by the **DocMDP** entry in the permissions dictionary (see 12.8.4, "Permissions").

A certification signature should have a legal attestation dictionary (see 12.8.5, "Legal Content Attestations") that specifies all content that might result in unexpected rendering of the document contents, along with the author's attestation to such content. This dictionary may be used to establish an author's intent if the integrity of the document is questioned.

The **P** entry in the **DocMDP** transform parameters dictionary (see Table 254) shall indicate the author's specification of which changes to the document will invalidate the signature. (These changes to the document shall also be prevented if the signature dictionary is referred from the **DocMDP** entry in the permissions dictionary.) A value of 1 for **P** indicates that the document shall be final; that is, any changes shall invalidate the signature. The values 2 and 3 shall permit modifications that are appropriate for form field or comment workflows.

12.8.2.2.2 Validating Signatures That Use the DocMDP Transform Method

To validate a signature that uses the DocMDP transform method, a conforming reader first shall verify the byte range digest. Next, it shall verify that any modifications that have been made to the document are permitted by the transform parameters.

Once the byte range digest is validated, the portion of the document specified by the **ByteRange** entry in the signature dictionary (see Table 252) is known to correspond to the state of the document at the time of signing.

Therefore, conforming readers may compare the signed and current versions of the document to see whether there have been modifications to any objects that are not permitted by the transform parameters.

Table 254 – Entries in the DocMDP transform parameters dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **TransformParams** for a transform parameters dictionary.
P	number	*(Optional)* The access permissions granted for this document. Valid values shall be: 1 No changes to the document shall be permitted; any change to the document shall invalidate the signature. 2 Permitted changes shall be filling in forms, instantiating page templates, and signing; other changes shall invalidate the signature. 3 Permitted changes shall be the same as for 2, as well as annotation creation, deletion, and modification; other changes shall invalidate the signature. Default value: 2.
V	name	*(Optional)* The **DocMDP** transform parameters dictionary version. The only valid value shall be **1.2**. NOTE this value is a name object, not a number. Default value: **1.2**.

12.8.2.3 UR

The **UR** transform method shall be used to detect changes to a document that shall invalidate a *usage rights* signature, which is referred to from the **UR3** entry in the permissions dictionary (see 12.8.4, “Permissions”). Usage rights signatures shall be used to enable additional interactive features that may not available by default in a conforming reader. The signature shall be used to validate that the permissions have been granted by a bonafide granting authority. The transform parameters dictionary (see Table 255) specifies the additional rights that shall be enabled if the signature is valid. If the signature is invalid because the document has been modified in a way that is not permitted or the identity of the signer is not granted the extended permissions, additional rights shall not be granted.

EXAMPLE Adobe Systems grants permissions to enable additional features in Adobe Reader, using public-key cryptography. It uses certificate authorities to issue public key certificates to document creators with which it has entered into a business relationship. Adobe Reader verifies that the rights-enabling signature uses a certificate from an Adobe-authorized certificate authority. Other conforming readers are free to use this same mechanism for their own purposes.

UR3 (*PDF 1.6*): The **ByteRange** entry in the signature dictionary (see Table 252) shall be present. First, a conforming reader shall verify the byte range digest to determine whether the portion of the document specified by **ByteRange** corresponds to the state of the document at the time of signing. Next, a conforming reader shall examine the current version of the document to see whether there have been modifications to any objects that are not permitted by the transform parameters.

Table 255 – Entries in the UR transform parameters dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **TransformParams** for a transform parameters dictionary.
Document	array	*(Optional)* An array of names specifying additional document-wide usage rights for the document. The only defined value shall be FullSave, which permits a user to save the document along with modified form and/or annotation data. *(PDF 1.5)* Any usage right that permits the document to be modified implicitly shall enable the *FullSave* right. If the PDF document contains a **UR3** dictionary, only rights specified by the **Annots** entry that permit the document to be modified shall implicitly enable the *FullSave* right. For all other rights, *FullSave* shall be explicitly enabled in order to save the document. (Signature rights shall permit saving as part of the signing process but not otherwise). If the **P** entry in the **UR** transform parameters dictionary is **true** *(PDF 1.6)* and greater conforming readers shall permit only those rights that are enabled by the entries in the dictionary. However, conforming readers shall permit saving the document as long as any rights that permit modifying the document are enabled.
Msg	text string	*(Optional)* A text string that may be used to specify any arbitrary information, such as the reason for adding usage rights to the document.
V	name	*(Optional)* The **UR** transform parameters dictionary version. The value shall be **2.2**. If an unknown version is present, no rights shall be enabled. NOTE This value is a name object, not a number. Default value: **2.2**.
Annots	array	*(Optional)* An array of names specifying additional annotation-related usage rights for the document. Valid names *(PDF 1.5)* are Create, Delete, Modify, Copy, Import, and Export, which shall permit the user to perform the named operation on annotations. The following names *(PDF 1.6)* shall be permitted only when the signature dictionary is referenced from the **UR3** entry of the permissions dictionary (see Table 258): Online — Permits online commenting; that is, the ability to upload or download markup annotations from a server. SummaryView — Permits a user interface to be shown that summarizes the comments (markup annotations) in a document.

Table 255 – Entries in the UR transform parameters dictionary (continued)

Key	Type	Value	
Form	array	*(Optional)* An array of names specifying additional form-field-related usage rights for the document. Valid names *(PDF 1.5)* are:	
		Add	Permits the user to add form fields to the document.
		Delete	Permits the user to delete form fields to the document.
		FillIn	Permits the user to save a document on which form fill-in has been done.
		Import	Permits the user to import form data files in FDF, XFDF and text (CSV/TSV) formats.
		Export	Permits the user to export form data files as FDF or XFDF.
		SubmitStandalone	Permits the user to submit form data when the document is not open in a Web browser.
		SpawnTemplate	Permits new pages to be instantiated from named page templates.
		The following names *(PDF 1.6)* shall be permitted only when the signature dictionary is referenced from the **UR3** entry of the permissions dictionary; see Table 258:	
		BarcodePlaintext	Permits *(PDF 1.6)* text form field data to be encoded as a plaintext two-dimensional barcode.
		Online	Permits *(PDF 1.6)* the use of forms-specific online mechanisms such as SOAP or Active Data Object.
Signature	array	*(Optional)* An array of names specifying additional signature-related usage rights for the document. The only defined value shall be Modify, which permits a user to apply a digital signature to an existing signature form field or clear a signed signature form field.	
EF	array	*(Optional; PDF 1.6)* An array of names specifying additional usage rights for named embedded files in the document. Valid names shall be Create, Delete, Modify, and Import, which shall permit the user to perform the named operation on named embedded files.	
P	boolean	*(Optional; PDF 1.6)* If **true**, permissions for the document shall be restricted in all consumer applications to those permissions granted by a conforming reader, while allowing permissions for rights enabled by other entries in this dictionary. Default value: **false**.	

12.8.2.4 FieldMDP

The **FieldMDP** transform method shall be used to detect changes to the values of a list of form fields. The entries in its transform parameters dictionary are listed in Table 256.

Table 256 – Entries in the FieldMDP transform parameters dictionary

Key	Type	Value	
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **TransformParams** for a transform parameters dictionary.	
Action	name	*(Required)* A name that, along with the **Fields** array, describes which form fields do not permit changes after the signature is applied.	
		Valid values shall be:	
		All	All form fields.
		Include	Only those form fields that specified in **Fields**.
		Exclude	Only those form fields not specified in **Fields**.
Fields	array	*(Required if **Action** is Include or Exclude)* An array of text strings containing field names.	

Table 256 – Entries in the FieldMDP transform parameters dictionary (continued)

Key	Type	Value
V	name	*(Optional*: PDF 1.5 required) The transform parameters dictionary version. The value for PDF 1.5 and later shall be *1.2*. NOTE This value is a name object, not a number. Default value: *1.2*.

On behalf of a document author creating a document containing both form fields and signatures the following shall be supported by conforming writers:

- The author specifies that form fields shall be filled in without invalidating the approval or certification signature. The **P** entry of the **DocMDP** transform parameters dictionary shall be set to either 2 or 3 (see Table 254).
- The author can also specify that after a specific recipient has signed the document, any modifications to specific form fields shall invalidate that recipient's signature. There shall be a separate signature field for each designated recipient, each having an associated signature field lock dictionary (see Table 233) specifying the form fields that shall be locked for that user.
- When the recipient signs the field, the signature, signature reference, and transform parameters dictionaries shall be created. The **Action** and **Fields** entries in the transform parameters dictionary shall be copied from the corresponding fields in the signature field lock dictionary.

NOTE This copying is done because all objects in a signature dictionary must be direct objects if the dictionary contains a byte range signature. Therefore, the transform parameters dictionary cannot reference the signature field lock dictionary indirectly.

FieldMDP signatures shall be validated in a similar manner to **DocMDP** signatures. See Validating Signatures That Use the DocMDP Transform Method in 12.8.2.2, "DocMDP" for details.

12.8.3 Signature Interoperability

12.8.3.1 General

It is intended that conforming readers allow interoperability between signature handlers; that is, a PDF file signed with a handler from one vendor shall be able to be validated with a handler from a different vendor.

If present, the **SubFilter** entry in the signature dictionary shall specify the encoding of the signature value and key information, while the **Filter** entry shall specify the preferred handler that should be used to validate the signature. When handlers are being registered according to Annex E they shall specify the SubFilter encodings they support enabling handlers other than the preferred handler to validate the signatures that the preferred handler creates.

There are several defined values for the **SubFilter** entry, all based on public-key cryptographic standards published by RSA Security and also as part of the standards issued by the Internet Engineering Task Force (IETF) Public Key Infrastructure (PKIX) working group; see the Bibliography for references.

12.8.3.2 PKCS#1 Signatures

The PKCS#1 standard supports several public-key cryptographic algorithms and digest methods, including RSA encryption, DSA signatures, and SHA-1 and MD5 digests (see the Bibliography for references). For signing PDF files using PKCS#1, the only value of SubFilter that should be used is adbe.x509.rsa_sha1, which uses the RSA encryption algorithm and SHA-1 digest method. The certificate chain of the signer shall be stored in the **Cert** entry.

12.8.3.3 PKCS#7 Signatures as used in ISO 32000

12.8.3.3.1 General

When PKCS#7 signatures are used, the value of **Contents** shall be a DER-encoded PKCS#7 binary data object containing the signature. The PKCS#7 object shall conform to RFC3852 Cryptographic Message Syntax. Different subfilters may be used and shall be registered in accordance with Annex E. **SubFilter** shall take one of the following values:

- **adbe.pkcs7.detached**: The original signed message digest over the document’s byte range shall be incorporated as the normal PKCS#7 SignedData field. No data shall be encapsulated in the PKCS#7 SignedData field.

- **adbe.pkcs7.sha1**: The SHA1 digest of the document’s byte range shall be encapsulated in the PKCS#7 SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal PKCS#7 digest.

The PKCS#7 object shall conform to the PKCS#7 specification in Internet RFC 2315, *PKCS #7: Cryptographic Message Syntax, Version 1.5* (see the Bibliography). At minimum, it shall include the signer’s X.509 signing certificate. This certificate shall be used to verify the signature value in **Contents**.

The PKCS#7 object should contain the following:

- Time stamp information as an unsigned attribute *(PDF 1.6)*: The timestamp token shall conform to RFC 3161 and shall be computed and embedded into the PKCS#7 object as described in Appendix A of RFC 3161. The specific treatment of timestamps and their processing is left to the particular signature handlers to define.

- Revocation information as an signed attribute *(PDF 1.6)*: This attribute may include all the revocation information that is necessary to carry out revocation checks for the signer's certificate and its issuer certificates. Since revocation information is a signed attribute, it must be obtained before the computation of the digital signature. This means that the software used by the signer must be able to construct the certification path and the associated revocation information. If one of the elements cannot be obtained (e.g. no connection is possible), a signature with this attribute will not be possible.

- *(PDF 1.6)*. This differs from the treatment when using *adbe.x509.rsa_sha1* when the certificates shall be placed in the **Cert** key of the signature dictionary as defined in Table 252.

- One or more RFC 3281 attribute certificates associated with the signer certificate *(PDF 1.7)*. The specific treatment of attribute certificates and their processing is left to the particular signature handlers to define.

NOTE For maximum compatibility with earlier versions, conforming writers should follow this practice.

The policy of how to establish trusted identity lists to validate embedded certificates is up to the validation signature handler.

12.8.3.3.2 Revocation Information

The adbe Revocation Information attribute:

```
adbe-revocationInfoArchival OBJECT IDENTIFIER ::=
                              { adbe(1.2.840.113583) acrobat(1) security(1) 8 }
```

The value of the revocation information attribute can include any of the following data types:

- Certificate Revocation Lists (CRLs), described in RFC 3280 (see the Bibliography): CRLs are generally large and therefore should not be embedded in the PKCS#7 object.

- Online Certificate Status Protocol (OCSP) Responses, described in RFC 2560, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP* (see the Bibliography): These are generally small and constant in size and should be the data type included in the PKCS#7 object.
- Custom revocation information: The format is not prescribed by this specification, other than that it be encoded as an OCTET STRING. The application should be able to determine the type of data contained within the OCTET STRING by looking at the associated OBJECT IDENTIFIER.

adbe's Revocation Information attribute value has ASN.1 type RevocationInfoArchival:

```
RevocationInfoArchival ::= SEQUENCE {
   crl          [0] EXPLICIT SEQUENCE of CRLs, OPTIONAL
   ocsp         [1] EXPLICIT SEQUENCE of OCSP Responses, OPTIONAL
   otherRevInfo [2] EXPLICIT SEQUENCE of OtherRevInfo, OPTIONAL
}
OtherRevInfo ::= SEQUENCE {
Type  OBJECT IDENTIFIER
Value OCTET STRING
}
```

For byte range signatures, **Contents** shall be a hexadecimal string with “<” and “>” delimiters. It shall fit precisely in the space between the ranges specified by **ByteRange**. Since the length of PKCS#7 objects is not entirely predictable, the value of **Contents** shall be padded with zeros at the end of the string (before the “>” delimiter) before writing the PKCS#7 to the allocated space in the file.

The format for encoding signature values should be **adbe.pkcs7.detached**. This encoding allows the most options in terms of algorithm use. The following table shows the algorithms supported for the various **SubFilter** values.

Table 257 – SubFilter value algorithm support

	SubFilter value		
	adbe.pkcs7.detached	**adbe.pkcs7.sha1**	**adbe.x509.rsa.sha1**[a]
Message Digest	SHA1 *(PDF 1.3)* SHA256 *(PDF 1.6)* SHA384 *(PDF 1.7)* SHA512 *(PDF 1.7)* RIPEMD160 *(PDF 1.7)*	SHA1 *(PDF 1.3)*[b]	SHA1 *(PDF 1.3)* SHA256 *(PDF 1.6)* SHA384 *(PDF 1.7)* SHA512 *(PDF 1.7)* RIPEMD160 *(PDF 1.7)*
RSA Algorithm Support	Up to 1024-bit *(PDF 1.3)* Up to 2048-bit *(PDF 1.5)* Up to 4096-bit *(PDF 1.5)*	See **adbe.pkcs7.detached**	See **adbe.pkcs7.detached**
DSA Algorithm Support	Up to 4096-bits *(PDF 1.6)*	See **adbe.pkcs7.detached**	No

[a] Despite the appearance of **sha1** in the name of this **SubFilter** value, supported encodings shall not be limited to the SHA1 algorithm. The PKCS#1 object contains an identifier that indicates which algorithm shall be used.

[b] Other digest algorithms may be used to digest the signed-data field; however, SHA1 shall be used to digest the data that is being signed.

12.8.4 Permissions

The **Perms** entry in the document catalogue (see Table 28) shall specify a *permissions dictionary (PDF 1.5)*. Each entry in this dictionary (see Table 258 for the currently defined entries) shall specify the name of a permission handler that controls access permissions for the document. These permissions are similar to those defined by security handlers (see Table 22) but do not require that the document be encrypted. For a permission to be actually granted for a document, it shall be allowed by each permission handler that is present in the permissions dictionary as well as by the security handler.

NOTE An example of a permission is the ability to fill in a form field.

Table 258 – Entries in a permissions dictionary

Key	Type	Value
DocMDP	dictionary	*(Optional)* An indirect reference to a signature dictionary (see Table 252). This dictionary shall contain a **Reference** entry that shall be a signature reference dictionary (see Table 252) that has a **DocMDP** transform method (see 12.8.2.2, “DocMDP”) and corresponding transform parameters. If this entry is present, consumer applications shall enforce the permissions specified by the **P** attribute in the **DocMDP** transform parameters dictionary and shall also validate the corresponding signature based on whether any of these permissions have been violated.
UR3	dictionary	*(Optional)* A signature dictionary that shall be used to specify and validate additional capabilities (usage rights) granted for this document; that is, the enabling of interactive features of the conforming reader that are not available by default. For example, A conforming reader does not permit saving documents by default, but an agent may grant permissions that enable saving specific documents. The signature shall be used to validate that the permissions have been granted by the agent that did the signing. The signature dictionary shall contain a **Reference** entry that shall be a signature reference dictionary that has a **UR** transform method (see 12.8.2.3, “UR”). The transform parameter dictionary for this method indicates which additional permissions shall be granted for the document. If the signature is valid, the conforming reader shall allow the specified permissions for the document, in addition to the application’s default permissions.

12.8.5 Legal Content Attestations

The PDF language provides a number of capabilities that can make the rendered appearance of a PDF document vary. These capabilities could potentially be used to construct a document that misleads the recipient of a document, intentionally or unintentionally. These situations are relevant when considering the legal implications of a signed PDF document.

Therefore, a mechanism shall be provided by which a document recipient can determine whether the document can be trusted. The primary method is to accept only documents that contain certification signatures (one that has a **DocMDP** signature that defines what shall be permitted to change in a document; see 12.8.2.2, “DocMDP”).

When creating certification signatures, conforming writers should also create a *legal attestation dictionary*, whose entries are shown in Table 259. This dictionary shall be the value of the **Legal** entry in the document catalogue (see Table 28). Its entries shall specify all content that may result in unexpected rendering of the document contents. The author may provide further clarification of such content by means of the **Attestation** entry. Reviewers should establish for themselves that they trust the author and contents of the document. In the case of a legal challenge to the document, any questionable content can be reviewed in the context of the information in this dictionary.

Table 259 – Entries in a legal attestation dictionary

Key	Type	Value
JavaScriptActions	integer	*(Optional)* The number of JavaScript actions found in the document (see 12.6.4.16, “JavaScript Actions”).
LaunchActions	integer	*(Optional)* The number of launch actions found in the document (see 12.6.4.5, “Launch Actions”).
URIActions	integer	*(Optional)* The number of URI actions found in the document (see 12.6.4.7, “URI Actions”).
MovieActions	integer	*(Optional)* The number of movie actions found in the document (see 12.6.4.9, “Movie Actions”).
SoundActions	integer	*(Optional)* The number of sound actions found in the document (see 12.6.4.8, “Sound Actions”).
HideAnnotationActions	integer	*(Optional)* The number of hide actions found in the document (see 12.6.4.10, “Hide Actions”).
GoToRemoteActions	integer	*(Optional)* The number of remote go-to actions found in the document (see 12.6.4.3, “Remote Go-To Actions”).
AlternateImages	integer	*(Optional)* The number of alternate images found in the document (see 8.9.5.4, “Alternate Images”)
ExternalStreams	integer	*(Optional)* The number of external streams found in the document.
TrueTypeFonts	integer	*(Optional)* The number of TrueType fonts found in the document (see 9.6.3, “TrueType Fonts”).
ExternalRefXobjects	integer	*(Optional)* The number of reference XObjects found in the document (see 8.10.4, “Reference XObjects”).
ExternalOPIdicts	integer	*(Optional)* The number of OPI dictionaries found in the document (see 14.11.7, “Open Prepress Interface (OPI)”).
NonEmbeddedFonts	integer	*(Optional)* The number of non-embedded fonts found in the document (see 9.9, “Embedded Font Programs””)
DevDepGS_OP	integer	*(Optional)* The number of references to the graphics state parameter **OP** found in the document (see Table 58).
DevDepGS_HT	integer	*(Optional)* The number of references to the graphics state parameter **HT** found in the document (see Table 58).
DevDepGS_TR	integer	*(Optional)* The number of references to the graphics state parameter **TR** found in the document (see Table 58).
DevDepGS_UCR	integer	*(Optional)* The number of references to the graphics state parameter **UCR** found in the document (see Table 58).
DevDepGS_BG	integer	*(Optional)* The number of references to the graphics state parameter **BG** found in the document (see Table 58).
DevDepGS_FL	integer	*(Optional)* The number of references to the graphics state parameter **FL** found in the document (see Table 58).
Annotations	integer	*(Optional)* The number of annotations found in the document (see 12.5, “Annotations”).
OptionalContent	boolean	*(Optional)* **true** if optional content is found in the document (see 8.11, “Optional Content”).
Attestation	text string	*(Optional)* An attestation, created by the author of the document, explaining the presence of any of the other entries in this dictionary or the presence of any other content affecting the legal integrity of the document.

12.9 Measurement Properties

PDF documents, such as those created by CAD software, may contain graphics that are intended to represent real-world objects. Users of such documents often require information about the scale and units of measurement of the corresponding real-world objects and their relationship to units in PDF user space.

This information enables users of conforming readers to perform measurements that yield results in the units intended by the creator of the document. A measurement in this context is the result of a canonical function that takes as input a set of *n* coordinate pairs

$$\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$$

and produces a single number as output depending on the type of measurement. For example, distance measurement is equivalent to

$$\sum_{i=0}^{n-2} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

for $n \geq 2$.

Beginning with PDF 1.6, such information may be stored in a *measure dictionary* (see Table 261). Measure dictionaries provide information about measurement units associated with a rectangular area of the document known as a *viewport*.

A viewport *(PDF 1.6)* is a rectangular region of a page. The optional **VP** entry in a page dictionary (see Table 30) shall specify an array of viewport dictionaries, whose entries shall be as shown in Table 260. Viewports allow different measurement scales (specified by the **Measure** entry) to be used in different areas of a page, if necessary.

The dictionaries in the **VP** array shall be in drawing order. Since viewports might overlap, to determine the viewport to use for any point on a page, the dictionaries in the array shall be examined, starting with the last one and iterating in reverse, and the first one whose **BBox** entry contains the point shall be chosen.

NOTE 1 Any measurement that potentially involves multiple viewports, such as one specifying the distance between two points, shall use the information specified in the viewport of the first point.

Table 260 – Entries in a viewport dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; shall be **Viewport** for a viewport dictionary.
BBox	rectangle	*(Required)* A rectangle in default user space coordinates specifying the location of the viewport on the page. The two coordinate pairs of the rectangle shall be specified in normalized form; that is, lower-left followed by upper-right, relative to the measuring coordinate system. This ordering shall determine the orientation of the measuring coordinate system (that is, the direction of the positive *x* and *y* axes) in this viewport, which may have a different rotation from the page. The coordinates of this rectangle are independent of the origin of the measuring coordinate system, specified in the ***O*** entry (see Table 262) of the measurement dictionary specified by **Measure**.

Table 260 – Entries in a viewport dictionary (continued)

Key	Type	Value
Name	text string	*(Optional)* A descriptive text string or title of the viewport, intended for use in a user interface.
Measure	dictionary	*(Optional)* A measure dictionary (see Table 261) that specifies the scale and units that shall apply to measurements taken on the contents within the viewport.

A measure dictionary shall specify an alternate coordinate system for a region of a page. Along with the viewport dictionary, it shall provide the information needed to convert coordinates in the page's coordinate system to coordinates in the measuring coordinate system. The measure dictionary shall provide information for formatting the resulting values into textual form for presentation in a graphical user interface.

Table 261 shows the entries in a measure dictionary. PDF 1.6 defines only a single type of coordinate system, a *rectilinear* coordinate system, that shall be specified by the value **RL** for the **Subtype** entry, which is defined as one in which the *x* and *y* axes are perpendicular and have units that increment linearly (to the right and up, respectively). Other subtypes may be used, providing the flexibility to measure using other types of coordinate systems.

Table 261 – Entries in a measure dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; shall be **Measure** for a measure dictionary.
Subtype	name	*(Optional)* A name specifying the type of coordinate system to use for measuring. Default value: **RL**, which specifies a rectilinear coordinate system

Table 262 shows the additional entries in a rectilinear measure dictionary. Many of the entries in this dictionary shall be *number format arrays*, which are arrays of *number format dictionaries* (see Table 263). Each number format dictionary shall represent a specific unit of measurement (such as miles or feet). It shall contain information about how each unit shall be expressed in text and factors for calculating the number of units.

Number format arrays specify all the units that shall be used when expressing a specific measurement. Each array shall contain one or more number format dictionaries, in descending order of granularity. If one unit of measurement X is larger than one unit of measurement Y then X has a larger order of granularity than Y. All the elements in the array shall contain text strings that, concatenated together, specify how the units shall be displayed.

NOTE 2 For example, a measurement of 1.4505 miles might be expressed as "1.4505 mi", which would require one number format dictionary for miles, or as "1 mi 2,378 ft 7 5/8 in", which would require three dictionaries (for miles, feet, and inches).

EXAMPLE 1 A number format dictionary specifying feet should precede one specifying inches.

Table 262 – Additional entries in a rectilinear measure dictionary

Key	Type	Value
R	text string	*(Required)* A text string expressing the *scale ratio* of the drawing in the region corresponding to this dictionary. Universally recognized unit abbreviations should be used, either matching those of the number format arrays in this dictionary or those of commonly used scale ratios. EXAMPLE 1 a common scale in architectural drawings is “1/4 in = 1 ft”, indicating that 1/4 inches in default user space is equivalent to 1 foot in real-world measurements. If the scale ratio differs in the *x* and *y* directions, both scales should be specified. EXAMPLE 2 “in X 1 cm = 1 m, in Y 1 cm = 30 m”.
X	array	*(Required)* A number format array for measurement of change along the *x* axis and, if **Y** is not present, along the *y* axis as well. The first element in the array shall contain the scale factor for converting from default user space units to the largest units in the measuring coordinate system along that axis. The directions of the *x* and *y* axes are in the measuring coordinate system and are independent of the page rotation. These directions shall be determined by the **BBox** entry of the containing viewport (see Table 260).
Y	array	*(Required when the* x *and* y *scales have different units or conversion factors)* A number format array for measurement of change along the *y* axis. The first element in the array shall contain the scale factor for converting from default user space units to the largest units in the measuring coordinate system along the *y* axis.
D	array	*(Required)* A number format array for measurement of distance in any direction. The first element in the array shall specify the conversion to the largest distance unit from units represented by the first element in **X**. The scale factors from **X**, **Y** (if present) and **CYX** (if **Y** is present) shall be used to convert from default user space to the appropriate units before applying the distance function.
A	array	*(Required)* A number format array for measurement of area. The first element in the array shall specify the conversion to the largest area unit from units represented by the first element in **X**, squared. The scale factors from **X**, **Y** (if present) and **CYX** (if **Y** is present) shall be used to convert from default user space to the appropriate units before applying the area function.
T	array	*(Optional)* A number format array for measurement of angles. The first element in the array shall specify the conversion to the largest angle unit from degrees. The scale factor from **CYX** (if present) shall be used to convert from default user space to the appropriate units before applying the angle function.
S	array	*(Optional)* A number format array for measurement of the slope of a line. The first element in the array shall specify the conversion to the largest slope unit from units represented by the first element in **Y** divided by the first element in **X**. The scale factors from **X**, **Y** (if present) and **CYX** (if **Y** is present) shall be used to convert from default user space to the appropriate units before applying the slope function.
O	array	*(Optional)* An array of two numbers that shall specify the origin of the measurement coordinate system in default user space coordinates. The directions by which *x* and *y* increase in value from this origin shall be determined by the viewport’s **BBox** entry (see Table 260). Default value: the first coordinate pair (lower-left corner) of the rectangle specified by the viewport’s **BBox** entry.

Table 262 – Additional entries in a rectilinear measure dictionary (continued)

Key	Type	Value
CYX	number	*(Optional; meaningful only when* ***Y*** *is present)* A factor that shall be used to convert the largest units along the *y* axis to the largest units along the *x* axis. It shall be used for calculations (distance, area, and angle) where the units are be equivalent; if not specified, these calculations may not be performed (which would be the case in situations such as *x* representing time and *y* representing temperature). Other calculations (change in *x*, change in *y*, and slope) shall not require this value.

The **X** and **Y** entries in a measure dictionary shall be number format arrays that shall specify the units used for measurements in the *x* and *y* directions, respectively, and the ratio between user space units and the specified units. **Y** is present only when the *x* and *y* measurements are in different units or have different ratios; in this case, the **CYX** entry shall be used to convert *y* values to *x* values when appropriate.

Table 263 – Entries in a number format dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; shall be **NumberFormat** for a number format dictionary.
U	text string	*(Required)* A text string specifying a label for displaying the units represented by this dictionary in a user interface; the label should use a universally recognized abbreviation.
C	number	*(Required)* The conversion factor used to multiply a value in partial units of the previous number format array element to obtain a value in the units of this dictionary. When this entry is in the first number format dictionary in the array, its meaning (that is, what it shall be multiplied by) depends on which entry in the rectilinear measure dictionary (see Table 262) references the number format array.
F	name	*(Optional; meaningful only for the last dictionary in a number format array)* A name indicating whether and in what manner to display a fractional value from the result of converting to the units of this dictionary by means of the **C** entry. Valid values shall be: **D** Show as decimal to the precision specified by the **D** entry. **F** Show as a fraction with denominator specified by the **D** entry. **R** No fractional part; round to the nearest whole unit. **T** No fractional part; truncate to achieve whole units. Default value: **D**.
D	integer	*(Optional; meaningful only for the last dictionary in a number format array)* A positive integer that shall specify the precision or denominator of a fractional amount: When the value of **F** is **D**, this entry shall be the precision of a decimal display; it shall be a multiple of 10. Low-order zeros may be truncated unless **FD** is **true**. Default value: 100 (hundredths, corresponding to two decimal digits). When the value of **F** is **F**, this entry shall be the denominator of a fractional display. The fraction may be reduced unless the value of **FD** is **true**. Default value: 16.
FD	boolean	*(Optional; meaningful only for the last dictionary in a number format array)* If **true**, a fractional value formatted according to the **D** entry may not have its denominator reduced or low-order zeros truncated. Default value: **false**.

Table 263 – Entries in a number format dictionary (continued)

Key	Type	Value
RT	text string	*(Optional)* Text that shall be used between orders of thousands in display of numerical values. An empty string indicates that no text shall be added. Default value: COMMA (2Ch).
RD	text string	*(Optional)* Text that shall be used as the decimal position in displaying numerical values. An empty string indicates that the default shall be used. Default value: PERIOD (2Eh).
PS	text string	*(Optional)* Text that shall be concatenated to the left of the label specified by **U**. An empty string indicates that no text shall be added. Default value: A single ASCII SPACE character (20h).
SS	text string	*(Optional)* Text that shall be concatenated after the label specified by **U**. An empty string indicates that no text shall be added. Default value: A single ASCII SPACE character (20h).
O	name	*(Optional)* A name indicating the position of the label specified by **U** with respect to the calculated unit value. Valid values shall be: **S** The label is a suffix to the value. **P** The label is a prefix to the value. The characters specified by ***PS*** and ***SS*** shall be concatenated before considering this entry. Default value: **S**.

To use a number format array to create a text string containing the appropriately formatted units for display in a user interface, apply the following algorithm:

Algorithm: Use of a number format array to create a formatted text string

a) The entry in the rectilinear measure dictionary (see Table 262) that references the number format array determines the meaning of the initial measurement value. For example, the **X** entry specifies user space units, and the **T** entry specifies degrees.

b) Multiply the value specified previously by the **C** entry of the first number format dictionary in the array, which converts the measurement to units of the largest granularity specified in the array. Apply the value of **RT** as appropriate.

c) If the result contains no nonzero fractional portion, concatenate the label specified by the **U** entry in the order specified by **O**, after adding spacing from **PS** and **SS**. The formatting is then complete.

d) If there is a nonzero fractional portion and no more elements in the array, format the fractional portion as specified by the **RD**, **F**, **D**, and **FD** entries of the last dictionary. Concatenate the label specified by the **U** entry in the order specified by **O**, after adding spacing from **PS** and **SS**. The formatting is then complete.

e) If there is a nonzero fractional portion and more elements in the array, proceed to the next number format dictionary in the array. Multiply its **C** entry by the fractional result from the previous step. Apply the value of **RT** as appropriate. Then proceed to step 3.

The concatenation of elements in this process assumes left-to-right order. Documents using right-to-left languages may modify the process and the meaning of the entries as appropriate to produce the correct results.

EXAMPLE 2 The following example shows a measure dictionary that specifies that changes in *x* or *y* are expressed in miles; distances are expressed in miles, feet, and inches; and area is expressed in acres. Given a sample distance in scaled units of 1.4505 miles, the formatted text produced by applying the number format array would be
"1 mi 2,378 ft 7 5/8 in".

```
<</Type /Measure
    /Subtype /RL
    /R (1in = 0.1 mi)
    /X [ <</U (mi)             % x offset represented in miles
            /C .00139          % Conversion from user space units to miles
            /D 100000
        ]
    /D  [<< /U (mi) /C 1 >>    % Distance: initial unit is miles; no conversion needed
         << /U (ft) /C 5280 >> % Conversion from miles to feet
         << /U (in) /C 12      % Conversion from feet to inches
             /F /F /D 8 >>     % Fractions of inches rounded to nearest 1/8
        ]
    /A [<</U (acres)           % Area: measured in acres
            /C 640 >>          % Conversion from square miles to acres
        ]
>>
```

12.10 Document Requirements

12.10.1 General

Beginning with PDF 1.7, a document may specify requirements that shall be present in a conforming reader in order for the document to function properly. The **Requirements** entry in the document catalogue (see 7.7.2, "Document Catalog") shall specify an array of *requirement dictionaries*, whose entries are shown in Table 264.

Table 264 – Entries common to all requirement dictionaries

Key	Type	Description
Type	name	*(Optional)* The type of PDF object that this dictionary describes. If present, shall be **Requirement** for a requirement dictionary.
S	name	*(Required)* The type of requirement that this dictionary describes. The value shall be **EnableJavaScripts**.
RH	array	*(Optional)* An array of *requirement handler dictionaries* (see Table 265). This array lists the requirement handlers that shall be disabled (not executed) if the conforming reader can check the requirement specified in the **S** entry.

The **RH** entry ensures backward-capability for this feature. Some PDF documents include JavaScript segments that verify compliance with certain requirements. Such JavaScript segments are called *requirement handlers*. Backward-compatibility shall be achieved by ensuring that either the conforming reader checks the requirement or the JavaScript segment checks the requirement, but not both.

When a PDF document is first opened, all JavaScript segments in the document shall be executed, including the requirement handlers. If the conforming reader understands the requirement dictionary, it shall disable execution of the requirement handlers named by the **RH** entry. If the requirement handler is in JavaScript, the conforming reader shall look up the segment using the **Names** dictionary (7.7.4, "Name Dictionary").

In PDF 1.7, the only defined requirement type shall be **EnableJavaScripts**. This requirement indicates that the document requires JavaScript execution to be enabled in the conforming reader. If the **EnableJavaScripts** requirement is present, an interactive conforming reader may allow the user to choose between keeping JavaScript execution disabled or temporarily enabling it to benefit from the full function of the document.

If the **EnableJavaScripts** requirement is present in a requirement dictionary, the inclusion of the **RH** entry that specifies a JavaScript segment would be pointless. Writing a JavaScript segment to verify that JavaScript is enabled would not achieve the desired goal. The **RH** entry shall not be used in PDF 1.7.

12.10.2 Requirement Handlers

A requirement handler is a program that verifies certain requirements are satisfied. Table 265 describes the entries in a requirement handler dictionary.

Table 265 – Entries in a requirement handler dictionary

Key	Type	Description
Type	name	*(Optional)* The type of PDF object that this dictionary describes. If present, shall be **ReqHandler** for a requirement handler dictionary.
S	name	*(Required)* The type of requirement handler that this dictionary describes. Valid requirement handler types shall be **JS** (for a JavaScript requirement handlers) and **NoOp**. A value of **NoOp** allows older conforming readers to ignore unrecognized requirements. This value does not add any specific entry to the requirement handler dictionary.
Script	text string	(*Optional; valid only if the **S** entry has a value of **JS***) The name of a document-level JavaScript action stored in the document name dictionary (see 7.7.4, "Name Dictionary"). If the conforming reader understands the parent requirement dictionary and can verify the requirement specified in that dictionary, it shall disable execution of the requirement handler identified in this dictionary.

13 Multimedia Features

13.1 General

This clause describes those features of PDF that support embedding and playing multimedia content. It contains the following sub-clauses:

- 13.2, “Multimedia,” describes the comprehensive set of multimedia capabilities that were introduced in PDF 1.5.
- 13.3, “Sounds,” and 13.4, “Movies,” describe features that have been supported since PDF 1.2.
- 13.5, “Alternate Presentations,” describes a slideshow capability that was introduced in PDF 1.4.
- 13.6, “3D Artwork,” describes the capability of embedding three-dimensional graphics in a document, introduced in PDF 1.6.

13.2 Multimedia

13.2.1 General

PDF 1.5 introduces a comprehensive set of language constructs to enable the following capabilities:

- Arbitrary media types may be embedded in PDF files.
- Embedded media, as well as referenced media outside a PDF file, may be played with a variety of player software. (In some situations, the player software may be the conforming reader itself.)

NOTE 1 The term playing is used with a wide variety of media, and is not restricted to audio or video. For example, it may be applied to static images such as JPEGs.

- Media objects may have multiple *renditions*, which may be chosen at play-time based on considerations such as available bandwidth.
- Document authors may control play-time requirements, such as which player software should be used to play a given media object.
- Media objects may be played in various ways; for example, in a floating window as well as in a region on a page.
- Future extensions to the media constructs may be handled in an appropriate manner by current conforming readers. Authors may control how old conforming readers treat future extensions.
- Document authors may adapt the use of multimedia to accessibility requirements.
- On-line media objects may be played efficiently, even when very large.

The following list summarizes the multimedia features and indicates where each feature is discussed:

- 13.2.2, “Viability,” describes the rules for determining when media objects are suitable for playing on a particular system.
- Rendition actions (see 12.6.4.13, “Rendition Actions”) shall be used to begin the playing of multimedia content.
- A rendition action associates a screen annotation (see 12.5.6.18, “Screen Annotations”) with a rendition (see 13.2.3, “Renditions”).

- Renditions are of two varieties: media renditions (see 13.2.3.2, “Media Renditions”) that define the characteristics of the media to be played, and selector renditions (see 13.2.3.3, “Selector Renditions”) that enables choosing which of a set of media renditions should be played.

- Media renditions contain entries that specify what should be played (see 13.2.4, “Media Clip Objects”), how it should be played (see 13.2.5, “Media Play Parameters”), and where it should be played (see 13.2.6, “Media Screen Parameters”).

- 13.2.7, “Other Multimedia Objects,” describes several PDF objects that are referenced by the preceding major objects.

NOTE 2 Some of the features described in the following sub-clauses have references to corresponding elements in the *Synchronized Multimedia Integration Language (SMIL 2.0) standard (see the Bibliography).*

13.2.2 Viability

When playing multimedia content, the conforming reader shall often make decisions such as which player software and which options, such as volume and duration, to use.

In making these decisions, the viewer shall determine the *viability* of the objects used. If an object is considered non-viable, the media should not be played. If the object is viable, the media should be played, though possibly under less than optimum conditions.

There are several entries in the multimedia object dictionaries whose values shall have an effect on viability. In particular, some of the object dictionaries define two entries that divide options into one of two categories:

- **MH** *(“must honour”)*: The options specified by this entry shall be honoured; otherwise, the containing object shall be considered non-viable.

- **BE** *(“best effort*”): An attempt should be made to honour the options; however, if they cannot be honoured, the containing object is still considered viable.

MH and **BE** are both dictionaries, and the same entries shall be defined for both of them. In any dictionary where these entries are allowed, both entries may be present, or only one, or neither.

EXAMPLE The media play parameters dictionary (see Table 279) allows the playback volume to be set by means of the **V** entry in its **MH** and **BE** dictionaries (see Table 280).

If the specified volume cannot be honoured, the object shall be considered non-viable if **V** is in the **MH** dictionary, and playback shall not occur. If **V** is in the **BE** dictionary (and not also in the **MH** dictionary), playback should still occur: the playing software attempts to honour the specified option as best it can.

Using this mechanism, authors may specify minimum requirements (**MH**) and preferred options (**BE**). They may also specify how entries that are added in the future to the multimedia dictionaries shall be interpreted by old conforming readers. If an entry that is unrecognized by the viewer is in the **MH** dictionary, the object shall be considered non-viable. If an unrecognized entry is in a **BE** dictionary, the entry shall be ignored and viability shall be unaffected. Unless otherwise stated, an object shall be considered non-viable if its **MH** dictionary contains an unrecognized key or an unrecognized value for a recognized key.

The following rules apply to the entries in **MH** and **BE** dictionaries, which behave somewhat differently from other PDF dictionaries:

- If an entry is required, the requirement is met if the entry is present in either the **MH** dictionary or the **BE** dictionary.

- If an optional entry is not present in either dictionary, it shall be considered to be present with its default value (if one is defined) in the **BE** dictionary.

- If an instance of the same entry is present in both **MH** and **BE**, the instance in the **BE** dictionary shall be ignored unless otherwise specified.
- If the value of an entry in an **MH** or a **BE** dictionary is a dictionary or array, it shall be treated as an atomic unit when determining viability. That is, all entries within the dictionary or array shall be honoured for the containing object to be viable.

NOTE When determining whether entries can be honoured, it is not required that each one be evaluated independently, since they may be dependent on one another. That is, a conforming reader or player may examine multiple entries at once (even within different dictionaries) to determine whether their values can be honoured.

The following media objects may have **MH** and **BE** dictionaries. They function as described previously, except where noted in the individual sub-clauses:

- Rendition (Table 267)
- Media clip data (Table 276)
- Media clip section (Table 278)
- Media play parameters (Table 280)
- Media screen parameters (Table 283)

13.2.3 Renditions

13.2.3.1 General

There are two types of *rendition* objects:

- A *media rendition* (see 13.2.3.2, “Media Renditions”) is a basic media object that specifies what to play, how to play it, and where to play it.
- A *selector rendition* (see 13.2.3.3, “Selector Renditions”) contains an ordered list of renditions. This list may include other selector renditions, resulting in a tree whose leaves are media renditions. The conforming reader should play the first viable media rendition it encounters in the tree (see 13.2.2, “Viability”).

NOTE 1 Table 266 shows the entries common to all rendition dictionaries. The N entry in a rendition dictionary specifies a name that can be used to access the rendition object by means of name tree lookup (see Table 31). JavaScript actions (see 12.6.4.16, “JavaScript Actions”), for example, use this mechanism.

Since the values referenced by name trees shall be indirect objects, all rendition objects should be indirect objects.

NOTE 2 A rendition dictionary is not required to have a name tree entry. When it does, the conforming reader should ensure that the name specified in the tree is kept the same as the value of the ***N*** entry (for example, if the user interface allows the name to be changed). A document should not contain multiple renditions with the same name.

The **MH** and **BE** entries are dictionaries whose entries may be present in one or the other of them, as described in 13.2.2, “Viability.” For renditions, these dictionaries shall have a single entry **C** (see Table 267), whose value shall have a *media criteria dictionary* specifying a set of criteria that shall be met for the rendition to be considered viable (see Table 268).

The media criteria dictionary behaves somewhat differently than other **MH**/**BE** entries, as they are described in 13.2.2, “Viability.” The criteria specified by all of its entries shall be met regardless of whether they are in an **MH** or a **BE** dictionary. The only exception is that if an entry in a **BE** dictionary is *unrecognized* by the conforming reader, it shall not affect the viability of the object. If a media criteria dictionary is present in both **MH** and **BE**, the entries in both dictionaries shall be individually evaluated, with **MH** taking precedence (corresponding **BE** entries shall be ignored).

Table 266 – Entries common to all rendition dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that dictionary describes; if present, shall be **Rendition** for a rendition object.
S	name	*(Required)* The type of rendition that this dictionary describes. May be **MR** for media rendition or **SR** for selector rendition. The rendition shall be considered non-viable if the conforming reader does not recognize the value of this entry.
N	text string	*(Optional)* A Unicode-encoded text string specifying the name of the rendition for use in a user interface and for name tree lookup by JavaScript actions.
MH	dictionary	*(Optional)* A dictionary whose entries (see Table 267) shall be honoured for the rendition to be considered viable.
BE	dictionary	*(Optional)* A dictionary whose entries (see Table 267) shall only be honoured in a “best effort” sense.

Table 267 – Entries in a rendition MH/BE dictionary

Key	Type	Value
C	dictionary	*(Optional)* A *media criteria* dictionary (see Table 268). The media criteria dictionary behaves somewhat differently than other ***MH***/***BE*** entries described in 13.2.2, “Viability.” The criteria specified by all of its entries shall be met regardless of whether it is in an ***MH*** or a ***BE*** dictionary. The only exception is that if an *entry in a* ***BE*** *dictionary is* unrecognized *by the* conforming reader, it shall not affect the viability of the object.

Table 268 – Entries in a media criteria dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaCriteria** for a media criteria dictionary.
A	boolean	*(Optional)* If specified, the value of this entry shall match the user’s preference for whether to hear audio descriptions in order for this object to be viable. NOTE 1 Equivalent to SMIL’s *systemAudioDesc* attribute.
C	boolean	*(Optional)* If specified, the value of this entry shall match the user’s preference for whether to see text captions in order for this object to be viable. NOTE 2 Equivalent to SMIL’s *systemCaptions* attribute.
O	boolean	*(Optional)* If specified, the value of this entry shall match the user’s preference for whether to hear audio overdubs in order for this object to be viable.
S	boolean	*(Optional)* If specified, the value of this entry shall match the user’s preference for whether to see subtitles in order for this object to be viable.

Table 268 – Entries in a media criteria dictionary (continued)

Key	Type	Value
R	integer	*(Optional)* If specified, the system's bandwidth (in bits per second) shall be greater than or equal to the value of this entry in order for this object to be viable. NOTE 3 Equivalent to SMIL's *systemBitrate* attribute.
D	dictionary	*(Optional)* A dictionary (see Table 269) specifying the minimum bit depth required in order for this object to be viable. NOTE 4 Equivalent to SMIL's *systemScreenDepth* attribute.
Z	dictionary	*(Optional)* A dictionary (see Table 270) specifying the minimum screen size required in order for this object to be viable. NOTE 5 Equivalent to SMIL's *systemScreenSize* attribute.
V	array	*(Optional)* An array of software identifier objects (see 13.2.7.4, "Software Identifier Dictionary"). If this entry is present and non-empty, the conforming reader shall be identified by one or more of the objects in the array in order for this object to be viable.
P	array	*(Optional)* An array containing one or two name objects specifying a minimum and optionally a maximum PDF language version, in the same format as the **Version** entry in the document catalog (see Table 28). If this entry is present and non-empty, the version of multimedia constructs fully supported by the conforming reader shall be within the specified range in order for this object to be viable.
L	array	*(Optional)* An array of language identifiers (see 14.9.2.2, "Language Identifiers"). If this entry is present and non-empty, the language in which the conforming reader is running shall exactly match a language identifier, or consist only of a primary code that matches the primary code of an identifier, in order for this object to be viable. NOTE 6 Equivalent to SMIL's *systemLanguage* attribute.

Table 269 – Entries in a minimum bit depth dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MinBitDepth** for a minimum bit depth dictionary.
V	integer	*(Required)* A positive integer (0 or greater) specifying the minimum screen depth (in bits) of the monitor for the rendition to be viable. A negative value shall not be allowed.
M	integer	*(Optional)* A monitor specifier (see Table 270) that specifies which monitor the value of **V** should be tested against. If the value is unrecognized, the object shall not be viable. Default value: 0.

Table 270 – Entries in a minimum screen size dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MinScreenSize** for a rendition object.
V	array	*(Required)* An array containing two non-negative integers. The width and height (in pixels) of the monitor specified by M shall be greater than or equal to the values of the first and second integers in the array, respectively, in order for this object to be viable.
M	integer	*(Optional)* A monitor specifier (see Table 293) that specifies which monitor the value of **V** should be tested against. If the value is unrecognized, the object shall be not viable. Default value: 0.

13.2.3.2 Media Renditions

Table 271 lists the entries in a media rendition dictionary. Its entries specify what media should be played (**C**), how (**P**), and where (**SP**) it should be played. A media rendition object shall be viable if and only if the objects referenced by its **C**, **P**, and **SP** entries are viable.

C may be omitted only in cases where a referenced player takes no meaningful input. This requires that **P** shall be present and that its referenced media play parameters dictionary (see Table 279) shall contain a **PL** entry, whose referenced media players dictionary (see 13.2.7.2, "Media Players Dictionary") has a non-empty **MU** array or a non-empty **A** array.

Table 271 – Additional entries in a media rendition dictionary

Key	Type	Value
C	dictionary	*(Optional)* A *media clip* dictionary (see 13.2.4, "Media Clip Objects") that specifies what should be played when the media rendition object is played.
P	dictionary	*(Required if **C** is not present, otherwise optional)* A *media play parameters* dictionary (see 13.2.5, "Media Play Parameters"") that specifies how the media rendition object should be played. Default value: a media play parameters dictionary whose entries (see Table 279) all contain their default values.
SP	dictionary	*(Optional)* A *media screen parameters* dictionary (see 13.2.6, "Media Screen Parameters") that specifies where the media rendition object should be played. Default value: a media screen parameters dictionary whose entries (see Table 282) all contain their default values.

13.2.3.3 Selector Renditions

A *selector rendition dictionary* shall specify an array of rendition objects in its **R** entry (see Table 272). The renditions in this array should be ordered by preference, with the most preferred rendition first. At play-time, the renditions in the array shall be evaluated and the first viable media rendition, if any, shall be played. If one of the renditions is itself a selector, that selector shall be evaluated in turn, yielding the equivalent of a depth-first tree search. A selector rendition itself may be non-viable; in this case, none of its associated media renditions shall be evaluated (in effect, this branch of the tree is skipped).

NOTE This mechanism may be used, for example, to specify that a large video clip should be used on high-bandwidth machines and a smaller clip should be used on low-bandwidth machines.

Table 272 – Additional entries specific to a selector rendition dictionary

Key	Type	Value
R	array	*(Required)* An array of rendition objects. The first viable media rendition object found in the array, or nested within a selector rendition in the array, should be used. An empty array is legal.

13.2.4 Media Clip Objects

13.2.4.1 General

There are two types of media clip objects, determined by the subtype **S**, which can be either **MCD** for media clip data (see 13.2.4.2, “Media Clip Data”) or **MCS** for media clip section (see 13.2.4.3, “Media Clip Section”). The entries common to all media clip dictionaries are listed in Table 273.

Table 273 – Entries common to all media clip dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaClip** for a media clip dictionary.
S	name	*(Required)* The subtype of media clip that this dictionary describes. May be **MCD** for media clip data (see 13.2.4.2, “Media Clip Data”) or **MCS** for a media clip section (see 13.2.4.3, “Media Clip Section”). The media clip shall be considered non-viable if the conforming reader does not recognize the value of this entry.
N	text string	*(Optional)* The name of the media clip, for use in the user interface.

13.2.4.2 Media Clip Data

A *media clip data dictionary* defines the data for a media object that can be played. Its entries are listed in Table 274

NOTE 1 It may reference a URL to a streaming video presentation or a movie embedded in the PDF file.

Table 274 – Additional entries in a media clip data dictionary

Key	Type	Value
D	file specification or stream	*(Required)* A full file specification or form XObject that specifies the actual media data.
CT	ASCII string	*(Optional; not allowed for form XObjects)* An ASCII string identifying the type of data in **D**. The string should conform to the content type specification described in Internet RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* (see the Bibliography).
P	dictionary	*(Optional)* A *media permissions dictionary* (see Table 275) containing permissions that control the use of the media data. Default value: a media permissions dictionary containing default values.
Alt	array	*(Optional)* An array that provides alternate text descriptions for the media clip data in case it cannot be played; see 14.9.2.4, “Multi-language Text Arrays.”

Table 274 – Additional entries in a media clip data dictionary (continued)

Key	Type	Value
PL	dictionary	*(Optional)* A *media players dictionary* (see 13.2.7.2, “Media Players Dictionary”) that identifies, among other things, players that are legal and not legal for playing the media. If the media players dictionary is non-viable, the media clip data shall be non-viable.
MH	dictionary	*(Optional)* A dictionary whose entries (see Table 276) shall be honoured for the media clip data to be considered viable.
BE	dictionary	*(Optional)* A dictionary whose entries (see Table 276) should only be honoured in a “best effort” sense.

The media clip data object shall be considered non-viable if the object referenced by the **D** entry does not contain a **Type** entry, the **Type** entry is unrecognized, or the referenced object is not a dictionary or stream.

This shall effectively exclude the use of simple file specifications (see 7.11, “File Specifications”).

If **D** references a file specification that has an embedded file stream (see 7.11.4, “Embedded File Streams”), the embedded file stream’s **Subtype** entry shall be ignored if present, and the media clip data dictionary’s **CT** entry shall identify the type of data.

If **D** references a form XObject, the associated player is implicitly the conforming reader, and the form XObject shall be rendered as if it were any other data type.

NOTE 2 The **F** and **D** entries in the media play parameters dictionary (see Table 279) should apply to a form XObject just as they do to a QuickTime movie.

For media other than form XObjects, the media clip object shall provide enough information to allow a conforming reader to locate an appropriate player. This may be done by providing one or both of the following entries, the first being the preferred method:

- A **CT** entry that specifies the content type of the media. If this entry is present, any player that is selected shall support this content type.
- A **PL** entry that specifies one or more players that may be used to play the referenced media. if **CT** is present, there should also be a **PL** present.

The **P** entry specifies a *media permissions dictionary* (see Table 275) specifying the manner in which the data referenced by the media may be used by a conforming reader. These permissions allow authors control over how their data is exposed to operations that could allow it to be copied. If the dictionary contains unrecognized entries or entries with unrecognized values, it shall be considered non-viable, and the conforming reader shall not play the media.

Table 275 – Entries in a media permissions dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaPermissions** for a media permissions dictionary.
TF	ASCII string	*(Optional)* An ASCII string indicating the circumstances under which it is acceptable to write a temporary file in order to play a media clip. Valid values are: (TEMPNEVER) Never allowed. (TEMPEXTRACT) Allowed only if the document permissions allow content extraction; when bit 5 of the user access permissions (see Table 22) is set. (TEMPACCESS) Allowed only if the document permissions allow content extraction, including for accessibility purposes; when bits 5 or 10 of the user access permissions (see Table 22) are set, or both. (TEMPALWAYS) Always allowed. Default value: (TEMPNEVER). An unrecognized value shall be treated as (TEMPNEVER).

The **BU** entry in the media clip data **MH** and **BE** dictionaries (see Table 276) specifies a base URL for the media data. Relative URLs in the media (which point to auxiliary files or are used for hyperlinking) should be resolved with respect to the value of **BU**. The following are additional requirements concerning the **BU** entry:

- If **BU** is in the **MH** dictionary and the base URL is not honoured the media clip data shall be non-viable.

NOTE 3 An example of this is that the player does not accept base URLs.

- Determining the viability of the object shall not require checking whether the base URL is valid

NOTE 4 The target host exists.

- Absolute URIs within the media shall not be affected.
- If the media itself contains a base URL, that value shall be used in preference to **BU**.

NOTE 5 An example of this is that the <BASE> element is defined in HTML.

- **BU** is completely independent of and unrelated to the value of the **URI** entry in the document catalogue (see 7.7.2, "Document Catalog").
- If **BU** is not present and the media is embedded within the document, the URL to the PDF file itself shall be used as if it were the value of a **BU** entry in the **BE** dictionary; that is, as an implicit best-effort base URL.

Table 276 – Entries in a media clip data MH/BE dictionary

Key	Type	Value
BU	ASCII string	*(Optional)* An absolute URL that shall be used as the base URL in resolving any relative URLs found within the media data.

13.2.4.3 Media Clip Section

A *media clip section* dictionary (see Table 277) defines a continuous section of another media clip object (known as the *next-level* media clip object). The next-level media clip object, specified by the **D** entry, may be either a media clip data object or another media clip section object. However, the linked list formed by the **D** entries of media clip sections shall terminate in a media clip data object. If the next-level media object is non-viable, the media clip section shall be also non-viable.

NOTE 1 A media clip section could define a 15-minute segment of a media clip data object representing a two-hour movie.

Table 277 – Additional entries in a media clip section dictionary

Key	Type	Value
D	dictionary	*(Required)* The media clip section or media clip data object (the next-level media object) of which this media clip section object defines a continuous section.
Alt	array	*(Optional)* An array that provides alternate text descriptions for the media clip section in case it cannot be played; see 14.9.2.4, “Multi-language Text Arrays.”
MH	dictionary	*(Optional)* A dictionary whose entries (see Table 278) shall be honoured for the media clip section to be considered viable.
BE	dictionary	*(Optional)* A dictionary whose entries (see Table 278) shall only be honoured in a “best effort” sense.

The **B** and **E** entries in the media clip section’s **MH** and **BE** dictionaries (see Table 278) shall define a subsection of the next-level media object referenced by **D** by specifying beginning and ending offsets into it. Depending on the media type, the offsets may be specified by time, frames, or markers (see 13.2.6.2, “Media Offset Dictionary”). **B** and **E** are not required to specify the same type of offset.

The following rules apply to these offsets:

- For media types where an offset makes no sense (such as JPEG images), **B** and **E** shall be ignored, with no effect on viability.
- When **B** or **E** are specified by time or frames, their value shall be considered to be relative to the start of the next-level media clip. However, if **E** specifies an offset beyond the end of the next-level media clip, the end value shall be used instead, and there is no effect on viability.
- When **B** or **E** are specified by markers, there shall be a corresponding absolute offset into the underlying media clip data object. If this offset is not within the range defined by the next-level media clip (if any), or if the marker is not present in the underlying media clip, the existence of the entry shall be ignored, and there is no effect on viability.
- If the absolute offset derived from the values of all **B** entries in a media clip section chain is greater than or equal to the absolute offset derived from the values of all **E** entries, an empty range shall be defined. An empty range is legal.
- Any **B** or **E** entry in a media clip section’s **MH** dictionary shall be honoured at play-time in order for the media clip section to be considered viable.

NOTE 2 The entry may not be honored if its value was not viable or if the player did not support its value; for example, the player did not support markers.

- If a **B** or **E** entry is in a media clip section’s **MH** dictionary, all **B** or **E** entries, respectively, at deeper levels (closer to the media clip data), shall be evaluated as if they were in an **MH** dictionary (even if they are actually within **BE** dictionaries).
- If **B** or **E** entry in a **BE** dictionary cannot be supported, it may be ignored at play-time.

Table 278 – Entries in a media clip section MH/BE dictionary

Key	Type	Value
B	dictionary	*(Optional)* A *media offset dictionary* (see 13.2.6.2, “Media Offset Dictionary”) that specifies the offset into the next-level media object at which the media clip section begins. Default: the start of the next-level media object.
E	dictionary	*(Optional)* A *media offset dictionary* (see 13.2.6.2, “Media Offset Dictionary”) that specifies the offset into the next-level media object at which the media clip section ends. Default: the end of the next-level media object.

13.2.5 Media Play Parameters

A media play parameters dictionary specifies how a media object should be played. It shall be referenced from a media rendition (see 13.2.3.2, “Media Renditions”).

Table 279 – Entries in a media play parameters dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaPlayParams** for a media play parameters dictionary.
PL	dictionary	*(Optional)* A media players dictionary (see 13.2.7.2, “Media Players Dictionary”) that identifies, among other things, players that are legal and not legal for playing the media. If this object is non-viable, the media play parameters dictionary shall be considered non-viable.
MH	dictionary	*(Optional)* A dictionary whose entries (see Table 278) shall be honoured for the media play parameters to be considered viable.
BE	dictionary	*(Optional)* A dictionary whose entries (see Table 278) shall only be honoured in a “best effort” sense.

Table 280 – Entries in a media play parameters MH/BE dictionary

Key	Type	Value
V	integer	*(Optional)* An integer that specifies the desired volume level as a percentage of recorded volume level. A zero value shall be equivalent to mute; negative values shall be illegal. Default value: 100.
C	boolean	*(Optional)* A flag specifying whether to display a player-specific controller user interface when playing. EXAMPLE play/pause/stop controls. Default value: **false**

Table 280 – Entries in a media play parameters MH/BE dictionary (continued)

Key	Type	Value
F	integer‘	*(Optional)* The manner in which the player shall treat a visual media type that does not exactly fit the rectangle in which it plays. 0The media’s width and height shall be scaled while preserving the aspect ratio so that the media and play rectangles have the greatest possible intersection while still displaying all media content. NOTE 1 Same as “meet” value of SMIL’s *fit* attribute. 1 The media’s width and height shall be scaled while preserving the aspect ratio so that the play rectangle is entirely filled, and the amount of media content that does not fit within the play rectangle shall be minimized. NOTE 2 Same as “slice” value of SMIL’s *fit* attribute. 2 The media’s width and height shall be scaled independently so that the media and play rectangles are the same; the aspect ratio shall not be preserved. NOTE 3 Same as “fill” value of SMIL’s *fit* attribute. 3 The media shall not be scaled. A scrolling user interface shall be provided if the media rectangle is wider or taller than the play rectangle. NOTE 4 Same as “scroll” value of SMIL’s *fit* attribute. 4 The media shall not be scaled. Only the portions of the media rectangle that intersect the play rectangle shall be displayed. NOTE 5 Same as “hidden” value of SMIL’s *fit* attribute. 5 Use the player’s default setting (author has no preference). Default value: 5. An unrecognized value shall be treated as the default value if the entry is in a **BE** dictionary. If the entry is in an **MH** dictionary and it has an unrecognized value, the object shall be considered non-viable.
D	dictionary	*(Optional)* A media duration dictionary (see Table 281). Default value: a dictionary specifying the intrinsic duration (see **RC**).
A	boolean	*(Optional)* If **true**, the media shall automatically play when activated. If **false**, the media shall be initially paused when activated. EXAMPLE The first frame is displayed. Relevant only for media that may be paused. Default value: **true**.
RC	number	*(Optional)* Specifies the number of iterations of the duration **D** to repeat. NOTE 6 Similar to SMIL’s *repeatCount* attribute. Zero means repeat forever. Negative values shall be illegal; non-integral values shall be legal. Default value: 1.0.

The value of the **D** entry is a *media duration dictionary*, whose entries are shown in Table 281. It specifies a temporal duration.

NOTE 1 The D entry dictionary temporal duration corresponds to the notion of a simple duration in SMIL.

The duration may be a specific amount of time, it may be infinity, or it may be the media’s *intrinsic duration.*

EXAMPLE The intrinsic duration of a two-hour QuickTime movie is two hours.

The intrinsic duration may be modified when a media clip section (see 13.2.4.3, “Media Clip Section”) is used: the intrinsic duration shall be the difference between the absolute begin and end offsets. For a media type having no notion of time (such as a JPEG image), the duration shall be considered to be infinity.

If the simple duration is longer than the intrinsic duration, the player shall freeze the media in its final state until the simple duration has elapsed. For visual media types, the last appearance (frame) shall be displayed. For aural media types, the media is logically frozen but shall not continue to produce sound.

NOTE 2 In this case, the ***RC*** entry, which specifies a repeat count, applies to the simple duration; therefore, the entire play-pause sequence is repeated ***RC*** times.

Table 281 – Entries in a media duration dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaDuration** for a media duration dictionary.
S	name	*(Required)* The subtype of media duration dictionary. Valid values are: **I** The duration is the intrinsic duration of the associated media **F** The duration is infinity **T** The duration shall be specified by the **T** entry The media duration dictionary shall be considered non-viable if the conforming reader does not recognize the value of this entry.
T	dictionary	*(Required if the value of **S** is **T**; otherwise ignored)* A timespan dictionary specifying an explicit duration (see Table 289). A negative duration is illegal.

13.2.6 Media Screen Parameters

13.2.6.1 General

A media screen parameters dictionary (see Table 282) shall specify where a media object should be played. It shall contain **MH** and **BE** dictionaries (see Table 283), which shall function as discussed in 13.2.2, “Viability.” All media clips that are being played shall be associated with a particular document and shall be stopped when the document is closed.

NOTE Conforming readers should disallow floating windows and full-screen windows unless specifically allowed by the user. The reason is that document-based security attacks are possible if windows containing arbitrary media content can be displayed without indicating to the user that the window is merely hosting a media object. This recommendation may be relaxed if it is possible to communicate the nature of such windows to the user; for example, with text in a floating window’s title bar.

Table 282 – Entries in a media screen parameters dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaScreenParams** for a media screen parameters dictionary.
MH	dictionary	*(Optional)* A dictionary whose entries (see Table 283) shall be honoured for the media screen parameters to be considered viable.
BE	dictionary	*(Optional)* A dictionary whose entries (see Table 283) should be honoured.

Table 283 – Entries in a media screen parameters MH/BE dictionary

Key	Type	Value
W	integer	*(Optional)* The type of window that the media object shall play in: 0 A floating window 1 A full-screen window that obscures all other windows 2 A hidden window 3 The rectangle occupied by the screen annotation (see 12.5.6.18, "Screen Annotations") associated with the media rendition Default value: 3. Unrecognized value in **MH**: object is non-viable; in **BE**: treat as default value.
B	array	*(Optional)* An array of three numbers in the range 0.0 to 1.0 that shall specify the components in the **DeviceRGB** colour space of the background colour for the rectangle in which the media is being played. This colour shall be used if the media object does not entirely cover the rectangle or if it has transparent sections. It shall be ignored for hidden windows. Default value: implementation-defined. The conforming reader should choose a reasonable value based on the value of **W**. EXAMPLE 1 A system default background colour for floating windows or a user-preferred background colour for full-screen windows. If a media format has an intrinsic background colour, ***B*** shall not override it. However, the ***B*** colour shall be visible if the media has transparent areas or otherwise does not cover the entire window.
O	number	*(Optional)* A number in the range 0.0 to 1.0 specifying the constant opacity value that shall be used in painting the background colour specified by **B**. A value below 1.0 means the window shall be transparent. EXAMPLE 2 Windows behind a floating window show through if the media does not cover the entire floating window. A value of 0.0 shall indicate full transparency and shall make **B** irrelevant. It shall be ignored for full-screen and hidden windows. Default value: 1.0 (fully opaque).
M	integer	*(Optional)* A monitor specifier (see Table 293) that shall specify which monitor in a multi-monitor system, a floating or full-screen window shall appear on. Ignored for other types. Default value: 0 (document monitor). Unrecognized value in **MH**: object is non-viable; in **BE**: treat as default value.
F	dictionary	*(Required if the value of **W** is 0; otherwise ignored)* A floating window parameters dictionary (see Table 284) that shall specify the size, position, and options used in displaying floating windows.

The **F** entry in the media screen parameters **MH**/**BE** dictionaries shall be a floating window parameters dictionary, whose entries are listed in Table 284. The entries in the floating window parameters dictionary shall be treated as if they were present in the **MH** or **BE** dictionaries that they are referenced from. That is, the contained entries shall be individually evaluated for viability rather than the dictionary being evaluated as a whole. (There may be an **F** entry in both **MH** and **BE**. In such a case, if a given entry is present in both floating window parameters dictionaries, the one in the **MH** dictionary shall take precedence.)

The **D**, **P**, and **RT** entries shall be used to specify the rectangle that the floating window occupies. Once created, the floating window's size and position shall not be tied to any other window, even if the initial size or position was computed relative to other windows.

Unrecognized values for the **R**, **P**, **RT**, and **O** entries shall be handled as follows: if they are nested within an **MH** dictionary, the floating window parameters object (and hence the media screen parameters object) shall be considered non-viable; if they are nested within a **BE** dictionary, they shall be considered to have their default values.

Table 284 – Entries in a floating window parameters dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **FWParams** for a floating window parameters dictionary.
D	array	*(Required)* An array containing two non-negative integers that shall represent the floating window's width and height, in pixels, respectively. These values shall correspond to the dimensions of the rectangle in which the media shall play, not including such items as title bar and resizing handles.
RT	integer	*(Optional)* The window relative to which the floating window shall be positioned: 0 The document window 1 The application window 2 The full virtual desktop 3 The monitor specified by **M** in the media screen parameters **MH** or **BE** dictionary (see 9.22) Default value: 0.
P	integer	*(Optional)* The location where the floating window (including such items as title bar and resizing handles) shall be positioned relative to the window specified by **RT**: 0 Upper-left corner 1 Upper center 2 Upper-right corner 3 Center left 4 Center 5 Center right 6 Lower-left corner 7 Lower center 8 Lower-right corner Default value: 4.
O	integer	*(Optional)* Specifies what shall occur if the floating window is positioned totally or partially offscreen (that is, not visible on any physical monitor): 0 Take no special action 1 Move and/or resize the window so that it is on-screen 2 Consider the object to be non-viable Default value: 1
T	boolean	*(Optional)* If **true**, the floating window shall have a title bar. Default value: **true**.
UC	boolean	*(Optional; meaningful only if **T** is **true**)* If **true**, the floating window shall include user interface elements that allow a user to close a floating window. Default value: **true**
R	integer	*(Optional)* Specifies whether the floating window may be resized by a user: 0 May not be resized 1 May be resized only if aspect ratio is preserved 2 May be resized without preserving aspect ratio Default value: 0.

Table 284 – Entries in a floating window parameters dictionary (continued)

Key	Type	Value
TT	array	*(Optional; meaningful only if **T** is **true**)* An array providing text to display on the floating window's title bar. See 14.9.2.4, "Multi-language Text Arrays." If this entry is not present, the conforming reader may provide default text.

13.2.6.2 Media Offset Dictionary

A *media offset dictionary* (Table 285) shall specify an offset into a media object. The **S** (subtype) entry indicates how the offset shall be specified: in terms of time, frames or markers. Different media types support different types of offsets.

EXAMPLE Time, "10 seconds"; frames, "frame 20"; markers, "Chapter One."

Table 285 – Entries common to all media offset dictionaries

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaOffset** for a media offset dictionary.
S	name	*(Required)* The subtype of media offset dictionary. Valid values shall be: T A media offset time dictionary (see Table 286) F A media offset frame dictionary (see Table 287) M A media offset marker dictionary (see Table 288) The rendition shall be considered non-viable if the conforming reader does not recognize the value of this entry.

Table 286 – Additional entries in a media offset time dictionary

Key	Type	Value
T	dictionary	*(Required)* A timespan dictionary (see Table 289) that shall specify a temporal offset into a media object. Negative timespans are not allowed in this context. The media offset time dictionary is non-viable if its timespan dictionary is non-viable.

Table 287 – Additional entries in a media offset frame dictionary

Key	Type	Value
F	integer	*(Required)* Shall specify a frame within a media object. Frame numbers begin at 0; negative frame numbers are not allowed.

Table 288 – Additional entries in a media offset marker dictionary

Key	Type	Value
M	text string	*(Required)* A text string that identifies a named offset within a media object.

13.2.6.3 Timespan Dictionary

A *timespan dictionary* shall specify a length of time; its entries are shown in Table 289.

Table 289 – Entries in a timespan dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Timespan** for a timespan dictionary.
S	name	*(Required)* The subtype of timespan dictionary. The value shall be **S** (simple timespan). The rendition shall be considered non-viable if the conforming reader does not recognize the value of this entry.
V	number	*(Required)* The number of seconds in the timespan. Non-integral values shall be allowed. Negative values shall be allowed, but may be disallowed in some contexts. *(PDF 1.5)* Negative values are not allowed. This entry shall be used only if the value of the ***S*** entry is ***S***. Subtypes defined in the future need not use this entry.

13.2.7 Other Multimedia Objects

13.2.7.1 General

This sub-clause defines several dictionary types that are referenced by the previous sub-clauses.

13.2.7.2 Media Players Dictionary

A media players dictionary may be referenced by media clip data (see 13.2.4.2, “Media Clip Data”) and media play parameters (see 13.2.5, “Media Play Parameters”) dictionaries, and shall allow them to specify which players may or may not be used to play the associated media. The media players dictionary references *media player info dictionaries* (see 13.2.7.3, “Media Player Info Dictionary”) that shall provide specific information about each player.

Table 290 – Entries in a media players dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaPlayers** for a media players dictionary.
MU	array	*(Optional)* An array of media player info dictionaries (see Table 291) that shall specify a set of players, one of which shall be used in playing the associated media object. Any players specified in ***NU*** are effectively removed from ***MU***. EXAMPLE If ***MU*** specifies versions 1 through 5 of a player and ***NU*** specifies versions 1 and 2 of the same player, ***MU*** is effectively versions 3 through 5.
A	array	*(Optional)* An array of media player info dictionaries (see Table 291) that shall specify a set of players, any of which *may* be used in playing the associated media object. If **MU** is also present and non-empty, **A** shall be ignored.
NU	array	*(Optional)* An array of media player info dictionaries (see Table 291) that shall specify a set of players that shall *not* be used in playing the associated media object (even if they are also specified in **MU**).

The **MU**, **A**, and **NU** entries each shall specify one or more media player info dictionaries. An empty array shall be treated as if it is not present. The media player info dictionaries shall be allowed to specify overlapping player ranges.

NOTE 1 **MU** could contain a media player info dictionary describing versions 1 to 10 of Player X and another describing versions 3 through 5 of Player X.

If a non-viable media player info dictionary is referenced by **MU**, **NU**, or **A**, it shall be treated as if it were not present in its original array, and a media player info dictionary containing the same software identifier dictionary (see 13.2.7.4, “Software Identifier Dictionary”) shall logically considered present in **NU**. The same rule shall apply to a media player info dictionary that contains a partially unrecognized software identifier dictionary.

Since both media clip data and media play parameters dictionaries may be employed in a play operation, and each may reference a media players dictionary, there is a potential for conflict between the contents of the two media players dictionaries. At play-time, the viewer shall use the following algorithm to determine whether a player present on the machine may be employed. The player may not be used if any of the following conditions are true:

Algorithm: Media Player

a) The content type is known and the player does not support the type.

b) The player is found in the **NU** array of either dictionary.

c) Both dictionaries have non-empty **MU** arrays and the player is not found in both of them, or only one of the dictionaries has a non-empty **MU** array and the player is not found in it.

d) Neither dictionary has a non-empty **MU** array, the content type is not known, and the player is not found in the **A** array of either dictionary.

If none of the conditions are true, the player may be used.

NOTE 2 A player is “found” in the ***NU***, ***MU***, or ***A*** arrays if it matches the information found in the PID entry of one of the entries, as described by the Algorithm in 13.2.7.4, “Software Identifier Dictionary.”

13.2.7.3 Media Player Info Dictionary

A *media player info dictionary* shall provide a variety of information regarding a specific media player. Its entries (see Table 291) shall associate information with a particular version or range of versions of a player. As of PDF 1.5, only the **PID** entry shall provide information about the player, as described in the next sub-clause, 13.2.7.4, “Software Identifier Dictionary.”

Table 291 – Entries in a media player info dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **MediaPlayerInfo** for a media player info dictionary.
PID	dictionary	*(Required)* A software identifier dictionary (see 13.2.7.4, “Software Identifier Dictionary”) that shall specify the player name, versions, and operating systems to which this media player info dictionary applies.
MH	dictionary	*(Optional)* A dictionary containing entries that shall be honored for this dictionary to be considered viable Currently, there are no defined entries for this dictionary
BE	dictionary	*(Optional)* A dictionary containing entries that need only be honored in a “best effort” sense. Currently, there are no defined entries for this dictionary

13.2.7.4 Software Identifier Dictionary

13.2.7.4.1 General

A *software identifier dictionary* shall allow software to be identified by name, range of versions, and operating systems; its entries are listed in Table 292. A conforming reader uses this information to determine whether a given media player may be used in a given situation. If the dictionary contains keys that are unrecognized by the conforming reader, it shall be considered to be partially recognized. The conforming reader may or may not decide to treat the software identifier as viable, depending on the context in which it is used.

The following procedure shall be used to determine whether a piece of software is considered to match a software identifier dictionary:

Algorithm: Software identifier

a) The software name shall match the name specified by the **U** entry (see "Software URIs" in 13.2.7.4, "Software Identifier Dictionary").

b) The software version shall be within the interval specified by the **L**, **H**, **LI**, and **H1** entries (see "Version arrays" in 13.2.7.4, "Software Identifier Dictionary").

c) The machine's operating system name shall be an exact match for one present in the **OS** array. If the array is not present or empty, a match shall also be considered to exist.

Table 292 – Entries in a software identifier dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SoftwareIdentifier** for a software identifier dictionary.
U	ASCII string	*(Required)* A URI that identifies a piece of software (see "Software URIs" in 13.2.7.4, "Software Identifier Dictionary").
L	array	*(Optional)* The lower bound of the range of software versions that this software identifier dictionary specifies (see "Version arrays" in 13.2.7.4, "Software Identifier Dictionary"). Default value: the array [0].
LI	boolean	*(Optional)* If **true**, the lower bound of the interval defined by **L** and **H** is inclusive; that is, the software version shall be greater than or equal to **L** (see "Version arrays" in 13.2.7.4, "Software Identifier Dictionary"). If **false**, it shall not be inclusive. Default value: **true**.
H	array	*(Optional)* The upper bound of the range of software versions that this software identifier dictionary specifies (see "Version arrays" in 13.2.7.4, "Software Identifier Dictionary"). Default value: an empty array [].
HI	boolean	*(Optional)* If **true**, the upper bound of the interval defined by **L** and **H** is inclusive; that is, the software version shall be less than or equal to **H** (see "Version arrays" in 13.2.7.4, "Software Identifier Dictionary"). If **false**, it shall not be inclusive. Default value: **true**.
OS	array	*(Optional)* An array of byte strings representing operating system identifiers that shall indicate to which operating systems this object applies. The defined values are the same as those defined for SMIL 2.0's *systemOperatingSystem* attribute. There may not be multiple copies of the same identifier in the array. An empty array shall be considered to represent all operating systems. Default value: an empty array.

13.2.7.4.2 Software URIs

The **U** entry is a URI (universal resource identifier) that identifies a piece of software. It shall be interpreted according to its scheme; the only presently defined scheme is *vnd.adobe.swname*. The scheme name is case-insensitive; if shall not be recognized by the conforming reader, the software shall be considered a non-match. The syntax of URIs of this scheme is

"vnd.adobe.swname:" *software_name*

where *software_name* shall be *reg_name* as defined in Internet RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*; see the Bibliography. *software_name* shall be a sequence of UTF-8-encoded characters that have been escaped with one pass of URL escaping (see 14.10.3.2, "URL Strings"). That is, to recover the original software name, *software_name* shall be unescaped and then treated as a sequence of UTF-8 characters. The actual software names shall be compared in a case-sensitive fashion.

Software names shall be second-class names (see Annex E).

EXAMPLE The URI for Adobe Acrobat is

vnd.adobe.swname:ADBE_Acrobat

13.2.7.4.3 Version arrays

The **L**, **H**, **LI**, and **HI** entries shall be used to specify a range of software versions. **L** and H shall be *version arrays* containing zero or more non-negative integers representing subversion numbers. The first integer shall be the major version numbers, and subsequent integers shall be increasingly minor. **H** shall be greater than or equal to **L**, according to the following rules for comparing version arrays:

Algorithm: Comparing version arrays

a) An empty version array shall be treated as infinity; that is, it shall be considered greater than any other version array except another empty array. Two empty arrays are equal.

b) When comparing arrays that contain different numbers of elements, the smaller array shall be implicitly padded with zero-valued integers to make the number of elements equal.

EXAMPLE When comparing [5 1 2 3 4] to [5], the latter is treated as [5 0 0 0 0].

c) The corresponding elements of the arrays shall be compared, starting with the first. When a difference is found, the array containing the larger element shall be considered to have the larger version number. If no differences are found, the versions are equal.

If a version array contains negative numbers, it shall be considered non-viable, as is the enclosing software identifier.

13.2.7.5 Monitor Specifier

A *monitor specifier* is an integer that shall identify a physical monitor attached to a system. It may have one of the values in Table 293:

Table 293 – Monitor specifier values

Value	Description
0	The monitor containing the largest section of the document window
1	The monitor containing the smallest section of the document window
2	Primary monitor. If no monitor is considered primary, shall treat as case 0
3	Monitor with the greatest colour depth
4	Monitor with the greatest area (in pixels squared)
5	Monitor with the greatest height (in pixels)
6	Monitor with the greatest width (in pixels)

For some of these values, it is possible have a “tie” at play-time; for example, two monitors might have the same colour depth. Ties may be broken in an implementation-dependent manner.

13.3 Sounds

A *sound object (PDF 1.2)* shall be a stream containing sample values that define a sound to be played through the computer’s speakers. The **Sound** entry in a sound annotation or sound action dictionary (see Table 185 and Table 208) shall identify a sound object representing the sound to be played when the annotation is activated.

Since a sound object is a stream, it may contain any of the standard entries common to all streams, as described in Table 5. In particular, if it contains an **F** (file specification) entry, the sound shall be defined in an external file. This sound file shall be self-describing, containing all information needed to render the sound; no additional information need be present in the PDF file.

NOTE The AIFF, AIFF-C (Mac OS), RIFF (*.wav*), and snd (*.au*) file formats are all self-describing.

If no **F** entry is present, the sound object itself shall contain the sample data and all other information needed to define the sound. Table 294 shows the additional dictionary entries specific to a sound object.

Table 294 – Additional entries specific to a sound object

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **Sound** for a sound object.
R	number	*(Required)* The sampling rate, in samples per second.
C	integer	*(Optional)* The number of sound channels. Default value: 1.
B	integer	*(Optional)* The number of bits per sample value per channel. Default value: 8.
E	name	*(Optional)* The encoding format for the sample data: Raw Unspecified or unsigned values in the range 0 to 2**B** – 1 Signed Twos-complement values muLaw m-law–encoded samples ALaw A-law–encoded samples Default value: Raw.

Table 294 – Additional entries specific to a sound object (continued)

Key	Type	Value
CO	name	*(Optional)* The sound compression format used on the sample data. (This is separate from any stream compression specified by the sound object's **Filter** entry; see Table 5 and 7.4, "Filters.") If this entry is absent, sound compression shall not be used; the data contains sampled waveforms that shall be played at **R** samples per second per channel.
CP	(various)	*(Optional)* Optional parameters specific to the sound compression format used. No standard values have been defined for the ***CO*** and ***CP*** entries.

Sample values shall be stored in the stream with the most significant bits first (big-endian order for samples larger than 8 bits). Samples that are not a multiple of 8 bits shall be packed into consecutive bytes, starting at the most significant end. If a sample extends across a byte boundary, the most significant bits shall be placed in the first byte, followed by less significant bits in subsequent bytes. For dual-channel stereophonic sounds, the samples shall be stored in an interleaved format, with each sample value for the left channel (channel 1) preceding the corresponding sample for the right (channel 2).

To maximize the portability of PDF documents containing embedded sounds, conforming readers should support at least the following formats (assuming the platform has sufficient hardware and OS support to play sounds at all):

R	8000, 11,025, or 22,050 samples per second
C	1 or 2 channels
B	8 or 16 bits per channel
E	Raw, Signed, or muLaw encoding

If the encoding (**E**) is Raw or Signed, **R** shall be 11,025 or 22,050 samples per channel. If the encoding is muLaw, **R** shall be 8000 samples per channel, **C** shall be 1 channel, and **B** shall be 8 bits per channel. Sound players shall convert between formats, downsample rates, and combine channels as necessary to render sound on the target platform.

13.4 Movies

The features described in this sub-clause are obsolescent and their use is no longer recommended. They are superseded by the general multimedia framework described in 13.2, "Multimedia."

PDF shall embed *movies* within a document by means of movie annotations (see 12.5.6.17, "Movie Annotations"). Despite the name, a movie may consist entirely of sound with no visible images to be displayed on the screen. The **Movie** and **A** (activation) entries in the movie annotation dictionary shall refer, respectively, to a *movie dictionary* (Table 295) that shall describe the static characteristics of the movie and a *movie activation dictionary* (Table 296) that shall specify how it shall be presented.

Table 295 – Entries in a movie dictionary

Key	Type	Value
F	file specification	*(Required)* A file specification identifying a self-describing movie file. NOTE The format of a self-describing movie file shall be left unspecified, and there is no guarantee of portability.
Aspect	array	*(Optional)* The width and height of the movie's bounding box, in pixels, and shall be specified as *[width height]*. This entry should be omitted for a movie consisting entirely of sound with no visible images.
Rotate	integer	*(Optional)* The number of degrees by which the movie shall be rotated clockwise relative to the page. The value shall be a multiple of 90. Default value: 0.
Poster	boolean or stream	*(Optional)* A flag or stream specifying whether and how a *poster image* representing the movie shall be displayed. If this value is a stream, it shall contain an image XObject (see 8.9, "Images") to be displayed as the poster. If it is the boolean value **true**, the poster image shall be retrieved from the movie file; if it is **false**, no poster shall be displayed. Default value: **false**.

Table 296 – Entries in a movie activation dictionary

Key	Type	Value
Start	(various)	*(Optional)* The starting time of the movie segment to be played. Movie time values shall be expressed in units of time based on a *time scale*, which defines the number of units per second. The default time scale shall be defined in the movie data. The starting time shall be nominally a non-negative 64-bit integer, specified as follows: • If it is representable as an integer (subject to the implementation limit for integers, as described in Annex C), it shall be specified as such. • If it is not representable as an integer, it shall be specified as an 8-byte string representing a 64-bit twos-complement integer, most significant byte first. • If it is expressed in a time scale different from that of the movie itself, it shall be represented as an array of two values: an integer or byte string denoting the starting time, followed by an integer specifying the time scale in units per second. If this entry is omitted, the movie shall be played from the beginning.
Duration	(various)	*(Optional)* The duration of the movie segment to be played, that shall be specified in the same form as **Start**. If this entry is omitted, the movie shall be played to the end.
Rate	number	*(Optional)* The initial speed at which to play the movie. If the value of this entry is negative, the movie shall be played backward with respect to **Start** and **Duration**. Default value: 1.0.
Volume	number	*(Optional)* The initial sound volume at which to play the movie, in the range −1.0 to 1.0. Higher values shall denote greater volume; negative values shall mute the sound. Default value: 1.0.
ShowControls	boolean	*(Optional)* A flag specifying whether to display a movie controller bar while playing the movie. Default value: **false**.

Table 296 – Entries in a movie activation dictionary (continued)

Key	Type	Value
Mode	name	*(Optional)* The *play mode* for playing the movie: Once Play once and stop. Open Play and leave the movie controller bar open. Repeat Play repeatedly from beginning to end until stopped. Palindrome Play continuously forward and backward until stopped. Default value: Once.
Synchronous	boolean	*(Optional)* A flag specifying whether to play the movie synchronously or asynchronously. If this value is **true**, the movie player shall retain control until the movie is completed or dismissed by the user. If the value is **false**, the player shall return control to the conforming reader immediately after starting the movie. Default value: **false**.
FWScale	array	*(Optional)* The magnification (zoom) factor at which the movie shall be played. The presence of this entry implies that the movie shall be played in a floating window. If the entry is absent, the movie shall be played in the annotation rectangle. The value of the entry shall be an array of two positive integers, *[numerator denominator]*, denoting a rational magnification factor for the movie. The final window size, in pixels, shall be (*numerator* ÷ *denominator*) × **Aspect** where the value of **Aspect** shall be taken from the movie dictionary (see Table 295).
FWPosition	array	*(Optional)* For floating play windows, the relative position of the window on the screen. The value shall be an array of two numbers *[horiz vert]* each in the range 0.0 to 1.0, denoting the relative horizontal and vertical position of the movie window with respect to the screen. EXAMPLE The value [0.5 0.5] centers the window on the screen. Default value: [0.5 0.5].

13.5 Alternate Presentations

Beginning with PDF 1.4, a PDF document shall contain *alternate presentations*, which specify alternate ways in which the document may be viewed. The optional **AlternatePresentations** entry *(PDF 1.4)* in a document's name dictionary (see Table 31) contains a name tree that maps name strings to the alternate presentations available for the document.

NOTE 1 Since conforming readers are not required to support alternate presentations, authors of documents containing alternate presentations should define the files such that something useful and meaningful can be displayed and printed. For example, if the document contains an alternate presentation slideshow of a sequence of photographs, the photographs should be viewable in a static form by viewers that are not capable of playing the slideshow.

As of PDF 1.5, the only type of alternate presentation is a *slideshow*. Slideshows may be invoked by means of JavaScript actions (see 12.6.4.16, "JavaScript Actions") initiated by user action on an interactive form element (see 12.7, "Interactive Forms").

The following table shows the entries in a slideshow dictionary.

Table 297 – Entries in a slideshow dictionary

Key	Type	Value
Type	name	*(Required; PDF 1.4)* The type of PDF object that this dictionary describes; shall be **SlideShow** for a slideshow dictionary.
Subtype	name	*(Required; PDF 1.4)* The subtype of the PDF object that this dictionary describes; shall be **Embedded** for a slideshow dictionary.
Resources	name tree	*(Required; PDF 1.4)* A name tree that maps name strings to objects referenced by the alternate presentation. NOTE Even though PDF treats the strings in the name tree as strings without a specified encoding, the slideshow shall interpret them as UTF-8 encoded Unicode.
StartResource	byte string	*(Required; PDF 1.4)* A byte string that shall match one of the strings in the **Resources** entry. It shall define the root object for the slideshow presentation.

NOTE 2 The **Resources** name tree represents a virtual file system to the slideshow. It associates strings (“file names”) with PDF objects that represent resources used by the slideshow. For example, a root stream may reference a file name, which would be looked up in the **Resources** name tree, and the corresponding object would be loaded as the file. (This virtual file system is flat; that is, there is no way to reference subfolders.)

NOTE 3 Typically, images are stored in the document as image XObjects (see 8.9.5, “Image Dictionaries”), thereby allowing them to be shared between the standard PDF representation and the slideshow. Other media objects are stored or embedded file streams (see 7.11.4, “Embedded File Streams”).

To allow conforming readers to verify content against their own supported features all referenced objects shall include a **Type** entry in their dictionary, even when the **Type** entry is normally optional for a given object.

EXAMPLE The following example illustrates the use of alternate presentation slideshows.

```
1 0 obj
    <</Type /Catalog
        /Pages 2 0 R
        /Names 3 0 R            % Indirect reference to name dictionary
    >>
...
3 0 obj                         % The name dictionary
    <</AlternatePresentations 4 0 R >>
endobj
4 0 obj                         % The alternate presentations name tree
    <</Names [(MySlideShow) 5 0 R]>>
endobj
5 0 obj                         % The slideshow definition
    <</Type /SlideShow
        /Subtype /Embedded
        /Resources <</Names [ (mysvg.svg) 31 0R
            (abc0001.jpg) 35 0 R (abc0002.jpg) 36 0 R
            (mysvg.js) 61 0 R (mymusic.mp3) 65 0 R ]>>
        /StartResource (mysvg.svg)
    >>
...
31 0 obj
    <</Type /Filespec           % The root object, which
        /F (mysvg.svg)          % points to an embedded file stream
        /EF <</F 32 0 R>>
    >>
endobj
32 0 obj                          % The embedded file stream
    <</Type /EmbeddedFile
        /Subtype /image#2Fsvg+xml
        /Length 72
    >>

    stream
        <?xml version="1.0" standalone="no"?>
        <svg><!-- Some SVG goes here--></svg>
    endstream
endobj

% ... other objects not shown
```

13.6 3D Artwork

13.6.1 General

Starting with PDF 1.6, collections of three-dimensional objects, such as those used by CAD software, may be embedded in PDF files. Such collections are often called *3D models*; in the context of PDF, they shall be referred to as *3D artwork*. The PDF constructs for 3D artwork support the following features:

- 3D artwork may be rendered within a page; that is, not as a separate window or user interface element.
- Multiple instances of 3D artwork may appear within a page or document.
- Specific views of 3D artwork may be specified, including a default view that shall be displayed initially and other views that may be selected. Views may have names that can be presented in a user interface.
- (*PDF 1.7*) Conforming readers may specify how 3D artwork shall be rendered, coloured, lit, and cross-sectioned, without the use of embedded JavaScript. They may also specify state information that shall be applied to individual nodes (3D graphic objects or collections thereof) in the 3D artwork, such as visibility, opacity, position, or orientation.
- Pages containing 3D artwork may be printed.
- Users may rotate and move the artwork, enabling them to examine complex objects from any angle or orientation.
- (*PDF 1.7*) Keyframe animations contained in 3D artwork may be played in specific styles and timescales, without programatic intervention.
- JavaScripts and other software may programmatically manipulate objects in the artwork, creating dynamic presentations in which objects move, spin, appear, and disappear.
- (*PDF 1.7*) The activation of 3D artwork can trigger the display of additional user interface items in the conforming reader. Such items may include model trees and toolbars.
- Two-dimensional (2D) content such as labels may be overlaid on 3D artwork. This feature is not the same as the ability to apply 2D markup annotations.
- *(PDF 1.7)* 2D markup annotations may be applied to specific views of the 3D artwork, using the **ExData** entry to identify the 3D annotation and the 3D view in that annotation.

The following sub-clauses describe the major PDF objects that relate to 3D artwork, as well as providing background information on 3D graphics:

- *3D annotations* provide a *virtual camera* through which the artwork shall be viewed. (see 13.6.2, “3D Annotations”).

- *3D streams* shall contain the actual specification of a piece of 3D artwork (see 13.6.3, “3D Streams””). This specification supports the Standard ECMA-363, *Universal 3D file format* developed by the 3D Industry Forum (see Bibliography).
- *3D views* shall specify information about the relationship between the camera and the 3D artwork (see 13.6.4, “3D Views”). Beginning with PDF 1.7, views may also describe additional parameters such as render mode, lighting, cross sections, and nodes. Nodes shall be 3D graphic objects or collections thereof.
- 3D coordinate systems are described in 13.6.5, “Coordinate Systems for 3D.”
- 2D markup annotations applied to 3D artwork views are described in 13.6.6, “3D Markup.”

NOTE Many of the concepts and terminology of 3D rendering are beyond the scope of this reference. Readers interested in further information are encouraged to consult outside references.

13.6.2 3D Annotations

3D annotations (*PDF 1.6*) are the means by which 3D artwork shall be represented in a PDF document. Table 298 shows the entries specific to a 3D annotation dictionary. Table 164 describes the entries common to all annotation dictionaries.

In addition to these entries, a 3D annotation shall provide an appearance stream in its **AP** entry (see Table 164) that has a normal appearance (the **N** entry in Table 168). This appearance may be used by applications that do not support 3D annotations and by all applications for the initial display of the annotation.

Table 298 – Additional entries specific to a 3D annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **3D** for a 3D annotation.
3DD	stream or dictionary	*(Required)* A 3D stream (see 13.6.3, “3D Streams”) or 3D reference dictionary (see 13.6.3.3, “3D Reference Dictionaries”) that specifies the 3D artwork to be shown.
3DV	(various)	*(Optional)* An object that specifies the default initial view of the 3D artwork that shall be used when the annotation is activated. It may be either a 3D view dictionary (see 13.6.4, “3D Views”) or one of the following types specifying an element in the **VA** array in the 3D stream (see Table 300): • An integer specifying an index into the **VA** array. • A text string matching the **IN** entry in one of the views in the **VA** array. • A name that indicates the first (**F**), last (**L**), or default (**D**) entries in the **VA** array. Default value: the default view in the 3D stream object specified by **3DD**.
3DA	dictionary	*(Optional)* An *activation dictionary* (see Table 299) that defines the times at which the annotation shall be activated and deactivated and the state of the 3D artwork instance at those times. Default value: an activation dictionary containing default values for all its entries.
3DI	boolean	*(Optional)* A flag indicating the primary use of the 3D annotation. If **true**, it is intended to be interactive; if **false**, it is intended to be manipulated programmatically, as with a JavaScript animation. Conforming readers may present different user interface controls for interactive 3D annotations (for example, to rotate, pan, or zoom the artwork) than for those managed by a script or other mechanism. Default value: **true**.

Table 298 – Additional entries specific to a 3D annotation (continued)

Key	Type	Value
3DB	rectangle	*(Optional)* The 3D view box, which is the rectangular area in which the 3D artwork shall be drawn. It shall be within the rectangle specified by the annotation's **Rect** entry and shall be expressed in the annotation's target coordinate system (see discussion following this Table). Default value: the annotation's **Rect** entry, expressed in the target coordinate system. This value is [-*w*/2 -*h*/2 *w*/2 *h*/2], where *w* and *h* are the width and height, respectively, of **Rect**.

The **3DB** entry specifies the *3D view box*, a rectangle in which the 3D artwork appears. The view box shall fit within the annotation's rectangle (specified by its **Rect** entry). It may be the same size, or it may be smaller if necessary to provide extra drawing area for additional 2D graphics within the annotation.

NOTE 1 Although 3D artwork can internally specify viewport size, conforming readers ignore it in favour of information provided by the ***3DB*** entry.

The view box shall be specified in the annotation's *target coordinate system*, whose origin is at the center of the annotation's rectangle. Units in this coordinate system are the same as default user space units. Therefore, the coordinates of the annotation's rectangle in the target coordinate system are

[-*w*/2 -*h*/2 *w*/2 *h*/2]

given *w* and *h* as the rectangle's width and height.

The **3DD** entry shall specify a 3D stream that contains the 3D artwork to be shown in the annotation; 3D streams are described in Section 13.6.3. The **3DD** entry may specify a 3D stream directly; it may also specify a 3D stream indirectly by means of a 3D reference dictionary (see 13.6.3.3, "3D Reference Dictionaries"). These options control whether annotations shall share the same run-time instance of the artwork.

The **3DV** entry shall specify the view of the 3D artwork that is displayed when the annotation is activated (as described in the next paragraph). 3D views, which are described in Section 13.6.4, represent settings for the virtual camera, such as position, orientation, and projection style. The view specified by **3DV** shall be one of the 3D view dictionaries listed in the **VA** entry in a 3D stream (see Table 300).

The **3DA** entry shall be an activation dictionary (see Table 299) that determines how the *state* of the annotation and its associated artwork may change.

NOTE 2 These states serve to delay the processing or display of 3D artwork until a user chooses to interact with it. Such delays in activating 3D artwork can be advantageous to performance.

At any given moment, a 3D annotation shall be in one of two states:

- *Inactive* (the default initial state): the annotation displays the annotation's normal appearance.

NOTE 3 It is typical, though not required, for the normal appearance to be a pre-rendered bitmap of the default view of the 3D artwork. Conforming writers should provide bitmaps of appropriate resolution for all intended uses of the document; for example, a high-resolution bitmap for high-quality printing and a screen-resolution bitmap for on-screen viewing. Optional content (see 8.11, "Optional Content") may be used to select the appropriate bitmap for each situation.

- *Active*: the annotation displays a rendering of the 3D artwork. This rendering shall be specified by the annotation's **3DV** entry.

Table 299 – Entries in a 3D activation dictionary

Key	Type	Value
A	name	*(Optional)* A name specifying the circumstances under which the annotation shall be activated. Valid values are: **PO** The annotation shall be activated as soon as the page containing the annotation is opened. **PV** The annotation shall be activated as soon as any part of the page containing the annotation becomes visible. **XA** The annotation shall remain inactive until explicitly activated by a script or user action. NOTE 1 At any one time, only a single page shall be considered open in a conforming reader, even though more than one page may be visible, depending on the page layout. Default value: **XA**. NOTE 2 For performance reasons, documents intended for viewing in a web browser should use explicit activation (***XA***). In non-interactive applications, such as printing systems or aggregating conforming reader, ***PO*** and ***PV*** indicate that the annotation shall be activated when the page is printed or placed; ***XA*** indicates that the annotation shall never be activated and the normal appearance shall be used.
AIS	name	*(Optional)* A name specifying the state of the artwork instance upon activation of the annotation. Valid values are: **I** The artwork shall be instantiated, but real-time script-driven animations shall be disabled. **L** Real-time script-driven animations shall be enabled if present; if not, the artwork shall be instantiated. Default value: **L**. NOTE 3 In non-interactive conforming readers, the artwork shall be instantiated and scripts shall be disabled.
D	name	*(Optional)* A name specifying the circumstances under which the annotation shall be deactivated. Valid values are: **PC** The annotation shall be deactivated as soon as the page is closed. **PI** The annotation shall be deactivated as soon as the page containing the annotation becomes invisible. **XD** The annotation shall remain active until explicitly deactivated by a script or user action. NOTE 4 At any one time, only a single page shall be considered open in the conforming reader, even though more than one page may be visible, depending on the page layout. Default value: **PI**.
DIS	name	*(Optional)* A name specifying the state of the artwork instance upon deactivation of the annotation. Valid values are **U** (uninstantiated), **I** (instantiated), and **L** (live). Default value: **U**. NOTE 5 If the value of this entry is ***L***, uninstantiation of instantiated artwork is necessary unless it has been modified. Uninstantiation is never required in non-interactive conforming readers.

Table 299 – Entries in a 3D activation dictionary (continued)

Key	Type	Value
TB	boolean	(*Optional; PDF 1.7*) A flag indicating the default behavior of an interactive toolbar associated with this annotation. If **true**, a toolbar shall be displayed by default when the annotation is activated and given focus. If **false**, a toolbar shall not be displayed by default. NOTE 6 Typically, a toolbar is positioned in proximity to the 3D annotation. Default value: **true**.
NP	boolean	(*Optional; PDF 1.7*) A flag indicating the default behavior of the user interface for viewing or managing information about the 3D artwork. Such user interfaces can enable navigation to different views or can depict the hierarchy of the objects in the artwork (the model tree). If **true**, the user interface should be made visible when the annotation is activated. If **false**, the user interface should not be made visible by default. Default value: **false**

The **A** and **D** entries of the activation dictionary determine when a 3D annotation may become active and inactive. The **AIS** and **DIS** entries determine what state the associated artwork shall be in when the annotation is activated or deactivated. 3D artwork may be in one of three states:

- *Uninstantiated:* the initial state of the artwork before it has been used in any way.
- *Instantiated*: the state in which the artwork has been read and a run-time instance of the artwork has been created. In this state, it may be rendered but script-driven real-time modifications (that is, animations) shall be disabled.
- *Live*: the artwork has been instantiated, and it is being modified in real time to achieve some animation effect. In the case of keyframe animation, the artwork shall be live while it is playing and then shall revert to an instantiated state when playing completes or is stopped.

NOTE 4 The live state is valid only for keyframe animations or in interactive conforming readers that have JavaScript support.

If 3D artwork becomes uninstantiated after having been instantiated, later use of the artwork requires re-instantiation (animations are lost, and the artwork appears in its initial form).

NOTE 5 For this reason, uninstantiation is not necessary unless the artwork has been modified in some way; consumers may choose to keep unchanged artwork instantiated for performance reasons.

NOTE 6 In non-interactive systems such as printing systems, the artwork cannot be changed. Therefore, applications may choose to deactivate annotations and uninstantiate artwork differently, based on factors such as memory usage and the time needed to instantiate artwork, and the ***TB***, ***NP***, ***D*** and ***DIS*** entries may be ignored.

Multiple 3D annotations may share an instance of 3D artwork, as described in 13.6.3.3, "3D Reference Dictionaries". In such a case, the state of the artwork instance shall be determined in the following way:

- If any active annotation dictates (through its activation dictionary) that the artwork shall be live, it shall be live.
- Otherwise, if any active annotation dictates that the artwork shall be instantiated, it shall be instantiated.
- Otherwise (that is, all active annotations dictate that the artwork shall be uninstantiated), the artwork shall be uninstantiated.

The rules described in 13.6.2, “3D Annotations”, apply only to active annotations. If all annotations referring to the artwork are inactive, the artwork nevertheless may be uninstantiated, instantiated, or live 3D Streams.

13.6.3 3D Streams

13.6.3.1 General

The specification of 3D artwork shall be contained in a *3D stream*. 3D stream dictionaries, whose entries (in addition to the regular stream dictionary's entries; see 7.3.7, “Dictionary Objects”) are shown in Table 300, may provide a set of predefined views of the artwork, as well as a default view. They may also provide scripts and resources for providing customized behaviours or presentations.

Table 300 – Entries in a 3D stream dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3D** for a 3D stream.
Subtype	name	*(Required)* A name specifying the format of the 3D data contained in the stream. The only valid value is **U3D**.
VA	array	*(Optional)* An array of *3D view dictionaries*, each of which specifies a named preset view of this 3D artwork (see Section 13.6.4, “3D Views”).
DV	(various)	*(Optional)* An object that specifies the default (initial) view of the 3D artwork. It may be a 3D view dictionary (see Section 13.6.4, “3D Views”) or one of the following types: • An integer specifying an index into the **VA** array. • A text string matching the **IN** entry in one of the views in the **VA** array. • A name that indicates the first (**F**) or last (**L**) entries in the **VA** array. Default value: 0 (the first entry in the **VA** array) if **VA** is present; if **VA** is not present, the default view shall be specified within the 3D stream itself.
Resources	name tree	*(Optional)* A name tree that maps name strings to objects that may be used by applications or scripts to modify the default view of the 3D artwork. The names in this name tree shall be text strings so as to be encoded in a way that will be accessible from JavaScript.
OnInstantiate	stream	*(Optional)* A JavaScript script that shall be executed when the 3D stream is instantiated.
AN	dictionary	*(Optional; PDF 1.7)* An *animation style dictionary* indicating the method that conforming readers should use to drive keyframe animations present in this artwork (see 13.6.3.2, "3D Animation Style Dictionaries"). Default value: an animation style dictionary whose **Subtype** entry has a value of **None**.

The **Subtype** entry specifies the format of the 3D stream data. The only valid value is **U3D**, which indicates that the stream data conforms to the *Universal 3D File Format* specification (see Bibliography). Conforming readers shall be prepared to encounter unknown values for **Subtype** and recover appropriately, which usually means leaving the annotation in its inactive state, displaying its normal appearance.

NOTE Conforming readers should follow the approach of falling back to the normal appearance with regard to entries in other dictionaries that may take different types or values than the ones specified here.

If present, the **VA** entry shall be an array containing a list of named present views of the 3D artwork. Each entry in the array shall be a 3D view dictionary (see 13.6.4, “3D Views”) that shall contain the name of the view and the information needed to display the view. The order of array elements determines the order in which the views shall be presented in a user interface. The **DV** entry specifies the view that shall be used as the initial view of the 3D artwork.

Default views shall be determined in the following order of precedence: in the annotation dictionary, in the 3D stream dictionary, or in the 3D artwork contained in the 3D stream.

3D streams contain information that may be used by conforming readers and by scripts to perform animations and other programmatically-defined behaviours, such as changing the viewing orientation or moving individual components of the artwork. If present, the **OnInstantiate** entry shall contain a JavaScript script that shall be executed by applications that support JavaScript whenever a 3D stream is read to create an instance of the 3D artwork. The **Resources** entry shall be a name tree that contains objects that may be used to modify the initial appearance of the 3D artwork.

13.6.3.2 3D Animation Style Dictionaries

A *3D animation style dictionary (PDF 1.7)* specifies the method that conforming readers should use to apply timeline scaling to keyframe animations. It may also specify that keyframe animations be played repeatedly. The **AN** entry of the 3D stream shall specify a 3D animation style dictionary.

A keyframe animation may be provided as the content of a 3D stream dictionary. A keyframe animation provides key frames and specifies the mapping for the position of geometry over a set period of time (*animation timeline*). Keyframe animation is an interactive feature that is highly dependent on the behaviour and controls provided by the conforming reader.

Table 301 shows the entries in an animation style dictionary.

Table 301 – Entries in an 3D animation style dictionary

Key	Type	Value
Type	name	(*Optional*). The type of PDF object that this dictionary describes; if present, shall be **3DAnimationStyle**.
Subtype	name	(*Optional*) The animation style described by this dictionary; see Table 302 for valid values. If an animation style is encountered other than those described in Table 302, an animation style of **None** shall be used. Default value: **None**
PC	integer	(*Optional*) An integer specifying the play count for this animation style. A non-negative integer represents the number of times the animation shall be played. A negative integer indicates that the animation shall be infinitely repeated. This value shall be ignored for animation styles of type **None**. Default value: 0
TM	number	(*Optional*) A positive number specifying the time multiplier to be used when running the animation. A value greater than one shortens the time it takes to play the animation, or effectively speeds up the animation. NOTE This allows authors to adjust the desired speed of animations, without having to re-author the 3D artwork. This value shall be ignored for animation styles of type **None**. Default value: 1

The descriptions of the animation styles (see Table 302) use the following variables to represent application time or keyframe settings specified in the 3D artwork.

- *t* is a point on the animation time line. This value shall be used in conjunction with the keyframe animation data to determine the state of the 3D artwork.
- $[r_0, r_1]$ is the keyframe animation time line.
- t_a is the current time of the conforming reader.
- t_0 is the time when the conforming reader starts the animation.
- *p* is the time it takes to play the keyframe animation through one cycle. In the case of the **Linear** animation style, one cycle consists of playing the animation through once from beginning to end. In the case of the **Oscillating** animation style, one cycle consists of playing the animation from beginning to end and then from end to beginning.
- *m* is the positive multiplier specified by the **TM** entry in the animation style dictionary.

Table 302 – Animation styles

None	Keyframe animations shall not be driven directly by the conforming reader. This value shall be used by documents that are intended to drive animations through an alternate means, such as JavaScript. The remaining entries in the animation style dictionary shall be ignored.
Linear	Keyframe animations shall be driven linearly from beginning to end. This animation style results in a repetitive playthrough of the animation, such as in a walking motion. $t = (m(t_a - t_0) + r_0) \% (r_1 - r_0)$ $p = (r_1 - r_0) / m$ The “%” symbol indicates the modulus operator.
Oscillating	Keyframe animations shall oscillate along their time range. This animation style results in a back-and-forth playing of the animation, such as exploding or collapsing parts. $t = (0.5)(r_1 - r_0)(1 - \cos(m(t_a - t_0))) + r_0$ $p = 2 * \text{pi} / m$

13.6.3.3 3D Reference Dictionaries

More than one 3D annotation may be associated with the same 3D artwork. There are two ways in which this association may occur, as determined by the annotation’s **3DD** entry (see Table 298):

- If the **3DD** entry specifies a 3D stream, the annotation shall have its own run-time instance of the 3D artwork. Any changes to the artwork shall be reflected only in this annotation. Other annotations that refer to the same stream shall have separate run-time instances.
- If the **3DD** entry specifies a 3D reference dictionary (whose entries are shown in Table 303), the annotation shall have a run-time instance of the 3D artwork with all other annotations that specify the same reference dictionary. Any changes to the artwork shall be reflected in all such annotations.

Table 303 – Entries in a 3D reference dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DRef** for a 3D reference dictionary.
3DD	stream	*(Required)* The 3D stream (see 13.6.3, "3D Streams") containing the specification of the 3D artwork.

EXAMPLE The following example and Figure 66 through Figure 68 show three annotations that use the same 3D artwork. Object 100 (Annotation 1) has its own run-time instance of the 3D stream (object 200); object 101(Annotation 2) and object 102 (Annotation 3) share a run-time instance through the 3D reference dictionary (object 201).

```
100 0 obj               % 3D annotation 1
  << /Type /Annot
     /Subtype /3D
     /3DD 200 0 R       % Reference to the 3D stream containing the 3D artwork
  >>
endobj

101 0 obj               % 3D annotation 2
  << /Type /Annot
     /Subtype /3D
     /3DD 201 0 R       % Reference to a 3D reference dictionary
  >>
endobj

102 0 obj               % 3D annotation 3
  << /Type /Annot
     /Subtype /3D
     /3DD 201 0 R       % Reference to the same 3D reference dictionary
  >>
endobj

200 0 obj               % The 3D stream
  << /Type /3D
     /Subtype /U3D
     ... other keys related to a stream, such as /Length
  >>
  stream
     ... U3D data...
  endstream
endobj

201 0 obj               % 3D reference dictionary
  << /Type /3DRef
     /3DD 200 0 R       % Reference to the actual 3D artwork.
  >>
endobj
```

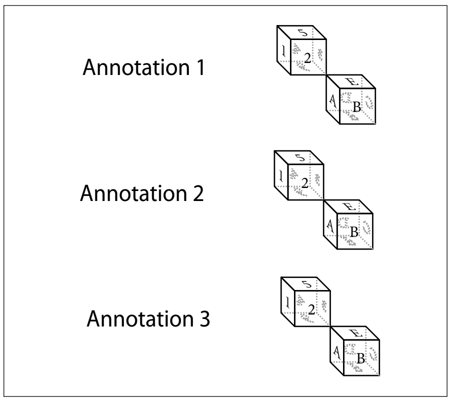


Figure 66 – Default view of artwork

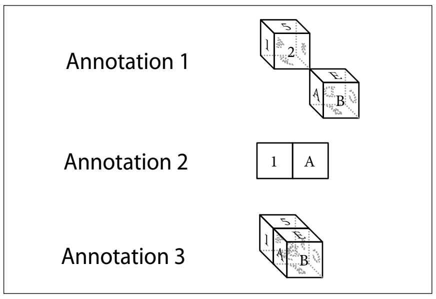


Figure 67 – Annotation 2 rotated

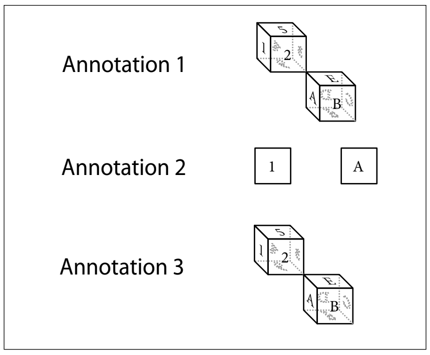


Figure 68 – Shared artwork (annotations 2 & 3) modified

The figures show how the objects in the Example in 13.5, “Alternate Presentations,” might be used. Figure 66 shows the same initial view of the artwork in all three annotations. Figure 67 shows the results of rotating the view of the artwork within Annotation 2. Figure 68 shows the results of manipulating the artwork shared by Annotation 2 and Annotation 3: they both reflect the change in the artwork because they share the same run-time instance. Annotation 1 remains unchanged because it has its own run-time instance.

NOTE When multiple annotations refer to the same instance of 3D artwork, the state of the instance is determined as described in 13.6.2, “3D Annotations.”

13.6.4 3D Views

13.6.4.1 General

A *3D view* (or simply *view*) specifies parameters that shall be applied to the virtual camera associated with a 3D annotation. These parameters may include orientation and position of the camera, details regarding the projection of camera coordinates into the annotation's target coordinate system, and a description of the background on which the artwork shall be drawn. Starting with PDF 1.7, views may specify how 3D artwork is rendered, coloured, lit, and cross-sectioned, without the use of embedded JavaScript. Views may also specify which nodes (three-dimensional areas) of 3D artwork shall be included in a view and whether those nodes are opaque or invisible.

NOTE 1 Users can manipulate views by performing interactive operations such as free rotation and translation. In addition, 3D artwork can contain a set of predefined views that the author deems to be of particular interest. For example, a mechanical drawing of a part may have specific views showing the top, bottom, left, right, front, and back of an object.

A 3D stream may contain a list of named preset views of the 3D artwork, as specified by the **VA** entry, which shall be an array of 3D view dictionaries. The entries in a 3D view dictionary are shown in Table 304.

Table 304 – Entries in a 3D view dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DView** for a 3D view dictionary.
XN	text string	*(Required)* The external name of the view, suitable for presentation in a user interface.
IN	text string	*(Optional)* The internal name of the view, used to refer to the view from other objects, such as the go-to-3D-view action (see 12.6.4.15, "Go-To-3D-View Actions").
MS	name	*(Optional)* A name specifying how the 3D camera-to-world transformation matrix shall be determined. The following values are valid: **M** Indicates that the **C2W** entry shall specify the matrix **U3D** Indicates that the view node selected by the **U3DPath** entry shall specify the matrix. If omitted, the view specified in the 3D artwork shall be used.
C2W	array	*(Required if the value of **MS** is **M**, ignored otherwise)* A 12-element 3D transformation matrix that specifies a position and orientation of the camera in world coordinates.
U3DPath	text string or array	*(Required if the value of **MS** is **U3D**, ignored otherwise)* A sequence of one or more text strings used to access a *view node* within the 3D artwork. The first string in the array is a *node ID* for the *root view node*, and each subsequent string is the node ID for a child of the view node specified by the prior string. Each view node specifies a 3D transformation matrix (see 13.6.5, "Coordinate Systems for 3D"); the concatenation of all the matrices forms the camera-to-world matrix. Conforming writers should specify only a single text string, not an array, for this entry. NOTE Do not confuse View Nodes with nodes. A View Node is a parameter in the 3D artwork that specifies a view, while a node is a PDF dictionary that specifies 3D graphic objects or collections thereof.

Table 304 – Entries in a 3D view dictionary (continued)

Key	Type	Value
CO	number	*(Optional; used only if **MS** is present)* A non-negative number indicating a distance in the camera coordinate system along the *z* axis to the center of orbit for this view; see discussion following this Table. If this entry is not present, the conforming reader shall determine the center of orbit.
P	dictionary	*(Optional)* A *projection dictionary* (see 13.6.4.2, “Projection Dictionaries”) that defines the projection of coordinates in the 3D artwork (already transformed into camera coordinates) onto the target coordinate system of the annotation. Default value: a projection dictionary where the value of **Subtype** is **Perspective**, the value of **FOV** is 90, and all other entries take their default values.
O	stream	*(Optional; meaningful only if **MS** and **P** are present)* A form XObject that shall be used to overlay 2D graphics on top of the rendered 3D artwork (see 13.6.6, “3D Markup”).
BG	dictionary	*(Optional)* A *background dictionary* that defines the background over which the 3D artwork shall be drawn (see 13.6.4.3, “3D Background Dictionaries”). Default value: a background dictionary whose entries take their default values.
RM	dictionary	(*Optional; PDF 1.7*) A *render mode dictionary* that specifies the render mode to use when rendering 3D artwork with this view (see 13.6.4.4, “3D Render Mode Dictionaries”). If omitted, the render mode specified in the 3D artwork shall be used.
LS	dictionary	(*Optional; PDF 1.7*) A *lighting scheme dictionary* that specifies the lighting scheme to be used when rendering 3D artwork with this view (see 13.6.4.5, “3D Lighting Scheme Dictionaries”). If omitted, the lighting scheme specified in the 3D artwork shall be used.
SA	array	(*Optional; PDF 1.7*) An array that contains *cross section dictionaries (*see 13.6.4.6, “3D Cross Section Dictionaries”). Each cross section dictionary provides parameters for applying a cross section to the 3D artwork when using this view. An empty array signifies that no cross sections shall be displayed.
NA	array	(*Optional; PDF 1.7*; meaningful only if **NR** is present) An array that contains 3D node dictionaries (see 13.6.4.7, “3D Node Dictionaries”). Each node dictionary may contain entries that change the node’s state, including its opacity and its position in world space. This entry and the **NR** entry specify how the state of each node shall be changed. If a node dictionary is present more than once, only the last such dictionary (using a depth-first traversal) shall be used.
NR	boolean	(*Optional; PDF 1.7*) Specifies whether nodes specified in the **NA** array shall be returned to their original states (as specified in the 3D artwork) before applying transformation matrices and opacity settings specified in the node dictionaries. If **true**, the artwork’s 3D node parameters shall be restored to their original states and then the dictionaries specified by the **NA** array shall be applied. If **false**, the dictionaries specified by the **NA** array shall be applied to the current states of the nodes. In addition to the parameters specified by a 3D node dictionary, this flag should also apply to any runtime parameters used by a conforming reader. Default value: **false**

For any view, the conforming writer may provide 2D content specific to the view, to be drawn on top of the 3D artwork. The **O** entry specifies a form XObject that shall be overlaid on the rendered 3D artwork. The coordinate system of the form XObject shall be defined to be the same as the (*x*, *y*, 0) plane in the camera coordinate system (see 13.6.5, “Coordinate Systems for 3D”).

Use of the **O** entry is subject to the following restrictions.

NOTE 2 Failure to abide by them could result in misalignment of the overlay with the rendered 3D graphics:

- It may be specified only in 3D view dictionaries in which both a camera-to-world matrix (**MS** and associated entries) and a projection dictionary (the **P** entry) are present.
- The form XObject shall be associated with a specific view (not with the camera position defined by the 3D view dictionary). The conforming reader should draw it only when the user navigates using the 3D view, not when the user happens to navigate to the same orientation by manual means.
- The confirming reader should draw it only if the user has not invoked any actions that alter the artwork-to-world matrix.

The **CO** entry specifies the distance from the camera to the *center of orbit* for the 3D view, which is the point around which the camera shall rotate when performing an orbit-style navigation. Figure 69 illustrates camera positioning when orbiting around the center of orbit.

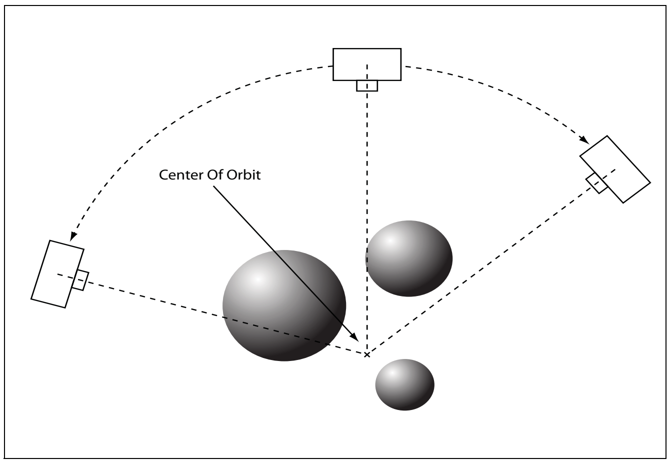


Figure 69 – Rotation around the center of orbit

NOTE 3 The **LS** entry allows the lighting of the 3D artwork to be changed without changing the artwork itself. This enables consumers to view a given piece of 3D artwork with a variety of lighting options without requiring multiple copies of the 3D artwork stream that differ only in lighting. It also enables artwork with poor lighting to be corrected in cases where the original content cannot be re-authored. See 13.6.4.5, “3D Lighting Scheme Dictionaries.”

The **SA** entry provides cross section information for clipping 3D artwork while its associated view is active. This allows view authors to be more clear in calling out the intended areas of interest for a particular view, some of which might otherwise be completely obscured. See 13.6.4.6, “3D Cross Section Dictionaries.”

The **NR** and **NA** entries are meant to give a more accurate representation of the 3D artwork at a given state. These keys give view authors finer granularity in manipulating the artwork to be presented in a particular way. They also provide a means for returning node parameters to a known state after potential changes by interactive features such as keyframe animations and JavaScript. See 13.6.4.7, “3D Node Dictionaries.”

13.6.4.2 Projection Dictionaries

A *projection dictionary* (see Table 305) defines the mapping of 3D camera coordinates onto the target coordinate system of the annotation. Each 3D view may specify a projection dictionary by means of its **P** entry.

NOTE Although view nodes can specify projection information, PDF consumers ignore it in favour of information in the projection dictionary.

PDF 1.6 introduces *near/far clipping*. This type of clipping defines a *near plane* and a *far plane* (as shown in Figure 70). Objects, or parts of objects, that are beyond the far plane or closer to the camera than the near plane are not drawn. 3D objects shall be projected onto the near plane and then scaled and positioned within the annotation's target coordinate system, as described Table 305.

Table 305 – Entries in a projection dictionary

Key	Type	Value
Subtype	name	*(Required)* The type of projection. Valid values shall be **O** (orthographic) or **P** (perspective).
CS	name	*(Optional)* The clipping style. Valid values shall be **XNF** (explicit near/far) or **ANF** (automatic near/far). Default value: **ANF**.
F	number	*(Optional; meaningful only if the value of **CS** is **XNF**)* The far clipping distance, expressed in the camera coordinate system. No parts of objects whose *z* coordinates are greater than the value of this entry are drawn. If this entry is absent, no far clipping occurs.
N	number	*(Meaningful only if the value of **CS** is **XNF**; required if the value of **Subtype** is **P**)* The near clipping distance, expressed in the camera coordinate system. No parts of objects whose *z* coordinates are less than the value of this entry are drawn. If **Subtype** is **P**, the value shall be positive; if **Subtype** is **O**, the value shall be non-negative, and the default value is 0.
FOV	number	*(Required if **Subtype** is **P**, ignored otherwise)* A number between 0 and 180, inclusive, specifying the field of view of the virtual camera, in degrees. It defines a cone in 3D space centered around the *z* axis and a circle where the cone intersects the near clipping plane. The circle, along with the value of **PS**, specify the scaling of the projected artwork when rendered in the 2D plane of the annotation.
PS	number or name	*(Optional; meaningful only if **Subtype** is **P**)* An object that specifies the scaling used when projecting the 3D artwork onto the annotation's target coordinate system. It defines the diameter of the circle formed by the intersection of the near plane and the cone specified by **FOV**. The value may be one of the following: • A positive number that explicitly specifies the diameter as a distance in the annotation's target coordinate system. • A name specifying that the diameter shall be set to the width (**W**), height (**H**), minimum of width and height (**Min**), or maximum of width and height (**Max**) of the annotation's 3D view box. Default value: **W**.
OS	number	*(Optional; meaningful only if **Subtype** is **O**)* A positive number that specifies the scale factor to be applied to both the *x* and *y* coordinates when projecting onto the annotation's target coordinate system (the *z* coordinate is discarded). Default value: 1.
OB	name	(*Optional; PDF 1.7; meaningful only if Subtype is **O***) A name that specifies a strategy for binding (scaling to fit) the near plane's *x* and *y* coordinates onto the annotation's target coordinate system. The scaling specified in this entry shall be applied in addition to the scaling factor specified by the **OS** entry. The value may be one of the following: **W** Scale to fit the width of the annotation **H** Scale to fit the height of the annotation **Min** Scale to fit the lesser of width or height of the annotation **Max** Scale to fit the greater of width or height of the annotation **Absolute** No scaling should occur due to binding. Default value: **Absolute**.

The **CS** entry defines how the near and far planes are determined. A value of **XNF** means that the **N** and **F** entries explicitly specify the *z* coordinate of the near and far planes, respectively. A value of **ANF** for **CS** means that the near and far planes shall be determined automatically based on the objects in the artwork.

The **Subtype** entry specifies the type of projection, which determines how objects are projected onto the near plane and scaled. The possible values are **O** for *orthographic projection* and **P** for *perspective projection*.

For orthographic projection, objects shall be projected onto the near plane by simply discarding their *z* value. They shall be scaled from units of the near plane's coordinate system to those of the annotation's target coordinate system by the combined factors specified by the **OS** entry and the OB entry.

For perspective projection, a given coordinate (x, y, z) shall be projected onto the near plane, defining a 2D coordinate (x_1, y_1) using the following formulas:

$$x_1 = x \times \frac{n}{z}$$

$$y_1 = y \times \frac{n}{z}$$

where *n* is the *z* coordinate of the near plane.

Scaling with perspective projection is more complicated than for orthographic projection. The **FOV** entry specifies an angle that defines a cone centered along the *z* axis in the camera coordinate system (see Figure 70). The cone intersects with the near plane, forming a circular area on the near plane. Figure 71 shows this circle and graphics from the position of the camera.

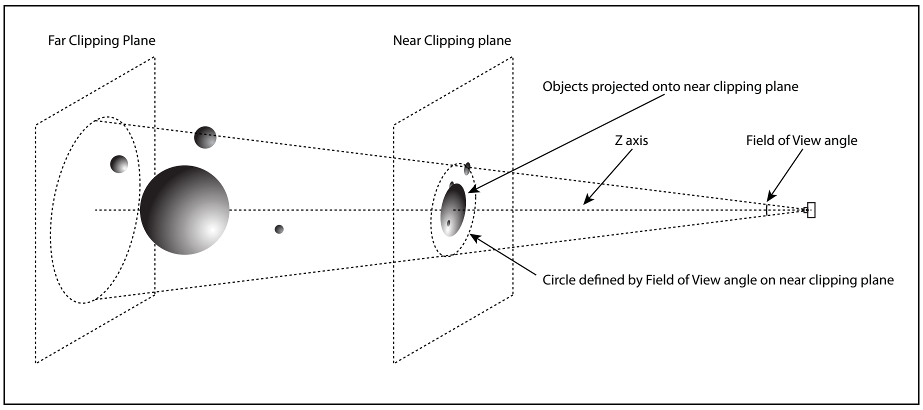

Far Clipping Plane
Near Clipping plane
Objects projected onto near clipping plane
Z axis
Field of View angle
Circle defined by Field of View angle on near clipping plane

Figure 70 – Perspective projection of 3D artwork onto the near plane

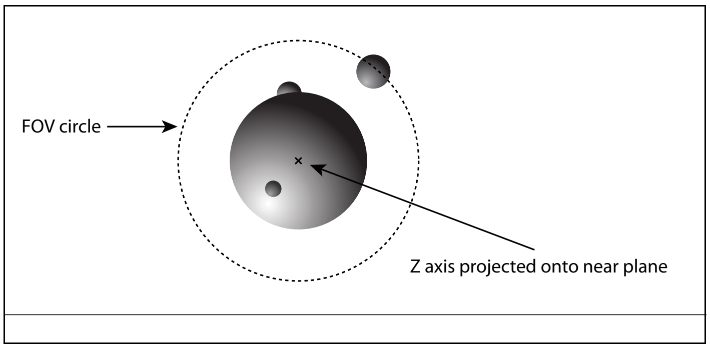


Figure 71 – Objects projected onto the near clipping plane, as seen from the position of the camera

The **PS** entry specifies the diameter that this circle will have when the graphics projected onto the near plane are rendered in the annotation's 3D view box (see Figure 72). Although the diameter of the circle determines the scaling factor, graphics outside the circle shall also be displayed, providing they fit within the view box, as seen in the figure.

Figure 73 shows the entire 3D annotation. In this case, the 3D view box is smaller than the annotation's rectangle, which also contains 2D content outside the 3D view box.

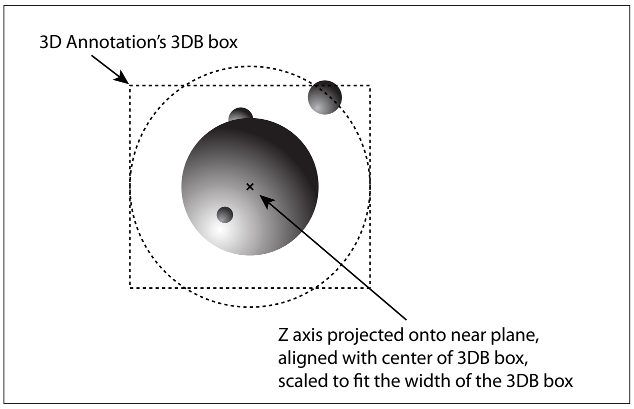

3D Annotation's 3DB box
Z axis projected onto near plane, aligned with center of 3DB box, scaled to fit the width of the 3DB box

Figure 72 – Positioning and scaling the near plane onto the annotation's 3D view box

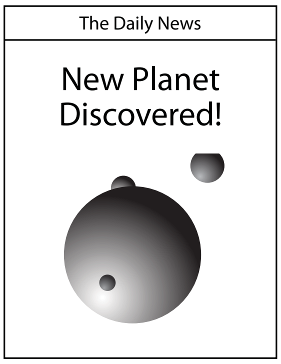


Figure 73 – 3D annotation positioned on the page

13.6.4.3 3D Background Dictionaries

A *3D background dictionary* defines the background over which a 3D view shall be drawn.Table 306 shows the entries in a background dictionary. Currently, only a single opaque colour is supported, where the colour shall be defined in the **DeviceRGB** colour space. 3D artwork may include transparent objects; however, there is no interaction between such objects and objects drawn below the annotation. In effect, the 3D artwork and its background form a transparency group whose flattened results have an opacity of 1 (see 11, "Transparency").

NOTE An annotation's normal appearance should have the same behaviour with respect to transparency when the appearance is intended to depict the 3D artwork. This does not apply when the appearance is used for another purpose, such as a compatibility warning message.

Table 306 – Entries in a 3D background dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be *3DBG* for a 3D background dictionary.
Subtype	name	*(Optional)* The type of background. The only valid value shall be *SC* (solid colour), which indicates a single opaque colour. Default value: *SC*.
CS	name or array	*(Optional)* The colour space of the background. The only valid value shall be the name **DeviceRGB**. Default value: **DeviceRGB**. PDF consumers shall be prepared to encounter other values that may be supported in future versions of PDF.
C	(various)	*(Optional)* The colour of the background, in the colour space defined by **CS**. Default value: an array [1 1 1] representing the colour white when the value of **CS** is **DeviceRGB**.
EA	boolean	*(Optional)* If **true**, the background shall apply to the entire annotation; if **false**, the background shall apply only to the rectangle specified by the annotation's 3D view box (the **3DB** entry in Table 298). Default value: **false**.

13.6.4.4 3D Render Mode Dictionaries

A *3D render mode dictionary (PDF 1.7)* specifies the style in which the 3D artwork shall be rendered.

NOTE 1 Surfaces may be filled with opaque colours, they may be stroked as a "wireframe," or the artwork may be rendered with special lighting effects.

NOTE 2 A render mode dictionary enables document authors to customize the rendered appearance of 3D artwork to suit the needs of the intended consumer, without reauthoring the artwork. For conforming readers concerned strictly with geometry, complex artwork rendered using the **Wireframe** or **Points** style may have much better performance without the added overhead of texturing and lighting effects. Artwork in a document intended for print may have a much more integrated feel when using the **Illustration** render mode style.

The **RM** entry in the 3D views dictionary may specify a 3D render mode dictionary.

Table 307 shows the entries in a render mode dictionary.

Table 307 – Entries in a render mode dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DRenderMode**.
Subtype	name	*(Required)* The type of render mode described by this dictionary; see Table 308 for specific values. If an unrecognized value is encountered, then this render mode dictionary shall be ignored.

Table 307 – Entries in a render mode dictionary (continued)

Key	Type	Value
AC	array	*(Optional)* An array that specifies the *auxiliary colour* that shall be used when rendering the 3D image. The first entry in the array shall be a colour space; the subsequent entries shall be values specifying colour values in that colour space. The interpretation of this entry depends on the render mode specified by the **Subtype** entry, but it is often used to specify a colour for drawing points or edges. The only valid colour space shall be **DeviceRGB**. If a colour space other than **DeviceRGB** is specified, this entry shall be ignored and the default value shall be used. Default value: **[/DeviceRGB 0 0 0]** representing the colour black.
FC	name or array	*(Optional)* A name or array that specifies the *face color* to be used when rendering the 3D image. This entry shall be relevant only when **Subtype** has a value of **Illustration**. If the value of **FC** is an array, the first entry in the array shall be a colour space and the subsequent entries shall be values specifying values in that colour space. The only valid colour space is **DeviceRGB**. Any colour space other than **DeviceRGB** shall be ignored and the default value shall be used. If the value of **FC** is a name, it shall describe a colour. The only valid name value shall **BG**, specifying the current background colour in use for displaying the artwork. If a name other than **BG** is encountered, this entry shall be ignored and the background colour for the host annotation shall be used (see Table 189). Default value: **BG**
O	number	*(Optional)* A number specifying the opacity of the added transparency applied by some render modes, using a standard additive blend. Default value: 0.5
CV	number	*(Optional)* A number specifying the angle, in degrees, that shall be used as the *crease value* when determining silhouette edges. If two front-facing faces share an edge and the angle between the normals of those faces is greater than or equal to the crease value, then that shared edge shall be considered a silhouette edge. Default value: 45

For render modes that add a level of transparency to the rendering, the **O** entry specifies the additional opacity that shall be used. All such transparency effects use a standard additive blend mode.

The **CV** entry sets the crease value that shall be used when determining silhouette edges, which may be used to adjust the appearance of illustrated render modes. An edge shared by two faces shall be considered a silhouette edge if either of the following conditions is met:

- One face is front-facing and the other is back-facing.
- The angle between the two faces is greater than or equal to the crease value.

Table 308 describes the render modes that may be specified in a render mode dictionary.

Table 308 – Render modes

Mode	Description
Solid	Displays textured and lit geometric shapes. In the case of artwork that conforms to the *Universal 3D File Format* specification, these shapes are triangles. The **AC** entry shall be ignored.
SolidWireframe	Displays textured and lit geometric shapes (triangles) with single colour edges on top of them. The colour of these edges shall be determined by the **AC** entry.
Transparent	Displays textured and lit geometric shapes (triangles) with an added level of transparency. The AC entry shall be ignored.
TransparentWireframe	Displays textured and lit geometric shapes (triangles) with an added level of transparency, with single colour opaque edges on top of it. The colour of these edges shall be determined by the **AC** entry.
BoundingBox	Displays the bounding box edges of each node, aligned with the axes of the local coordinate space for that node. The colour of the bounding box edges shall be determined by the **AC** entry.
TransparentBoundingBox	Displays bounding boxes faces of each node, aligned with the axes of the local coordinate space for that node, with an added level of transparency. The colour of the bounding box faces shall be determined by the **FC** entry.
TransparentBoundingBoxOutline	Displays bounding boxes edges and faces of each node, aligned with the axes of the local coordinate space for that node, with an added level of transparency. The colour of the bounding box edges shall be determined by the **AC** entry. The colour of the bounding boxes faces shall be determined by the **FC** entry.
Wireframe	Displays only edges in a single colour. The colour of these edges shall be determined by the **AC** entry.
ShadedWireframe	Displays only edges, though interpolates their colour between their two vertices and applies lighting. The **AC** entry shall be ignored.
HiddenWireframe	Displays edges in a single colour, though removes back-facing and obscured edges. The colour of these edges shall be determined by the **AC** entry.
Vertices	Displays only vertices in a single colour. The colour of these points shall be determined by the **AC** entry.
ShadedVertices	Displays only vertices, though uses their vertex colour and applies lighting. The **AC** entry shall be ignored.
Illustration	Displays silhouette edges with surfaces, removes obscured lines. The colour of these edges shall be determined by the **AC** entry, and the colour of the surfaces shall be determined by the **FC** entry.
SolidOutline	Displays silhouette edges with lit and textured surfaces, removes obscured lines. The colour of these edges shall be determined by the **AC** entry.
ShadedIllustration	Displays silhouette edges with lit and textured surfaces and an additional emissive term to remove poorly lit areas of the artwork. The colour of these edges shall be determined by the **AC** entry.

If a render mode type is encountered other than those described in Table 308, the render mode dictionary containing that entry shall be ignored by its consumers. This allows future documents using new render modes to behave consistently with future documents using new 3D view constructs that are ignored by older viewers.

13.6.4.5 3D Lighting Scheme Dictionaries

A *3D lighting scheme dictionary (PDF 1.7)* specifies the lighting to apply to 3D artwork. The **LS** entry in the 3D view may include a 3D lighting scheme dictionary.

Table 301 shows the entries in a 3D lighting scheme dictionary.

Table 309 – Entries in a 3D lighting scheme dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DLightingScheme**.
Subtype	name	*(Required)* The style of lighting scheme described by this dictionary (see Table 310).

Table 310 describes the supported lighting schemes. With the exception of the **Artwork** lighting style, all the lights specified in Table 310 are *infinite* lights (also known as *distant* lights). Unlike lights from a point source, all rays from an infinite light source are emitted along a single direction vector. For lights specifying an *ambient* term, this term shall be added to the diffuse colour of an object's material. All colours shall be specified in the **DeviceRGB** colour space.

When a style other than **Artwork** is used, only those lights described shall be present; any lighting described in the artwork shall not be used.

Table 310 – 3D lighting scheme styles

Scheme	Description
Artwork	Lights as specified in the 3D artwork. This has the same effect as if the 3D lighting scheme dictionary were omitted.
None	No lights shall be used. That is, lighting specified in the 3D artwork shall be ignored.
White	Three blue-grey infinite lights, no ambient term Light 1 Colour: < 0.38, 0.38, 0.45 > Direction: < -2.0, -1.5, -0.5 > Light 2 Colour: < 0.6, 0.6, 0.67 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < 0.5, 0.5, 0.57 > Direction: < -0.5, 0.0, 2.0 >
Day	Three light-grey infinite lights, no ambient term Light 1 Colour: < 0.5, 0.5, 0.5 > Direction: < -2.0, -1.5, -.5 > Light 2 Colour: < 0.8, 0.8, 0.9 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < 0.9, 0.9, 0.9 > Direction: < 0.02, 0.01, 2.0 >
Night	One yellow, one aqua, and one blue infinite light, no ambient term Light 1 Colour: < 1, .75, .39 > Direction: < -2.0, -1.5, -0.5 > Light 2 Colour: < 0.31, 0.47, 0.55 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < .5, .5, 1.0 > Direction: < 0.0, 0.0, 2.0 >

Table 310 – 3D lighting scheme styles (continued)

Scheme	Description
Hard	Three grey infinite lights, moderate ambient term Light Colour: < 0.5, 0.5, 0.5 > Direction: < -1.5, -1.5, -1.5 > Light 2 Colour: < 0.8, 0.8, 0.9 > Direction: < 1.5, 1.5, -1.5 > Light 3 Colour: < 0.9, 0.9, 0.9 > Direction: < -0.5, 0, 2.0 > Ambient Colour: < 0.5, 0.5, 0.5 >
Primary	One red, one green, and one blue infinite light, no ambient term Light 1 Colour: < 1, 0.2, 0.5 > Direction: < -2, -1.5, -0.5 > Light 2 Colour: < 0.2, 1.0, 0.5 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < 0, 0, 1 > Direction: < 0.0, 0.0, 2.0 >
Blue	Three blue infinite lights, no ambient term Light 1 Colour: < 0.4, 0.4, 0.7 > Direction: < -2.0, -1.5, -0.5 > Light 2 Colour: < 0.75, 0.75, 0.95 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < 0.7, 0.7, 0.95 > Direction: < 0.0, 0.0, 2.0 >
Red	Three red infinite lights, no ambient term Light 1 Colour: < 0.8, 0.3, 0.4 > Direction: < -2.0, -1.5, -0.5 > Light 2 Colour: < 0.95, 0.5, 0.7 > Direction: < 2.0, 1.1, -2.5 > Light 3 Colour: < 0.95, 0.4, 0.5 > Direction: < 0.0, 0.0, 2.0 >
Cube	Six grey infinite lights aligned with the major axes, no ambient term Light 1 Colour: < .4, .4, .4 > Direction: < 1.0, 0.01, 0.01 > Light 2 Colour: < .4, .4, .4 > Direction: < 0.01, 1.0, 0.01 > Light 3 Colour: < .4, .4, .4 > Direction: < 0.01, 0.01, 1.0 > Light 4 Colour: < .4, .4, .4 > Direction: < -1.0, 0.01, 0.01 > Light 5 Colour: < .4, .4, .4 > Direction: < 0.01, -1.0, 0.01 > Light 6 Colour: < .4, .4, .4 > Direction: < 0.01, 0.01, -1.0 >
CAD	Three grey infinite lights and one light attached to the camera, no ambient term Light 1 Colour: < 0.72, 0.72, 0.81 > Direction: < 0.0, 0.0, 0.0 > Light 2 Colour: < 0.2, 0.2, 0.2 > Direction: < -2.0, -1.5, -0.5 > Light 3 Colour: < 0.32, 0.32, 0.32 > Direction: < 2.0, 1.1, -2.5 > Light 4 Colour: < 0.36, 0.36, 0.36 > Direction: < 0.04, 0.01, 2.0 >
Headlamp	Single infinite light attached to the camera, low ambient term Light 1 Colour: < 0.8, 0.8, 0.9 > Direction: < 0.0, 0.0, 0.0 > Ambient Colour: < 0.1, 0.1, 0.1 >

NOTE If a lighting scheme style is encountered other than those described in Table 310, the lighting scheme dictionary containing that entry shall be ignored. This allows future documents using new lighting schemes to behave consistently with future documents using new 3D view constructs. That is, the expected behaviour is for the conforming reader to ignore unrecognized lighting styles and 3D view constructs.

13.6.4.6 3D Cross Section Dictionaries

A *3D cross section dictionary (PDF 1.7)* specifies how a portion of the 3D artwork shall be clipped for the purpose of showing artwork cross sections. The **SA** entry of a 3D view may specify multiple 3D cross section dictionaries.

NOTE Cross sections enable conforming readers to display otherwise hidden parts of the artwork. They also allow users to comment on cross sections, using markup annotations. For example, markup annotations can be used to apply markup annotations to a cross section or to measure distances in a cross section. If multiple cross sections are specified for a view, the markup annotations in the view apply to all cross sections in the view.

Table 311 shows the entries in a 3D cross section dictionary.

Table 311 – Entries in a 3D cross section dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DCrossSection** for a 3D cross section dictionary.
C	array	*(Optional)* A three element array specifying the center of rotation on the cutting plane in world space coordinates (see 13.6.5, "Coordinate Systems for 3D"). Default value: [0 0 0] specifying a cutting plane rotating about the origin of the world space.
O	array	*(Required)* A three-element array specifying the orientation of the cutting plane in world space, where each value represents the orientation in relation to the X, Y, and Z axes, respectively (see 13.6.5, "Coordinate Systems for 3D"). Exactly one of the values shall be **null**, indicating an initial state of the cutting plane that is perpendicular to the corresponding axis and clipping all geometry on the positive side of that axis. The other two values shall be numbers indicating the rotation of the plane, in degrees, around their corresponding axes. The order in which these rotations are applied shall match the order in which the values appear in the array. Default value: [null 0 0] specifying a cutting plane that is perpendicular to the X axis and coplanar with the Y and Z axes.
PO	number	*(Optional)* A number in the range [0, 1] indicating the opacity of the cutting plane using a standard additive blend mode. Default value: 0.5
PC	array	*(Optional)* An array that specifies the colour for the cutting plane. The first entry in the array is a colour space, and the remaining entries are values in that colour space. The only valid colour space is **DeviceRGB**. If a colour space other than **DeviceRGB** is specified, this entry shall be ignored and the default value shall be used. Default value: [/DeviceRGB 1 1 1] representing the colour white.
IV	boolean	*(Optional)* A flag indicating the visibility of the intersection of the cutting plane with any 3D geometry. If **true**, then the intersection shall be visible. If **false**, then the intersection shall not be visible. Default value: **false**
IC	array	*(Optional)* An array that specifies the colour for the cutting plane's intersection with the 3D artwork. The first entry in the array is a colour space, and the remaining entries are values in that colour space. The only valid colour space is **DeviceRGB**. If a colour space other than **DeviceRGB** is specified, this entry shall be ignored and the default value shall be used. This entry is meaningful only if **IV** is **true**. Default value: [/DeviceRGB 0 1 0] representing the colour green.

The **C** entry specifies the center of the cutting plane. This implies that the plane passes through the center point, but it is also the point of reference when determining the orientation of the plane.

The **O** array indicates the orientation of the cutting plane, taking into account its center. The orientation may be determined by a two-step process:

- The plane shall be situated such that it passes through point **C**, and oriented such that it is perpendicular to the axis specified by the array entry whose value is null.

- For each of the other two axes, the plane shall be rotated the specified number of degrees around the associated axis, while maintaining **C** as a fixed point on the plane. Since the two axes are perpendicular, the order in which the rotations are performed is irrelevant.

The **PO** entry specifies the opacity of the plane itself when rendered, while the **PC** entry provides its colour. When the **PO** entry is greater than 0, a visual representation of the cutting plane shall be rendered with the 3D artwork. This representation is a square with a side length equal to the length of the diagonal of the maximum bounding box for the 3D artwork, taking into account any keyframe animations present. When the **PO** entry is 0, no visible representation of the cutting plane shall be rendered.

The **IV** entry shall be a boolean value that determines whether a visual indication shall be drawn of the plane's intersection with the 3D artwork. If such an indication is drawn, the **IC** entry shall specify its colour.

EXAMPLE The following example describes a set of views and corresponding cross sections that illustrate the various effects of orientation.

```
3 0 obj                 %CrossSection1
  <<
    /Type /3DCrossSection
    /C [0 0 0]
    /O [null 0 0]
    /PO 0.35
    /PC [/DeviceRGB 0.75 0.86 1]
    /IV true
    /IC [/DeviceRGB 0 1 0]
  >>
endobj

4 0 obj                 %CrossSection2
  <<
    /Type /3DCrossSection
    /C [0 0 0]
    /O [null -30 0]
    /PO 0.35
    /PC [/DeviceRGB 0.75 0.86 1]
    /IV true
    /IC [/DeviceRGB 0 1 0]
  >>
endobj

5 0 obj                 %CrossSection3
  <<
    /Type /3DCrossSection
    /C [0 0 0]
    /O [null 0 30]
    /PO 0.35
    /PC [/DeviceRGB 0.75 0.86 1]
    /IV true
    /IC [/DeviceRGB 0 1 0]
  >>
endobj

6 0 obj                 %CrossSection4
  <<
    /Type /3DCrossSection
```

```
        /C [0 0 0]
        /O [null -30 30]
        /PO 0.35
        /PC [/DeviceRGB 0.75 0.86 1]
        /IV true
        /IC [/DeviceRGB 0 1 0]
    >>
endobj

7 0 obj                %View0
    <<
        /Type /3DView
        /XN (NoCrossSection)
        /SA []
        ...
    >>
endobj

8 0 obj             %View1
    <<
        /Type /3DView
        /XN (CrossSection1)
        /SA [3 0 R]
        ...
    >>
endobj

9 0 obj             %View2
    <<
        /Type /3DView
        /XN (CrossSection2)
        /SA [4 0 R]
        ...
    >>
endobj

10 0 obj            %View3
    <<
        /Type /3DView
        /XN (CrossSection3)
        /SA [5 0 R]
        ...
    >>
endobj

11 0 obj            %View4
    <<
        /Type /3DView
        /XN (CrossSection4)
        /SA [6 0 R]
        ...
    >>
endobj
```

The following illustrations show the views described in the previous example, some of which include cross sections.

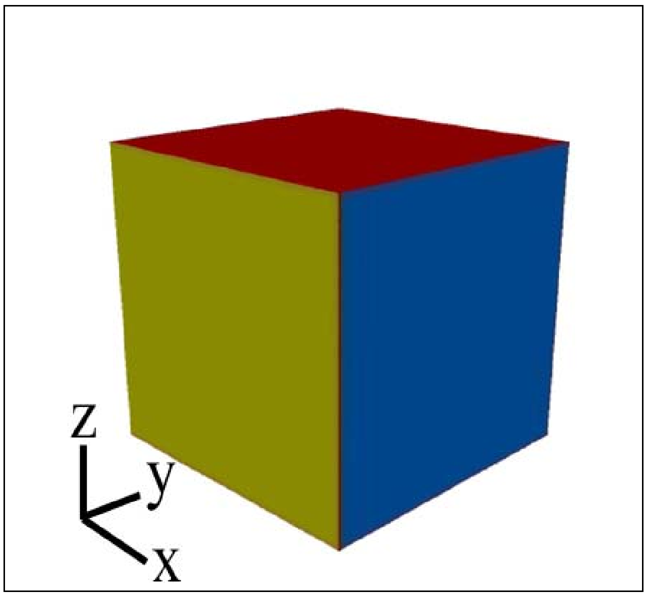


Figure 74 – Rendering of the 3D artwork using View0 (no cross section)

Figure 74 through Figure 78 use world coordinates whose origin is the center of the cube. The axes illustrated in each diagram show the relative orientation of the world coordinate axes, not the actual position of those axes. These axes are not part of the 3D artwork used in this example.

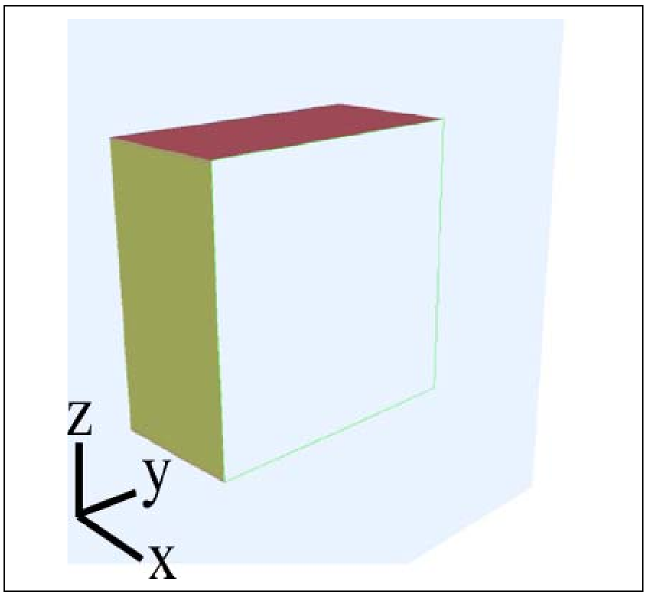


Figure 75 – Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)

Figure 75 shows the cross section specified for the **3DView** that references CrossSection1. The illustration shows the edges of the cutting plane ending at the edges of the annotation's rectangle. This cross section specifies a plane with the following characteristics:

- Includes the world art origin: /C [0 0 0]
- Perpendicular to the X axis and parallel to the Y and Z axes: /O [null 0 0]
- Opacity of the cutting plane is 35%: /PO 0.35

- Colour of the cutting plane is light blue: /PC [/DeviceRGB 0.75 0.86 1]
- Intersection of the cutting plane with the object is visible: /IV true
- Colour of the intersection of the cutting plane and the object is green:
  /IC [/DeviceRGB 0 1 0]

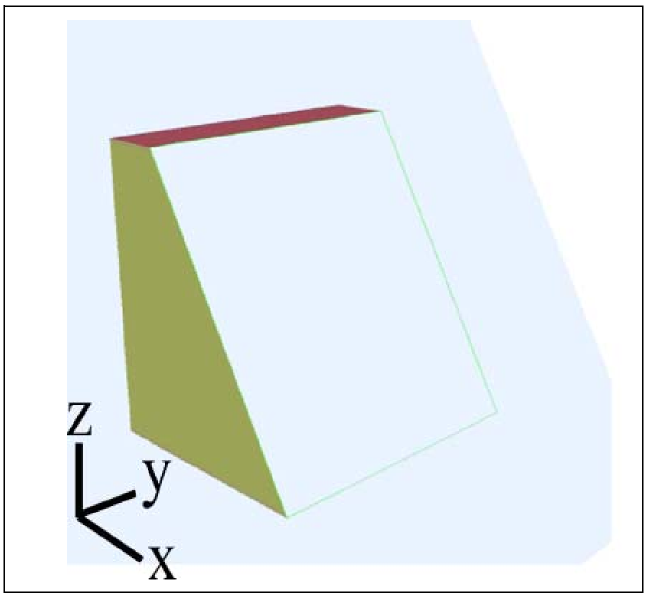


Figure 76 – Rendering of the 3D artwork using View2 (cross section rotated around the y axis by -30 degrees)

Figure 76 shows the cross section specified for the **3DView** that references CrossSection2. This cross section specifies a plane that differs from the one specified in CrossSection1 (Figure 75) in the following way:

- Perpendicular to the *X* axis, rotated -30 degrees around the *Y* axis, and parallel to the *Z* axis: /O [null -30 0]

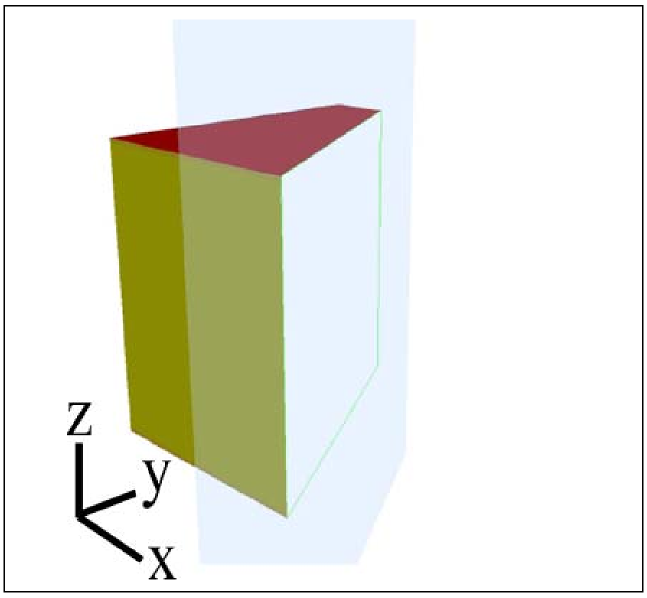


Figure 77 – Rendering of the 3D artwork using View3 (cross section rotated around the z axis by 30 degrees)

Figure 77 shows the cross section specified for the **3DView** that references CrossSection3. This cross section specifies a plane that differs from the one specified in CrossSection1 (Figure 75) in the following way:

- Perpendicular to the *X* axis, parallel to the *Y* axis, and rotated 30 degrees around the *Z* axis: /O [null 0 30]

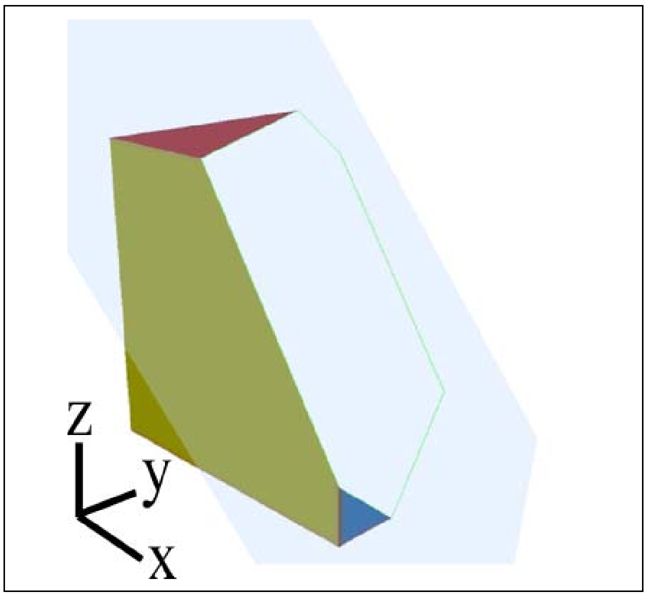


Figure 78 – Rendering of the 3D artwork using View4
(cross section rotated around the y axis by -30 degrees and around the z axis by 30 degrees)

Figure 78 shows the cross section specified for the **3DView** that references CrossSection4. This cross section specifies a plane that differs from the one specified in CrossSection1 (Figure 75) in the following way:

- Perpendicular to the *X* axis, rotated -30 degrees around the *Y* axis, and rotated 30 degrees around the *Z* axis: /O [null -30 30]

13.6.4.7 3D Node Dictionaries

A 3D view may specify a 3D node dictionary *(PDF 1.7)*, which specifies particular areas of 3D artwork and the opacity and visibility with which individual nodes shall be displayed. The 3D artwork shall be contained in the parent 3D stream object. The **NA** entry of the 3D views dictionary may specify multiple 3D node dictionaries for a particular view.

NOTE 1 While many PDF dictionaries reference 3D artwork in its entirety, it is often useful to reference 3D artwork at a more granular level. This enables properties such as visibility, opacity, and orientation to be applied to subsets of the 3D artwork. These controls enable underlying nodes to be revealed, by making the overlying nodes transparent or by moving them out of the way.

NOTE 2 Do not confuse nodes with view nodes. A node is a PDF dictionary that specifies an area in 3D artwork, while a view node is a parameter in the 3D artwork that specifies a view.

Table 312 shows the entries in a 3D node dictionary.

Table 312 – Entries in a 3D node dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **3DNode** for a 3D node dictionary.
N	text string	*(Required)* The name of the node being described by the node dictionary. If the **Subtype** of the corresponding 3D Stream is **U3D**, this entry corresponds to the field Node block name, as described in the *Universal 3D file format* specification (see Bibliography). In the future, nodes may be described using other 3D conventions. NOTE When comparing this entry to node names for a particular convention (such as Universal 3D), conforming readers shall translate between the PDF text encoding used by PDF and the character encoding specified in the 3D stream.
O	number	*(Optional)* A number in the range [0, 1] indicating the opacity of the geometry supplied by this node using a standard additive blend mode. If this entry is absent, the viewer shall use the opacity specified for the parent node or for the 3D artwork (in ascending order).
V	boolean	*(Optional)* A flag indicating the visibility of this node. If **true**, then the node is visible. If **false**, then the node shall not be visible. If this entry shall be absent, the viewer shall use the visibility specified for the parent node or for the 3D artwork (in ascending order).
M	array	*(Optional)* A 12-element 3D transformation matrix that specifies the position and orientation of this node, relative to its parent, in world coordinates (see 13.6.5, “Coordinate Systems for 3D”).

The **N** entry specifies which node in the 3D stream corresponds to this node dictionary.

The **O** entry describes the opacity that shall be used when rendering this node, and the **V** entry shall determine whether or not the node is rendered at all. While a node with an opacity of 0 shall be rendered in the same way as a non-visible node, having a separate value for the visibility of a node allows interactive conforming readers to show/hide partially transparent nodes, without overwriting the intended opacity of those nodes.

The **M** entry specifies the node’s matrix relative to its parent, in world coordinates. If an hierarchy of nodes is intended to be repositioned while still maintaining its internal structure, then only the node at the root of the hierarchy needs to be adjusted.

EXAMPLE The following example shows a 3D view specifying an array of node parameters.

```
3 0 obj             % Default node params with all shapes visible and opaque
   [   << /Type /3DNode
          /N (Sphere)
          /O 1
          /V  true
          /M [...]>>
       << /Type /3DNode
          /N (Cone)
          /O 1
          /V true >>
```

```
        << /Type /3DNode
            /N (Cube)
            /O 1
            /V true >>
    ]

4 0 obj             % Params with the cone hidden and the sphere semi-transparent
    [   << /Type /3DNode
            /N (Sphere)
            /O 0.5
            /V  true >>
        << /Type /3DNode
            /N (Cone)
            /O 1
            /V false >>
        << /Type /3DNode
            /N (Cube)
            /O 1
            /V true >>
    ]
endobj

5 0 obj                 %View1, using the default set of node params
    <<
        /Type /3DView
        /XN (View1)
        /NA 3 0 R
        ...
    >>
endobj

6 0 obj                 %View2, using the alternate set of node params
    <<
        /Type /3DView
        /XN (View2)
        /NA 4 0 R
        ...
    >>
endobj
```

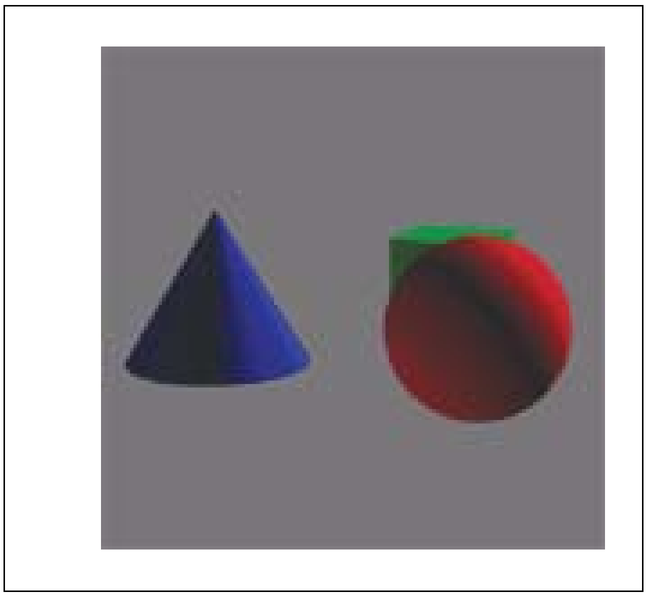


Figure 79 – Rendering of the 3D artwork using View1 (all shapes visible and opaque)

Figure 79 shows a view whose node array includes three nodes, all of which shall be rendered with the appearance opaque (/O 1) and visible (/V true).

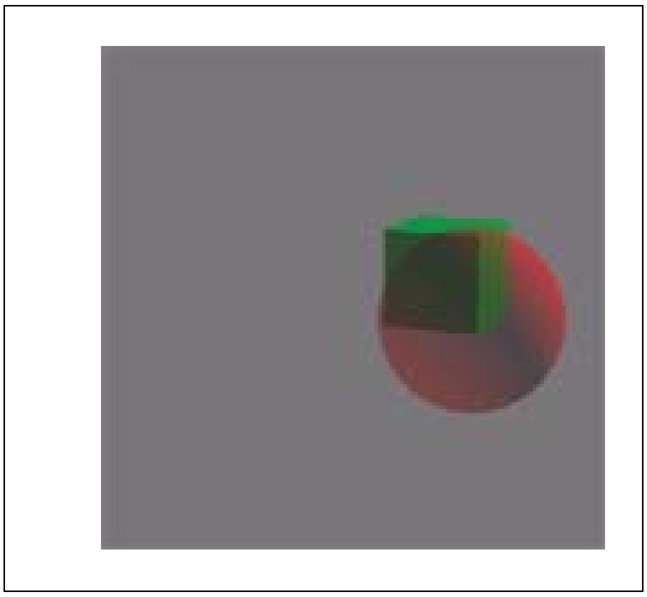


Figure 80 – Rendering of the 3D artwork using View2 (the cone is hidden and the sphere is semi-transparent)

Figure 80 shows a view with a node array that specifies the same three nodes used in Figure 79. These nodes have the following display characteristics:

- The node named Sphere is partially transparent (/O 0.5) and visible (/V true)
- The node named Cone is opaque (/O 1) and invisible (/V false)
- The node named Cube is opaque (/O 1) and visible (/V true)

13.6.5 Coordinate Systems for 3D

3D artwork is a collection of objects whose positions and geometry shall be specified using three-dimensional coordinates. 8.3, “Coordinate Systems,” discusses the concepts of two-dimensional coordinate systems, their geometry and transformations. This sub-clause extends those concepts to include the third dimension.

As described in 8.3, “Coordinate Systems,” positions shall be defined in terms of pairs of *x* and *y* coordinates on the Cartesian plane. The origin of the plane specifies the location (0, 0); *x* values increase to the right and *y* values increase upward. For three-dimensional graphics, a third axis, the *z* axis, shall be used. The origin shall be at (0, 0, 0); positive *z* values increase going into the page.

In two-dimensional graphics, the transformation matrix transforms the position, size, and orientation of objects in a plane. It is a 3-by-3 matrix, where only six of the elements may be changed; therefore, the matrix shall be expressed in PDF as an array of six numbers:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ tx & ty & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d & tx & ty \end{bmatrix}$$

In 3D graphics, a 4-by-4 matrix shall be used to transform the position, size, and orientations of objects in a three-dimensional coordinate system. Only the first three columns of the matrix may be changed; therefore, the matrix shall be expressed in PDF as an array of 12 numbers:

$$\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ tx & ty & tz & 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d & e & f & g & h & i & tx & ty & tz \end{bmatrix}$$

3D coordinate transformations shall be expressed as matrix transformations:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \times \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

Carrying out the multiplication has the following results:

$$x' = a \times x + d \times y + g \times z + tx$$

$$y' = b \times x + e \times y + h \times z + ty$$

$$z' = c \times x + f \times y + i \times z + tz$$

Position and orientation of 3D artwork typically involves translation (movement) and rotation along any axis. The virtual camera represents the view of the artwork. The relationship between camera and artwork may be thought of in two ways:

- The 3D artwork is in a fixed position and orientation, and the camera moves to different positions and orientations.

- The camera is in a fixed location, and the 3D artwork is translated and rotated.

Both approaches may achieve the same visual effects; in practice, 3D systems typically use a combination of both. Conceptually, there are three distinct coordinate systems:

- The *artwork coordinate system*.
- The *camera coordinate system*, in which the camera shall be positioned at (0, 0, 0) facing out along the positive *z* axis, with the positive *x* axis to the right and the positive *y* axis going straight up.
- An intermediate system called the *world coordinate system*.

Two 3D transformation matrices shall be used in coordinate conversions:

- The *artwork-to-world matrix* specifies the position and orientation of the artwork in the world coordinate system. This matrix shall be contained in the 3D stream.
- The *camera-to-world matrix* specifies the position and orientation of the camera in the world coordinate system. This matrix shall be specified by either the **C2W** or **U3DPath** entries of the 3D view dictionary.

When drawing 3D artwork in a 3D annotation’s target coordinate system, the following transformations take place:

a) Artwork coordinates shall be transformed to world coordinates:

$$\begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix} = \begin{bmatrix} x_a & y_a & z_a & 1 \end{bmatrix} \times aw$$

b) World coordinates shall be transformed to camera coordinates:

$$\begin{bmatrix} x_c & y_c & z_c & 1 \end{bmatrix} = \begin{bmatrix} x_w & y_w & z_w & 1 \end{bmatrix} \times (cw^{-1})$$

c) The first two steps can be expressed as a single equation, as follows:

$$\begin{bmatrix} x_c & y_c & z_c & 1 \end{bmatrix} = \begin{bmatrix} x_a & y_a & z_a & 1 \end{bmatrix} \times (aw \times cw^{-1})$$

d) Finally, the camera coordinates shall be projected into two dimensions, eliminating the *z* coordinate, then scaled and positioned within the annotation’s target coordinate system.

13.6.6 3D Markup

Beginning with PDF 1.7, users may comment on specific views of 3D artwork by using markup annotations (see 12.5.6.2, “Markup Annotations”). Markup annotations (as other annotations) are normally associated with a location on a page. To associate the markup with a specific view of a 3D annotation, the annotation dictionary for the markup annotation contains an **ExData** entry (see Table 174) that specifies the 3D annotation and view. Table 313 describes the entries in an external data dictionary used to markup 3D annotations.

Table 313 – Entries in an external data dictionary used to markup 3D annotations

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; if present, shall be **ExData** for an external data dictionary.
Subtype	name	*(Required)* The type of external data that this dictionary describes; shall be **Markup3D** for a 3D comment. The only defined value is **Markup3D**.
3DA	dictionary or text string	*(Required)* The 3D annotation to which this markup annotation applies. The 3D annotation may be specified as a child dictionary or as the name of a 3D annotation, as specified by its **NM** entry. In the latter case, the 3D annotation and the markup annotation shall be on the same page of the document.
3DV	dictionary	*(Required)* The 3D view that this markup annotation is associated with. The annotation will be hidden unless this view is currently being used for the 3D annotation specified by **3DA**.
MD5	byte string	*(Optional)* A 16-byte string that contains the checksum of the bytes of the 3D stream data that this 3D comment shall be associated with. The checksum shall be calculated by applying the standard MD5 message-digest algorithm (described in Internet RFC 1321, *The MD5 Message-Digest Algorithm*; see the Bibliography) to the bytes of the stream data. This value shall be used to determine if artwork data has changed since this 3D comment was created.

In a **Markup3D ExData** dictionary, the **3DA** entry identifies the 3D annotation to which the markup shall be associated. Even though the markup annotation exists alongside the associated annotation in the page's **Annots** array, the markup may be thought of as a child of the **3DA** annotation.

The **3DV** entry specifies the markup's associated 3D view. The markup shall only be printed and displayed when the specified view is the current view of its parent 3D annotation. This ensures that the proper context is preserved when the markup is displayed.

NOTE An equivalent view is not sufficient; if more than one markup specify equivalent views represented by different objects, the markups will not display simultaneously.

The **MD5** entry gives conforming readers a means to detect whether or not the 3D stream of the 3D annotation specified by **3DA** has changed. If the 3D stream has changed, the context provided by the **3DV** entry may no longer apply, and the markup may no longer be useful. Any action taken as a response to such a situation is dependent on the conforming reader, but a warning shall be issued to the user.

EXAMPLE The following example shows how markup annotations can be associated with particular views.

```
2 0 obj          % 3D stream data with two named views
  <<
    /Type /3D
    /Subtype /U3D
    /VA [4 0 R 5 0 R]
    ...
  >>
stream
  ...
endstream
endobj

3 0 obj          % 3D annotation
  <<
    /Type /Annot
    /Subtype /3D
    /3DD 2 0 R
```

```
        ...
    >>
endobj

4 0 obj           % CommentView1
    <<
        /Type /3DView
        /XN (CommentView1)
        ...
    >>
endobj

5 0 obj           % CommentView2
    <<
        /Type /3DView
        /XN (CommentView2)
        ...
    >>
endobj

6 0 obj           % Cloud comment with no ExData
    <<
        /Type /Annot
        /Subtype /Polygon
        /IT /PolygonCloud
        ...
    >>
endobj

7 0 obj           % Callout comment on CommentView1
    <<
        /Type /Annot
        /Subtype /FreeText
        /IT /FreeTextCallout
        /ExData <<
            /Type /Markup3D
            /3DA 3 0 R
            /3DV 4 0 R
        >>
        ...
    >>
endobj

8 0 obj           % Dimension comment on CommentView2
    <<
        /Type /Annot
        /Subtype /Line
        /IT /LineDimension
        /ExData <<
            /Type /Markup3D
            /3DA 3 0 R
            /3DV 5 0 R
        >>
        ...
    >>
```

```
          endobj

          9 0 obj          % Stamp comment on CommentView2
              <<
                  /Type /Annot
                  /Subtype /Stamp
                  /ExData <<
                      /Type /Markup3D
                      /3DA 3 0 R
                      /3DV 5 0 R
                  >>
                  ...
              >>
endobj
```

The following illustrations show the placement of markup on annotations on different views of the same 3D artwork.

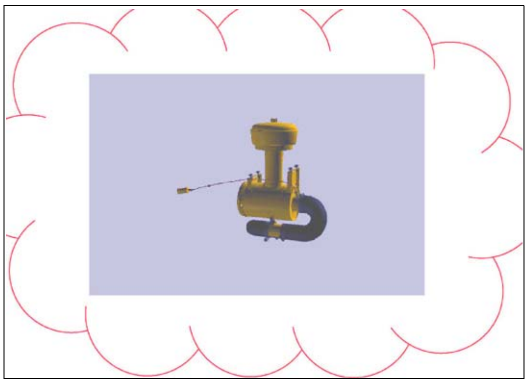


Figure 81 – 3D artwork set to its default view

Figure 81 shows the default view, which has no markup annotations.

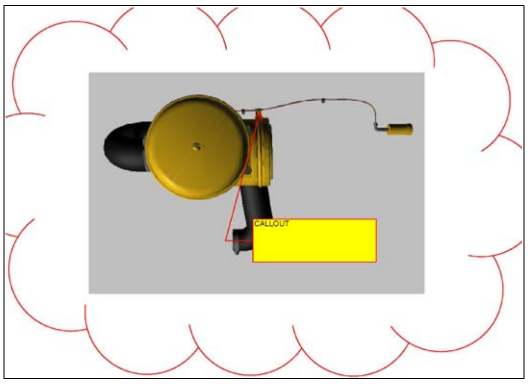


Figure 82 – 3D artwork set to CommentView1

Figure 82 shows another view to which a markup annotation is applied.

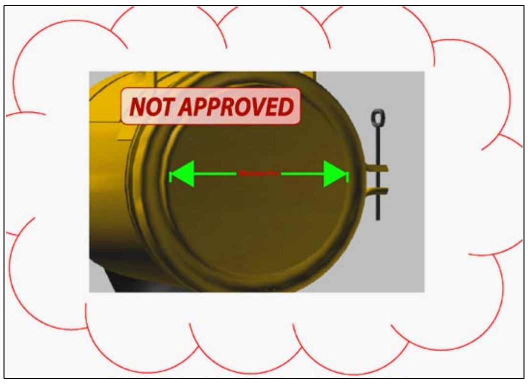


Figure 83 – 3D artwork set to CommentView2

Figure 83 shows a view referenced by two markup annotations:

- A line annotation (/Subtype /Line) with a line dimension intent (/IT/ LineDimension)
- A stamp annotation (/Subtype /Stamp)

14 Document Interchange

14.1 General

The features described in this clause do not affect the final appearance of a document. Rather, these features enable a document to include higher-level information that is useful for the interchange of documents among conforming products:

- *Procedure sets* (14.2, “Procedure Sets”) that define the implementation of PDF operators
- *Metadata* (14.3, “Metadata”) consisting of general information about a document or a component of a document, such as its title, author, and creation and modification dates
- *File identifiers* (14.4, “File Identifiers”) for reliable reference from one PDF file to another
- *Page-piece dictionaries* (14.5, “Page-Piece Dictionaries”) allowing a conforming product to embed private data in a PDF document for its own use
- *Marked-content* operators (14.6, “Marked Content”) for identifying portions of a content stream and associating them with additional properties or externally specified objects
- *Logical structure* facilities (14.7, “Logical Structure”) for imposing a hierarchical organization on the content of a document
- *Tagged PDF* (14.8, “Tagged PDF”), a set of conventions for using the marked content and logical structure facilities to facilitate the extraction and reuse of a document’s content for other purposes
- Various ways of increasing the *accessibility* of a document to users with disabilities (14.9, “Accessibility Support”), including the identification of the natural language in which it is written (such as English or Spanish) for the benefit of a text-to-speech engine
- The *Web Capture* extension (14.10, “ Web Capture”), which creates PDF files from Internet-based or locally resident HTML, PDF, GIF, JPEG, and ASCII text files
- Facilities supporting prepress production workflows (14.11, “Prepress Support”), such as the specification of *page boundaries* and the generation of *printer’s marks*, *colour separations*, *output intents*, *traps*, and low-resolution *proxies* for high-resolution images

14.2 Procedure Sets

The PDF operators used in content streams are grouped into categories of related operators called *procedure sets* (see Table 314). Each procedure set corresponds to a named resource containing the implementations of the operators in that procedure set. The **ProcSet** entry in a content stream’s resource dictionary (see 7.8.3, “Resource Dictionaries”) shall hold an array consisting of the names of the procedure sets used in that content stream. These procedure sets shall be used only when the content stream is printed to a PostScript output device. The names identify PostScript procedure sets that shall be sent to the device to interpret the PDF operators in the content stream. Each element of this array shall be one of the predefined names shown in Table 314.

Table 314 – Predefined procedure set

Name	**Category of operators**
PDF	Painting and graphics state
Text	Text
ImageB	Grayscale images or image masks

Table 314 – Predefined procedure set (continued)

Name	Category of operators
ImageC	Colour images
ImageI	Indexed (colour-table) images

Beginning with PDF 1.4, this feature is considered obsolete. For compatibility with existing conforming readers, conforming writers should continue to specify procedure sets (preferably, all of those listed in Table 314 unless it is known that fewer are needed). However, conforming readers should not depend on the correctness of this information.

14.3 Metadata

14.3.1 General

A PDF document may include general information, such as the document's title, author, and creation and modification dates. Such global information about the document (as opposed to its content or structure) is called *metadata* and is intended to assist in cataloguing and searching for documents in external databases. Beginning with PDF 1.4, metadata may also be specified for individual components of a document.

Metadata may be stored in a PDF document in either of the following ways:

- In a *metadata stream (PDF 1.4)* associated with the document or a component of the document (14.3.2, "Metadata Streams")
- In a *document information dictionary* associated with the document (14.3.3, "Document Information Dictionary")

NOTE Document information dictionaries is the original way that metadata was included in a PDF file. Metadata streams were introduced in PDF 1.4 and is now the preferred method to include metadata.

14.3.2 Metadata Streams

Metadata, both for an entire document and for components within a document, may be stored in PDF streams called *metadata streams (PDF 1.4)*.

NOTE 1 Metadata streams have the following advantages over the document information dictionary:

- PDF-based workflows often embed metadata-bearing artwork as components within larger documents. Metadata streams provide a standard way of preserving the metadata of these components for examination downstream. PDF-aware conforming products should be able to derive a list of all metadata-bearing document components from the PDF document itself.
- PDF documents are often made available on the Web or in other environments, where many tools routinely examine, catalogue, and classify documents. These tools should be able to understand the self-contained description of the document even if they do not understand PDF.

Besides the usual entries common to all stream dictionaries (see Table 5), the metadata stream dictionary shall contain the additional entries listed in Table 315.

The contents of a metadata stream shall be the metadata represented in Extensible Markup Language (XML).

NOTE 2 This information is visible as plain text to tools that are not PDF-aware only if the metadata stream is both unfiltered and unencrypted.

Table 315 – Additional entries in a metadata stream dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **Metadata** for a metadata stream.
Subtype	name	*(Required)* The type of metadata stream that this dictionary describes; shall be **XML**.

NOTE 3 The format of the XML representing the metadata is defined as part of a framework called the Extensible Metadata Platform (XMP) and described in the Adobe document *XMP: Extensible Metadata Platform* (see the Bibliography). This framework provides a way to use XML to represent metadata describing documents and their components and is intended to be adopted by a wider class of products than just those that process PDF. It includes a method to embed XML data within non-XML data files in a platform-independent format that can be easily located and accessed by simple scanning rather than requiring the document file to be parsed.

A metadata stream may be attached to a document through the **Metadata** entry in the document catalogue (see 7.7.2, "Document Catalog"). The metadata framework provides a date stamp for metadata expressed in the framework. If this date stamp is equal to or later than the document modification date recorded in the document information dictionary, the metadata stream shall be taken as authoritative. If, however, the document modification date recorded in the document information dictionary is later than the metadata stream's date stamp, the document has likely been saved by a writer that is not aware of metadata streams. In this case, information stored in the document information dictionary shall be taken to override any semantically equivalent items in the metadata stream. In addition, PDF document components represented as a stream or dictionary may have a **Metadata** entry (see Table 316).

Table 316 – Additional entry for components having metadata

Key	Type	Value
Metadata	stream	*(Optional; PDF 1.4)* A metadata stream containing metadata for the component.

In general, any PDF stream or dictionary may have metadata attached to it as long as the stream or dictionary represents an actual information resource, as opposed to serving as an implementation artifact. Some PDF constructs are considered implementational, and hence may not have associated metadata.

When there is ambiguity about exactly which stream or dictionary may bear the **Metadata** entry, the metadata shall be attached as close as possible to the object that actually stores the data resource described.

NOTE 4 Metadata describing a tiling pattern should be attached to the pattern stream's dictionary, but a shading should have metadata attached to the shading dictionary rather than to the shading pattern dictionary that refers to it. Similarly, metadata describing an **ICCBased** colour space should be attached to the ICC profile stream describing it, and metadata for fonts should be attached to font file streams rather than to font dictionaries.

NOTE 5 In tables describing document components in this specification, the **Metadata** entry is listed only for those in which it is most likely to be used. Keep in mind, however, that this entry may appear in other components represented as streams or dictionaries.

- In addition, metadata may also be associated with marked content within a content stream. This association shall be created by including an entry in the property list dictionary whose key shall be **Metadata** and whose value shall be the metadata stream dictionary. Because this construct refers to an object outside the content stream, the property list is referred to indirectly as a named resource (see 14.6.2, "Property Lists").

14.3.3 Document Information Dictionary

The optional **Info** entry in the trailer of a PDF file (see 7.5.5, "File Trailer") shall hold a *document information dictionary* containing metadata for the document; Table 317 shows its contents. Any entry whose value is not known should be omitted from the dictionary rather than included with an empty string as its value.

Some conforming readers may choose to permit searches on the contents of the document information dictionary. To facilitate browsing and editing, all keys in the dictionary shall be fully spelled out, not abbreviated. New keys should be chosen with care so that they make sense to users.

The value associated with any key not specifically mentioned in Table 317 shall be a text string.

Although conforming readers may store custom metadata in the document information dictionary, they may not store private content or structural information there. Such information shall be stored in the document catalogue instead (see 7.7.2, "Document Catalog").

Table 317 – Entries in the document information dictionary

Key	Type	Value
Title	text string	*(Optional; PDF 1.1*) The document's title.
Author	text string	*(Optional)* The name of the person who created the document.
Subject	text string	*(Optional; PDF 1.1)* The subject of the document.
Keywords	text string	*(Optional; PDF 1.1)* Keywords associated with the document.
Creator	text string	*(Optional)* If the document was converted to PDF from another format, the name of the conforming product that created the original document from which it was converted.
Producer	text string	*(Optional)* If the document was converted to PDF from another format, the name of the conforming product that converted it to PDF.
CreationDate	date	*(Optional)* The date and time the document was created, in human-readable form (see 7.9.4, "Dates").
ModDate	date	*(Required if **PieceInfo** is present in the document catalogue; otherwise optional; PDF 1.1)* The date and time the document was most recently modified, in human-readable form (see 7.9.4, "Dates").
Trapped	name	*(Optional; PDF 1.3*) A name object indicating whether the document has been modified to include trapping information (see 14.11.6, "Trapping Support"): **True** The document has been fully trapped; no further trapping shall be needed. This shall be the name **True**, not the boolean value **true**. **False** The document has not yet been trapped. This shall be the name **False**, not the boolean value **false**. **Unknown** Either it is unknown whether the document has been trapped or it has been partly but not yet fully trapped; some additional trapping may still be needed. Default value: **Unknown**. NOTE The value of this entry may be set automatically by the software creating the document's trapping information, or it may be known only to a human operator and entered manually.

EXAMPLE This example shows a typical document information dictionary.

```
1 0 obj
   << /Title (PostScript Language Reference, Third Edition)
      /Author (Adobe Systems Incorporated)
      /Creator (Adobe FrameMaker 5.5.3 for Power Macintosh®)
      /Producer (Acrobat Distiller 3.01 for Power Macintosh)
      /CreationDate (D:19970915110347-08'00')
      /ModDate (D:19990209153925-08'00')
   >>
endobj
```

14.4 File Identifiers

PDF files may contain references to other PDF files (see 7.11, “File Specifications”). Simply storing a file name, however, even in a platform-independent format, does not guarantee that the file can be found. Even if the file still exists and its name has not been changed, different server software applications may identify it in different ways. Servers running on DOS platforms convert all file names to 8 characters and a 3-character extension. Different servers may use different strategies for converting longer file names to this format.

External file references may be made more reliable by including a *file identifier (PDF 1.1)* in the file and using it in addition to the normal platform-based file designation. Matching the identifier in the file reference with the one in the file confirms whether the correct file was found.

File identifiers shall be defined by the optional **ID** entry in a PDF file’s trailer dictionary (see 7.5.5, “File Trailer”). The ID entry is optional but should be used. The value of this entry shall be an array of two byte strings. The first byte string shall be a permanent identifier based on the contents of the file at the time it was originally created and shall not change when the file is incrementally updated. The second byte string shall be a changing identifier based on the file’s contents at the time it was last updated. When a file is first written, both identifiers shall be set to the same value. If both identifiers match when a file reference is resolved, it is very likely that the correct and unchanged file has been found. If only the first identifier matches, a different version of the correct file has been found.

To help ensure the uniqueness of file identifiers, they should be computed by means of a message digest algorithm such as MD5 (described in Internet RFC 1321, *The MD5 Message-Digest Algorithm*; see the Bibliography), using the following information:

- The current time
- A string representation of the file’s location, usually a pathname
- The size of the file in bytes
- The values of all entries in the file’s document information dictionary (see 14.3.3, “Document Information Dictionary”)

NOTE The calculation of the file identifier need not be reproducible; all that matters is that the identifier is likely to be unique. For example, two implementations of the preceding algorithm might use different formats for the current time, causing them to produce different file identifiers for the same file created at the same time, but the uniqueness of the identifier is not affected.

14.5 Page-Piece Dictionaries

A *page-piece dictionary (PDF 1.3)* may be used to hold private conforming product data. The data may be associated with a page or form XObject by means of the optional **PieceInfo** entry in the page object (see Table 30) or form dictionary (see Table 95). Beginning with PDF 1.4, private data may also be associated with the PDF document by means of the **PieceInfo** entry in the document catalogue (see Table 28).

NOTE 1 Conforming products may use this dictionary as a place to store private data in connection with that document, page, or form. Such private data can convey information meaningful to the conforming product that produces it (such as information on object grouping for a graphics editor or the layer information used by Adobe Photoshop®) but may be ignored by general-purpose conforming readers.

As Table 318 shows, a page-piece dictionary may contain any number of entries, each keyed by the name of a distinct conforming product or of a well-known data type recognized by a family of conforming products. The value associated with each key shall be a *data dictionary* containing the private data that shall be used by the conforming product. The **Private** entry may have a value of any data type, but typically it is a dictionary containing all of the private data needed by the conforming product other than the actual content of the document, page, or form.

Table 318 – Entries in a page-piece dictionary

Key	Type	Value
any conforming product name or well-known data type	dictionary	A data dictionary (see Table 319).

Table 319 – Entries in an data dictionary

Key	Type	Value
LastModified	date	*(Required)* The date and time when the contents of the document, page, or form were most recently modified by this conforming product.
Private	(any)	*(Optional)* Any private data appropriate to the conforming product, typically in the form of a dictionary.

The **LastModified** entry indicates when this conforming product last altered the content of the page or form. If the page-piece dictionary contains several data dictionaries, their modification dates can be compared with those in the corresponding entry of the page object or form dictionary (see Table 30 and Table 95), or the **ModDate** entry of the document information dictionary (see Table 317), to ascertain which data dictionary corresponds to the current content of the page or form. Because some platforms may use only an approximate value for the date and time or may not deal correctly with differing time zones, modification dates shall be compared only for equality and not for sequential ordering.

NOTE 2 It is possible for two or more data dictionaries to have the same modification date. Conforming products can use this capability to define multiple or extended versions of the same data format. For example, suppose that earlier versions of a conforming product use an data dictionary named ***PictureEdit***, and later versions of the same conforming product extend the data to include additional items not previously used. The original data could continue to be kept in the ***PictureEdit*** dictionary and the additional items placed in a new dictionary named ***PictureEditExtended***. This allows the earlier versions of the conforming product to continue to work as before, and later versions are able to locate and use the extended data items.

14.6 Marked Content

14.6.1 General

Marked-content operators (PDF 1.2) may identify a portion of a PDF content stream as a *marked-content element* of interest to a particular conforming product. Marked-content elements and the operators that mark them shall fall into two categories:

- The **MP** and **DP** operators shall designate a single *marked-content point* in the content stream.
- The **BMC**, **BDC**, and **EMC** operators shall bracket a *marked-content sequence* of objects within the content stream.

NOTE 1 This is a sequence not simply of bytes in the content stream but of complete graphics objects. Each object is fully qualified by the parameters of the graphics state in which it is rendered.

NOTE 2 A graphics application, for example, might use marked content to identify a set of related objects as a group to be processed as a single unit. A text-processing application might use it to maintain a connection between a footnote marker in the body of a document and the corresponding footnote text at the bottom of the page. The PDF logical structure facilities use marked-content sequences to associate graphical content with structure elements (see 14.7.4, “Structure Content”). Table 320 summarizes the marked-content operators.

All marked-content operators except **EMC** shall take a *tag* operand indicating the role or significance of the marked-content element to the conforming reader. All such tags shall be registered with Adobe Systems (see

Annex E) to avoid conflicts between different applications marking the same content stream. In addition to the tag operand, the **DP** and **BDC** operators shall specify a *property list* containing further information associated with the marked content. Property lists are discussed further in 14.6.2, "Property Lists."

Marked-content operators may appear only *between* graphics objects in the content stream. They may not occur within a graphics object or between a graphics state operator and its operands. Marked-content sequences may be nested one within another, but each sequence shall be entirely contained within a single content stream.

NOTE 3 A marked-content sequence may not cross page boundaries.

NOTE 4 The ***Contents*** entry of a page object (see 7.7.3.3, "Page Objects"), which may be either a single stream or an array of streams, is considered a single stream with respect to marked-content sequences.

Table 320 – Marked-content operators

Operands	Operator	Description
tag	**MP**	Designate a marked-content point. *tag* shall be a name object indicating the role or significance of the point.
tag properties	**DP**	Designate a marked-content point with an associated property list. *tag* shall be a name object indicating the role or significance of the point. properties shall be either an inline dictionary containing the property list or a name object associated with it in the **Properties** subdictionary of the current resource dictionary (see 14.6.2, "Property Lists").
tag	**BMC**	Begin a marked-content sequence terminated by a balancing **EMC** operator. *tag* shall be a name object indicating the role or significance of the sequence.
tag properties	**BDC**	Begin a marked-content sequence with an associated property list, terminated by a balancing **EMC** operator. *tag* shall be a name object indicating the role or significance of the sequence. *properties* shall be either an inline dictionary containing the property list or a name object associated with it in the **Properties** subdictionary of the current resource dictionary (see 14.6.2, "Property Lists").
—	**EMC**	End a marked-content sequence begun by a **BMC** or **BDC** operator.

When the marked-content operators **BMC**, **BDC**, and **EMC** are combined with the text object operators **BT** and **ET** (see 9.4, "Text Objects"), each pair of matching operators (**BMC**…**EMC**, **BDC**…**EMC**, or **BT**…**ET**) shall be properly (separately) nested. Therefore, the sequences

```
BMC                                   BT
   BT                                    BMC
      …           and                       …
   ET                                    EMC
BMC                                   ET
```

are valid, but

```
BMC                                   BT
   BT                                    BMC
      …           and                       …
   EMC                                   ET
BT                                    EMC
```

are not valid.

14.6.2 Property Lists

The marked-content operators **DP** and **BDC** associate a *property list* with a marked-content element within a content stream. The property list is a dictionary containing private information meaningful to the conforming writer creating the marked content. Conforming products should use the dictionary entries in a consistent way; the values associated with a given key should always be of the same type (or small set of types).

If all of the values in a property list dictionary are direct objects, the dictionary may be written inline in the content stream as a direct object. If any of the values are indirect references to objects outside the content stream, the property list dictionary shall be defined as a named resource in the **Properties** subdictionary of the current resource dictionary (see 7.8.3, “Resource Dictionaries”) and referenced by name as the *properties* operand of the **DP** or **BDC** operator.

14.6.3 Marked Content and Clipping

Some PDF path and text objects are defined purely for their effect on the current clipping path, without the objects actually being painted on the page. This occurs when a path object is defined using the operator sequence **W n** or **W* n** (see 8.5.4, “Clipping Path Operators”) or when a text object is painted in text rendering mode 7 (see 9.3.6, “Text Rendering Mode”). Such clipped, unpainted path or text objects are called *clipping objects*. When a clipping object falls within a marked-content sequence, it shall not be considered part of the sequence unless the entire sequence consists only of clipping objects. In Example 1, for instance, the marked-content sequence tagged *Clip* includes the text string (Clip me) but not the rectangular path that defines the clipping boundary.

EXAMPLE 1

```
/Clip  BMC
    100  100  10  10  re  W  n                   % Clipping path
    (Clip me)  Tj                                % Object to be clipped
EMC
```

Only when a marked-content sequence consists entirely of clipping objects shall the clipping objects be considered part of the sequence. In this case, the sequence is known as a *marked clipping sequence*. Such sequences may be nested. In Example 2, for instance, multiple lines of text are used to clip a subsequent graphics object (in this case, a filled path). Each line of text shall be bracketed within a separate marked clipping sequence, tagged *Pgf*. The entire series shall be bracketed in turn by an outer marked clipping sequence, tagged *Clip*.

NOTE The marked-content sequence tagged ClippedText is *not* a marked clipping sequence, since it contains a filled rectangular path that is not a clipping object. The clipping objects belonging to the Clip and Pgf sequences are therefore not considered part of the ClippedText sequence.

EXAMPLE 2

```
/ClippedText  BMC
    /Clip  <<…>>
        BDC
            BT
                7  Tr                            % Begin text clip mode
                /Pgf  BMC
                    (Line 1)  Tj
                EMC
                /Pgf  BMC
                    (Line)  '
                    ( 2)  Tj
                EMC
            ET                                   % Set current text clip
        EMC
    100  100  10  10  re  f                      % Filled path
EMC
```

The precise rules governing marked clipping sequences shall be as follows:

- A *clipping object* shall be a path object ended by the operator sequence **W n** or **W* n** or a text object painted in text rendering mode 7.

- An *invisible graphics object* shall be a path object ended by the operator **n** only (with no preceding **W** or **W***) or a text object painted in text rendering mode 3.

- A *visible graphics object* shall be a path object ended by any operator other than **n**, a text object painted in any text rendering mode other than 3 or 7, or any XObject invoked by the **Do** operator.

- An *empty marked-content element* shall be a marked-content point or a marked-content sequence that encloses no graphics objects.

- A *marked clipping sequence* shall be a marked-content sequence that contains at least one clipping object and no visible graphics objects.

- Clipping objects and marked clipping sequences shall be considered part of an enclosing marked-content sequence only if it is a marked clipping sequence.

- Invisible graphics objects and empty marked-content elements shall always be considered part of an enclosing marked-content sequence, regardless of whether it is a marked clipping sequence.

- The **q** (save) and **Q** (restore) operators may not occur within a marked clipping sequence.

Example 3 illustrates the application of these rules. Marked-content sequence S4 is a marked clipping sequence because it contains a clipping object (clipping path 2) and no visible graphics objects. Clipping path 2 is therefore considered part of sequence S4. Marked-content sequences S1, S2, and S3 are *not* marked clipping sequences, since they each include at least one visible graphics object. Thus, clipping paths 1 and 2 are not part of any of these three sequences.

EXAMPLE 3

```
/S1  BMC
    /S2  BMC
        /S3  BMC
            0  0  m
            100  100  l
            0  100  l  W  n                  % Clipping path 1

            0  0  m
            200  200  l
            0  100  l  f                     % Filled path
        EMC

        /S4  BMC
            0  0  m
            300  300  l
            0  100  l  W  n                  % Clipping path 2
        EMC
    EMC
    100  100  10  10  re  f                  % Filled path
EMC
```

In Example 4 marked-content sequence S1 is a marked clipping sequence because the only graphics object it contains is a clipping path. Thus, the empty marked-content sequence S3 and the marked-content point P1 are both part of sequence S2, and S2, S3, and P1 are all part of sequence S1.

EXAMPLE 4

```
/S1  BMC
    …Clipping path…
    /S2  BMC
        /S3  BMC
        EMC
        /P1  DP
    EMC
EMC
```

In Example 5 marked-content sequences S1 and S4 are marked clipping sequences because the only object they contain is a clipping path. Hence the clipping path is part of sequences S1 and S4; S3 is part of S2; and S2, S3, and S4 are all part of S1.

EXAMPLE 5

```
/S1  BMC
    /S2  BMC
        /S3  BMC
        EMC
    EMC

    /S4  BMC
        …Clipping path…
    EMC
EMC
```

14.7 Logical Structure

14.7.1 General

PDF's *logical structure* facilities *(PDF 1.3)* shall provide a mechanism for incorporating structural information about a document's content into a PDF file. Such information may include the organization of the document into chapters and sections or the identification of special elements such as figures, tables, and footnotes. The logical structure facilities shall be extensible, allowing conforming writers to choose what structural information to include and how to represent it, while enabling conforming readers to navigate a file without knowing the producer's structural conventions.

PDF logical structure shares basic features with standard document markup languages such as HTML, SGML, and XML. A document's logical structure shall be expressed as a hierarchy of *structure elements*, each represented by a dictionary object. Like their counterparts in other markup languages, PDF structure elements may have content and attributes. In PDF, rendered document content takes over the role occupied by text in HTML, SGML, and XML.

A PDF document's logical structure shall be stored separately from its visible content, with pointers from each to the other. This separation allows the ordering and nesting of logical elements to be entirely independent of the order and location of graphics objects on the document's pages.

The **Markings** entry in the document catalogue (see 7.7.2, "Document Catalog") shall specify a *mark information dictionary*, whose entries are shown in Table 321. It provides additional information relevant to specialized uses of structured PDF documents.

Table 321 – Entries in the mark information dictionary

Key	Type	Value
Marked	boolean	*(Optional)* A flag indicating whether the document conforms to Tagged PDF conventions (see 14.8, "Tagged PDF"). Default value: **false**. If **Suspects** is **true**, the document may not completely conform to Tagged PDF conventions.
UserProperties	boolean	*(Optional; PDF 1.6)* A flag indicating the presence of structure elements that contain user properties attributes (see 14.7.5.4, "User Properties"). Default value: **false**.
Suspects	boolean	*(Optional; PDF 1.6)* A flag indicating the presence of tag suspects (see 14.8.2.3, "Page Content Order"). Default value: **false**.

14.7.2 Structure Hierarchy

The logical structure of a document shall be described by a hierarchy of objects called the *structure hierarchy* or *structure tree*. At the root of the hierarchy shall be a dictionary object called the *structure tree root*, located by means of the **StructTreeRoot** entry in the document catalogue (see 7.7.2, “Document Catalog”). Table 322 shows the entries in the structure tree root dictionary. The **K** entry shall specify the immediate children of the structure tree root, which shall be *structure elements*.

Structure elements shall be represented by a dictionary, whose entries are shown in Table 323. The **K** entry shall specify the children of the structure element, which may be zero or more items of the following kinds:

- Other structure elements
- References to *content items*, which are either marked-content sequences (see 14.6, “Marked Content”) or complete PDF objects such as XObjects and annotations. These content items represent the graphical content, if any, associated with a structure element. Content items are discussed in detail in 14.7.4, “Structure Content.”

Table 322 – Entries in the structure tree root

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **StructTreeRoot** for a structure tree root.
K	dictionary or array	*(Optional)* The immediate child or children of the structure tree root in the structure hierarchy. The value may be either a dictionary representing a single structure element or an array of such dictionaries.
IDTree	name tree	*(Required if any structure elements have element identifiers)* A name tree that maps element identifiers (see Table 323) to the structure elements they denote.
ParentTree	number tree	*(Required if any structure element contains content items)* A number tree (see 7.9.7, “Number Trees”) used in finding the structure elements to which content items belong. Each integer key in the number tree shall correspond to a single page of the document or to an individual object (such as an annotation or an XObject) that is a content item in its own right. The integer key shall be the value of the **StructParent** or **StructParents** entry in that object (see 14.7.4.4, “Finding Structure Elements from Content Items”). The form of the associated value shall depend on the nature of the object: For an object that is a content item in its own right, the value shall be an indirect reference to the object’s parent element (the structure element that contains it as a content item). For a page object or content stream containing marked-content sequences that are content items, the value shall be an array of references to the parent elements of those marked-content sequences. See 14.7.4.4, “Finding Structure Elements from Content Items” for further discussion.
ParentTreeNextKey	integer	*(Optional)* An integer greater than any key in the parent tree, shall be used as a key for the next entry added to the tree.
RoleMap	dictionary	*(Optional)* A dictionary that shall map the names of structure types used in the document to their approximate equivalents in the set of standard structure types (see 14.8.4, “Standard Structure Types”).
ClassMap	dictionary	*(Optional)* A dictionary that shall map name objects designating attribute classes to the corresponding attribute objects or arrays of attribute objects (see 14.7.5.2, “Attribute Classes”).

Table 323 – Entries in a structure element dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **StructElem** for a structure element.
S	name	*(Required)* The structure type, a name object identifying the nature of the structure element and its role within the document, such as a chapter, paragraph, or footnote (see 14.7.3, "Structure Types"). Names of structure types shall conform to the guidelines described in Annex E.
P	dictionary	*(Required; shall be an indirect reference)* The structure element that is the immediate parent of this one in the structure hierarchy.
ID	byte string	*(Optional)* The element identifier, a byte string designating this structure element. The string shall be unique among all elements in the document's structure hierarchy. The **IDTree** entry in the structure tree root (see Table 322) defines the correspondence between element identifiers and the structure elements they denote.
Pg	dictionary	*(Optional; shall be an indirect reference)* A page object representing a page on which some or all of the content items designated by the **K** entry shall be rendered.
K	(various)	*(Optional)* The children of this structure element. The value of this entry may be one of the following objects or an array consisting of one or more of the following objects: • A structure element dictionary denoting another structure element • An integer marked-content identifier denoting a marked-content sequence • A marked-content reference dictionary denoting a marked-content sequence • An object reference dictionary denoting a PDF object Each of these objects other than the first (structure element dictionary) shall be considered to be a content item; see 14.7.4, "Structure Content" for further discussion of each of these forms of representation. If the value of **K** is a dictionary containing no **Type** entry, it shall be assumed to be a structure element dictionary.
A	(various)	*(Optional)* A single attribute object or array of attribute objects associated with this structure element. Each attribute object shall be either a dictionary or a stream. If the value of this entry is an array, each attribute object in the array may be followed by an integer representing its revision number (see 14.7.5, "Structure Attributes," and 14.7.5.3, "Attribute Revision Numbers").
C	name or array	*(Optional)* An attribute class name or array of class names associated with this structure element. If the value of this entry is an array, each class name in the array may be followed by an integer representing its revision number (see 14.7.5.2, "Attribute Classes," and 14.7.5.3, "Attribute Revision Numbers"). If both the **A** and **C** entries are present and a given attribute is specified by both, the one specified by the **A** entry shall take precedence.
R	integer	*(Optional)* The current revision number of this structure element (see 14.7.5.3, "Attribute Revision Numbers"). The value shall be a non-negative integer. Default value: 0.

Table 323 – Entries in a structure element dictionary (continued)

Key	Type	Value
T	text string	*(Optional)* The title of the structure element, a text string representing it in human-readable form. The title should characterize the specific structure element, such as Chapter 1, rather than merely a generic element type, such as Chapter.
Lang	text string	*(Optional; PDF 1.4)* A language identifier specifying the natural language for all text in the structure element except where overridden by language specifications for nested structure elements or marked content (see 14.9.2, “Natural Language Specification”). If this entry is absent, the language (if any) specified in the document catalogue applies.
Alt	text string	*(Optional)* An alternate description of the structure element and its children in human-readable form, which is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes (see 14.9.3, “Alternate Descriptions”).
E	text string	*(Optional; PDF 1.5)* The expanded form of an abbreviation.
ActualText	text string	*(Optional; PDF 1.4)* Text that is an exact replacement for the structure element and its children. This replacement text (which should apply to as small a piece of content as possible) is useful when extracting the document’s contents in support of accessibility to users with disabilities or for other purposes (see 14.9.4, “Replacement Text”).

14.7.3 Structure Types

Every structure element shall have a *structure type*, a name object that identifies the nature of the structure element and its role within the document (such as a chapter, paragraph, or footnote). To facilitate the interchange of content among conforming products, PDF defines a set of standard structure types; see 14.8.4, “Standard Structure Types.” Conforming products are not required to adopt them, however, and may use any names for their structure types.

Where names other than the standard ones are used, a *role map* may be provided in the structure tree root, mapping the structure types used in the document to their nearest equivalents in the standard set.

NOTE 1 A structure type named Section used in the document might be mapped to the standard type Sect. The equivalence need not be exact; the role map merely indicates an approximate analogy between types, allowing conforming products to share nonstandard structure elements in a reasonable way.

NOTE 2 The same structure type may occur as both a key and a value in the role map, and circular chains of association are explicitly permitted. Therefore, a single role map may define a bidirectional mapping. A conforming reader using the role map should follow the chain of associations until it either finds a structure type it recognizes or returns to one it has already encountered.

NOTE 3 In PDF versions earlier than 1.5, standard element types were never remapped. Beginning with PDF 1.5, an element name shall always be mapped to its corresponding name in the role map, if there is one, even if the original name is one of the standard types. This shall be done to allow the element, for example, to represent a tag with the same name as a standard role, even though its use differs from the standard role.

14.7.4 Structure Content

14.7.4.1 General

Any structure element may have associated graphical content, consisting of one or more *content items*. Content items shall be graphical objects that exist in the document independently of the structure tree but are associated with structure elements as described in the following sub-clauses. Content items are of two kinds:

- Marked-content sequences within content streams (see 14.7.4.2, "Marked-Content Sequences as Content Items")
- Complete PDF objects such as annotations and XObjects (see 14.7.4.3, "PDF Objects as Content Items")

The **K** entry in a structure element dictionary (see Table 323) shall specify the children of the structure element, which may include any number of content items, as well as child structure elements that may in turn have content items of their own.

Content items shall be leaf nodes of the structure tree; that is, they may not have other content items nested within them for purposes of logical structure. The hierarchical relationship among structure elements shall be represented entirely by the **K** entries of the structure element dictionaries, not by nesting of the associated content items. Therefore, the following restrictions shall apply:

- A marked-content sequence delimiting a structure content item may not have another marked-content sequence for a content item nested within it though non-structural marked content shall be allowed.
- A structure content item shall not invoke (with the **Do** operator) an XObject that is itself a structure content item.

14.7.4.2 Marked-Content Sequences as Content Items

A sequence of graphics operators in a content stream may be specified as a content item of a structure element in the following way:

- The operators shall be bracketed as a marked-content sequence between **BDC** and **EMC** operators (see 14.6, "Marked Content"). Although the tag associated with a marked-content sequence is not directly related to the document's logical structure, it should be the same as the structure type of the associated structure element.
- The marked-content sequence shall have a property list (see 14.6.2, "Property Lists") containing an **MCID** entry, which i shall be an integer *marked-content identifier* that uniquely identifies the marked-content sequence within its content stream, as shown in the following example:

EXAMPLE 1
```
2  0  obj                               % Page object
    <<  /Type  /Page
         /Contents  3 0 R               % Content stream
         …
    >>
endobj

3  0  obj                               % Page's content stream
    <<  /Length  …  >>
stream
    …
    /P  << /MCID  0 >>                  % Start of marked-content sequence
        BDC
            …
            (Here is some text)  Tj
            …
        EMC                             % End of marked-content sequence
    …
```

```
endstream
endobj
```

NOTE This example and the following examples omit required ***StructParents*** entries in the objects used as content items (see 14.7.4.4, “Finding Structure Elements from Content Items”).

A structure element dictionary may include one or more marked-content sequences as content items by referring to them in its **K** entry (see Table 323). This reference may have two forms:

- A dictionary object called a *marked-content reference.* Table 324 shows the contents of this type of dictionary, which shall specify the marked-content identifier, as well other information identifying the stream in which the sequence is contained. Example 2 illustrates the use of a marked-content reference to the marked-content sequence shown in Example 3.
- An integer that specifies the marked-content identifier. This may be done in the common case where the marked-content sequence is contained in the content stream of the page that is specified in the **Pg** entry of the structure element dictionary. Example 3 shows a structure element that has three children: a marked-content sequence specified by a marked-content identifier, as well as two other structure elements.

EXAMPLE 2

```
1  0  obj                                 % Structure element
   <<  /Type  /StructElem
       /S  /P                             % Structure type
       /P  …                              % Parent in structure hierarchy
      /K  << /Type  /MCR
             /Pg  2 0 R                   % Page containing marked-content sequence
             /MCID  0                     % Marked-content identifier
          >>
   >>
endobj
```

Table 324 – Entries in a marked-content reference dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **MCR** for a marked-content reference.
Pg	dictionary	*(Optional; shall be an indirect reference)* The page object representing the page on which the graphics objects in the marked-content sequence shall be rendered. This entry overrides any **Pg** entry in the structure element containing the marked-content reference; it shall be required if the structure element has no such entry.
Stm	stream	*(Optional; shall be an indirect reference)* The content stream containing the marked-content sequence. This entry should be present only if the marked-content sequence resides in a content stream other than the content stream for the page (see 8.10, “Form XObjects” and 12.5.5, “Appearance Streams”). If this entry is absent, the marked-content sequence shall be contained in the content stream of the page identified by **Pg** (either in the marked-content reference dictionary or in the parent structure element).
StmOwn	(any)	*(Optional; shall be an indirect reference)* The PDF object owning the stream identified by **Stems** annotation to which an appearance stream belongs.
MCID	integer	*(Required)* The marked-content identifier of the marked-content sequence within its content stream.

EXAMPLE 3

```
1  0  obj                                 % Containing structure element
   <<  /Type  /StructElem
       /S  /MixedContainer                % Structure type
       /P  …                              % Parent in structure hierarchy
```

```
          /Pg  2 0 R                      % Page containing marked-content sequence
          /K  [  4 0 R                    % Three children: a structure element
                 0                        %        a marked-content identifier
                 5 0 R                    %        another structure element
              ]
      >>
  endobj

  2  0  obj                             % Page object
      << /Type  /Page
          /Contents  3 0 R                % Content stream
          …
      >>
  endobj

  3  0  obj                             % Page's content stream
      << /Length  …  >>
  stream
      …
      /P  << /MCID  0 >>                  % Start of marked-content sequence
          BDC
              (Here is some text)  Tj
              …
          EMC                             % End of marked-content sequence
      …
  endstream
  endobj
```

Content streams other than page contents may also contain marked content sequences that are content items of structure elements. The content of form XObjects may be incorporated into structure elements in one of the following ways:

- A **Do** operator that paints a form XObject may be part of a marked-content sequence that shall be associated with a structure element (see Example 4). In this case, the entire form XObject shall be considered to be part of the structure element's content, as if it were inserted into the marked-content sequence at the point of the **Do** operator. The form XObject shall not in turn contain any marked-content sequences associated with this or other structure elements.

- The content stream of a form XObject may contain one or more marked-content sequences that shall be associated with structure elements (see Example 5). The form XObject may have arbitrary substructure, containing any number of marked-content sequences associated with logical structure elements. However, any **Do** operator that paints the form XObject should not be part of a logical structure content item.

A form XObject that is painted with multiple invocations of the **Do** operator may be incorporated into the document's logical structure only by the first method, with each invocation of **Do** individually associated with a structure element.

EXAMPLE 4

```
1  0  obj                                      % Structure element
    << /Type  /StructElem
         /S  /P                                 % Structure type
         /P  …                                  % Parent in structure hierarchy
         /Pg  2 0 R                             % Page containing marked-content sequence
         /K  0                                  % Marked-content identifier
    >>
endobj

2  0  obj                                      % Page object
    << /Type  /Page
        /Resources    <<  /XObject  << /Fm4  4 0 R >>    % Resource dictionary
                      >>                                 %     containing form XObject
        /Contents  3 0 R                                 % Content stream
        …
    >>
```

```
endobj

3  0  obj                          % Page's content stream
    <<  /Length  …  >>
stream
    …
    /P  << /MCID  0 >>                 % Start of marked-content sequence
        BDC
            /Fm4  Do                   % Paint form XObject
        EMC                            % End of marked-content sequence
    …
endstream
endobj

4  0  obj                          % Form XObject
    <<  /Type  /XObject
        /Subtype  /Form
        /Length  …
    >>
stream
    …
    (Here is some text)  Tj
    …
endstream
endobj
```

EXAMPLE 5

```
1  0  obj                          % Structure element
    <<  /Type  /StructElem
        /S  /P                         % Structure type
        /P  …                          % Parent in structure hierarchy
        /K  <<  /Type  /MCR
                /Pg  2 0 R             % Page containing marked-content sequence
                /Stm  4 0 R            % Stream containing marked-content sequence
                /MCID  0               % Marked-content identifier
            >>
    >>
endobj

2  0  obj                                               % Page object
    <<  /Type  /Page
        /Resources    <<  /XObject  << /Fm4  4 0 R >>   % Resource dictionary
                      >>                                %     containing form XObject
        /Contents  3 0 R                                % Content stream
        …
    >>
endobj

3 0 obj                            % Page's content stream
    <<  /Length  …  >>
stream
    …
    /Fm4  Do                       % Paint form XObject
    …
endstream
endobj

4  0  obj                          % Form XObject
    <<  /Type  /XObject
        /Subtype  /Form
        /Length  …
    >>
stream
    …
    /P  << /MCID  0 >>             % Start of marked-content sequence
        BDC
```

```
            …
            (Here is some text)  Tj
            …
         EMC                              % End of marked-content sequence
         …
      endstream
      endobj
```

14.7.4.3 PDF Objects as Content Items

When a structure element's content includes an entire PDF object, such as an XObject or an annotation, that is associated with a page but not directly included in the page's content stream, the object shall be identified in the structure element's **K** entry by an *object reference dictionary* (see Table 325).

NOTE 1 This form of reference is used only for entire objects. If the referenced content forms only part of the object's content stream, it is instead handled as a marked-content sequence, as described in the preceding sub-clause.

Table 325 – Entries in an object reference dictionary

Key	Type	Value
Type	name	*(Required)* The type of PDF object that this dictionary describes; shall be **OBJR** for an object reference.
Pg	dictionary	*(Optional; shall be an indirect reference*) The page object of the page on which the object shall be rendered. This entry overrides any **Pg** entry in the structure element containing the object reference; it shall be used if the structure element has no such entry.
Obj	(any)	*(Required; shall be an indirect reference*) The referenced object.

NOTE 2 If the referenced object is rendered on multiple pages, each rendering requires a separate object reference. However, if it is rendered multiple times on the same page, just a single object reference suffices to identify all of them. (If it is important to distinguish between multiple renditions of the same XObject on the same page, they should be accessed by means of marked-content sequences enclosing particular invocations of the **Do** operator rather than through object references.)

14.7.4.4 Finding Structure Elements from Content Items

Because a stream may not contain object references, there is no way for content items that are marked-content sequences to refer directly back to their parent structure elements (the ones to which they belong as content items). Instead, a different mechanism, the *structural parent tree*, shall be provided for this purpose. For consistency, content items that are entire PDF objects, such as XObjects, also shall use the parent tree to refer to their parent structure elements.

The parent tree is a number tree (see 7.9.7, "Number Trees"), accessed from the **ParentTree** entry in a document's structure tree root (Table 322). The tree shall contain an entry for each object that is a content item of at least one structure element and for each content stream containing at least one marked-content sequence that is a content item. The key for each entry shall be an integer given as the value of the **StructParent** or **StructParents** entry in the object (see Table 326). The values of these entries shall be as follows:

- For an object identified as a content item by means of an object reference (see 14.7.4.3, "PDF Objects as Content Items"), the value shall be an indirect reference to the parent structure element.

- For a content stream containing marked-content sequences that are content items, the value shall be an array of indirect references to the sequences' parent structure elements. The array element corresponding to each sequence shall be found by using the sequence's marked-content identifier as a zero-based index into the array.

NOTE Because marked-content identifiers serve as indices into an array in the structural parent tree, their assigned values should be as small as possible to conserve space in the array.

The **ParentTreeNextKey** entry in the structure tree root shall hold an integer value greater than any that is currently in use as a key in the structural parent tree. Whenever a new entry is added to the parent tree, the current value of **ParentTreeNextKey** shall be used as its key. The value shall be then incremented to prepare for the next new entry to be added.

To locate the relevant parent tree entry, each object or content stream that is represented in the tree shall contain a special dictionary entry, **StructParent** or **StructParents** (see Table 326). Depending on the type of content item, this entry may appear in the page object of a page containing marked-content sequences, in the stream dictionary of a form or image XObject, in an annotation dictionary, or in any other type of object dictionary that is included as a content item in a structure element. Its value shall be the integer key under which the entry corresponding to the object shall be found in the structural parent tree.

Table 326 – Additional dictionary entries for structure element access

Key	Type	Value
StructParent	integer	*(Required for all objects that are structural content items; PDF 1.3)* The integer key of this object's entry in the structural parent tree.
StructParents	integer	*(Required for all content streams containing marked-content sequences that are structural content items; PDF 1.3)* The integer key of this object's entry in the structural parent tree. At most one of these two entries shall be present in a given object. An object may be either a content item in its entirety or a container for marked-content sequences that are content items, but not both.

For a content item identified by an object reference, the parent structure element may be found by using the value of the **StructParent** entry in the item's object dictionary as a retrieval key in the structural parent tree (found in the **ParentTree** entry of the structure tree root). The corresponding value in the parent tree shall be a reference to the parent structure element (see Example 1).

EXAMPLE 1

```
1  0  obj                              % Parent structure element
   <<  /Type  /StructElem
        …
        /K  <<  /Type  /OBJR            % Object reference
                 /Pg  2 0 R             % Page containing form XObject
                 /Obj  4 0 R            % Reference to form XObject
            >>
   >>
endobj

2  0  obj                                         % Page object
   <<  /Type  /Page
       /Resources    <<  /XObject  << /Fm4  4 0 R >>     % Resource dictionary
                     >>                                  %     containing form XObject
        /Contents  3 0 R                                 % Content stream
        …
   >>
endobj

3 0 obj                                % Page's content stream
   <<  /Length  …  >>
stream
   …
   /Fm4  Do                             % Paint form XObject
   …
endstream
endobj
```

```
4  0  obj                                  % Form XObject
    <<  /Type  /XObject
        /Subtype  /Form
        /Length  …
        /StructParent  6                       % Parent tree key
    >>
stream
    …
endstream
endobj

100  0  obj                                % Parent tree (accessed from structure tree root)
    <<  /Nums  [  0  101 0 R
                    1  102 0 R
                    …
                    6  1 0 R                   % Entry for page object 2; points back
                    …                          %     to parent structure element
                ]
    >>
endobj
```

For a content item that is a marked-content sequence, the retrieval method is similar but slightly more complicated. Because a marked-content sequence is not an object in its own right, its parent tree key shall be found in the **StructParents** entry of the page object or other content stream in which the sequence resides. The value retrieved from the parent tree shall not be a reference to the parent structure element itself but to an array of such references—one for each marked-content sequence contained within that content stream. The parent structure element for the given sequence shall be found by using the sequence’s marked-content identifier as an index into this array (see Example 2).

EXAMPLE 2

```
1  0  obj                                  % Parent structure element
    <<  /Type  /StructElem
        …
        /Pg  2 0 R                             % Page containing marked-content sequence
        /K  0                                  % Marked-content identifier
    >>
endobj

2  0  obj                                  % Page object
    <<  /Type  /Page
        /Contents  3 0 R                       % Content stream
        /StructParents  6                      % Parent tree key
        …
    >>
endobj

3  0  obj                                  % Page's content stream
    <<  /Length  …  >>
stream
    …
    /P  << /MCID  0 >>                         % Start of marked-content sequence
        BDC
            (Here is some text)  TJ
            …
        EMC                                    % End of marked-content sequence
    …
endstream
endobj

100  0  obj                                % Parent tree (accessed from structure tree root)
    <<  /Nums  [  0  101 0 R
                    1  102 0 R
                    …
                    6  [1 0 R]                  % Entry for page object 2; array element at index 0
                    …                           %     points back to parent structure element
                  ]
            >>
        endobj
```

14.7.5 Structure Attributes

14.7.5.1 General

A conforming product that processes logical structure may attach additional information, called *attributes*, to any structure element. The attribute information shall be held in one or more *attribute objects* associated with the structure element. An attribute object shall be a dictionary or stream that includes an **O** entry (see Table 327) identifying the conforming product that owns the attribute information. Other entries shall represent the attributes: the keys shall be attribute names, and values shall be the corresponding attribute values. To facilitate the interchange of content among conforming products, PDF defines a set of standard structure attributes identified by specific standard owners; see 14.8.5, "Standard Structure Attributes." In addition, *(PDF 1.6)* attributes may be used to represent user properties (see 14.7.5.4, "User Properties").

Table 327 – Entry common to all attribute object dictionaries

Key	Type	Value
O	name	*(Required)* The name of the conforming product owning the attribute data. The name shall conform to the guidelines described in Annex E.

Any conforming product may attach attributes to any structure element, even one created by another conforming product. Multiple conforming products may attach attributes to the same structure element. The **A** entry in the structure element dictionary (see Table 323) shall hold either a single attribute object or an array of such objects, together with *revision numbers* for coordinating attributes created by different conforming products (see 14.7.5.3, "Attribute Revision Numbers"). A conforming product creating or destroying the second attribute object for a structure element shall be responsible for converting the value of the **A** entry from a single object to an array or vice versa, as well as for maintaining the integrity of the revision numbers. No inherent order shall be defined for the attribute objects in an **A** array, but new objects should be added at the end of the array so that the first array element is the one belonging to the conforming product that originally created the structure element.

14.7.5.2 Attribute Classes

If many structure elements share the same set of attribute values, they may be defined as an *attribute class* sharing the identical attribute object. Structure elements shall refer to the class by name. The association between class names and attribute objects shall be defined by a dictionary called the *class map*, that shall be kept in the **ClassMap** entry of the structure tree root (see Table 322). Each key in the class map shall be a name object denoting the name of a class. The corresponding value shall be an attribute object or an array of such objects.

NOTE PDF attribute classes are unrelated to the concept of a class in object-oriented programming languages such as Java and C++. Attribute classes are strictly a mechanism for storing attribute information in a more compact form; they have no inheritance properties like those of true object-oriented classes.

The **C** entry in a structure element dictionary (see Table 323) shall contain a class name or an array of class names (typically accompanied by revision numbers as well; see 14.7.5.3, "Attribute Revision Numbers"). For each class named in the **C** entry, the corresponding attribute object or objects shall be considered to be attached to the given structure element, along with those identified in the element's **A** entry. If both the **A** and **C** entries are present and a given attribute is specified by both, the one specified by the **A** entry shall take precedence.

14.7.5.3 Attribute Revision Numbers

When a conforming product modifies a structure element or its contents, the change may affect the validity of attribute information attached to that structure element by other conforming products. A system of *revision numbers* shall allow conforming products to detect such changes and update their own attribute information accordingly, as described in this sub-clause.

A structure element shall have a revision number, that shall be stored in the **R** entry in the structure element dictionary (see Table 323) or default to 0 if no **R** entry is present. Initially, the revision number shall be 0. When a conforming product modifies the structure element or any of its content items, it may signal the change by incrementing the revision number.

NOTE 1 The revision number is unrelated to the generation number associated with an indirect object (see 7.3.10, “Indirect Objects”).

NOTE 2 If their is no R entry and the revision number is to be incremented from the default value of 0 to 1, an R entry must be created in the structure element dictionary in order to record the 1.

Each attribute object attached to a structure element shall have an associated revision number. The revision number shall be stored in the array that associates the attribute object with the structure element or if not stored in the array that associates the attribute object with the structure element shall default to 0.

- Each attribute object in a structure element’s **A** array shall be represented by a single or a pair of array elements, the first or only element shall contain the attribute object itself and the second (when present) shall contain the integer revision number associated with it in this structure element.
- The structure element’s **C** array shall contain a single or a pair of elements for each attribute class, the first or only shall contain the class name and the second (when present) shall contain the associated revision number.

The revision numbers are optional in both the **A** and **C** arrays. An attribute object or class name that is not followed by an integer array element shall have a revision number of 0 and is represented by a single entry in the array.

NOTE 3 The revision number is not stored directly in the attribute object because a single attribute object may be associated with more than one structure element (whose revision numbers may differ). Since an attribute object reference is distinct from an integer, that distinction is used to determine whether the attribute object is represented in the array by a single or a pair of entries.

NOTE 4 When an attribute object is created or modified, its revision number is set to the current value of the structure element’s **R** entry. By comparing the attribute object’s revision number with that of the structure element, an application can determine whether the contents of the attribute object are still current or whether they have been outdated by more recent changes in the underlying structure element.

Changes in an attribute object shall not change the revision number of the associated structure element, which shall change only when the structure element itself or any of its content items is modified.

Occasionally, a conforming product may make extensive changes to a structure element that are likely to invalidate all previous attribute information associated with it. In this case, instead of incrementing the structure element’s revision number, the conforming product may choose to delete all unknown attribute objects from its **A** and **C** arrays. These two actions shall be mutually exclusive: the conforming product should *either* increment the structure element’s revision number *or* remove its attribute objects, but not both.

NOTE 5 Any conforming product creating attribute objects needs to be prepared for the possibility that they can be deleted at any time by another conforming product.

14.7.5.4 User Properties

Most structure attributes (see 14.8.5, “Standard Structure Attributes”) specify information that is reflected in the element’s appearance; for example, **BackgroundColor** or **BorderStyle**.

Some conforming writers, such as CAD applications, may use objects that have a standardized appearance, each of which contains non-graphical information that distinguishes the objects from one another. For example, several transistors might have the same appearance but different attributes such as type and part number.

User properties (PDF 1.6) may be used to contain such information. Any graphical object that corresponds to a structure element may have associated user properties, specified by means of an attribute object dictionary that shall have a value of **UserProperties** for the **O** entry (see Table 328).

Table 328 – Additional entries in an attribute object dictionary for user properties

Key	Type	Value
O	name	*(Required)* The attribute owner. Shall be **UserProperties**.
P	array	*(Required)* An array of dictionaries, each of which represents a user property (see Table 329).

The **P** entry shall be an array specifying the user properties. Each element in the array shall be a *user property dictionary* representing an individual property (see Table 329). The order of the array elements shall specify attributes in order of importance.

Table 329 – Entries in a user property dictionary

Key	Type	Value
N	text	*(Required)* The name of the user property.
V	any	*(Required)* The value of the user property. While the value of this entry shall be any type of PDF object, conforming writers should use only text string, number, and boolean values. Conforming readers should display text, number and boolean values to users but need not display values of other types; however, they should not treat other values as errors.
F	text string	*(Optional)* A formatted representation of the value of **V**, that shall be used for special formatting; for example “($123.45)” for the number -123.45. If this entry is absent, conforming readers should use a default format.
H	boolean	*(Optional)* If **true**, the attribute shall be hidden; that is, it shall not be shown in any user interface element that presents the attributes of an object. Default value: **false**.

PDF documents that contain user properties shall provide a **UserProperties** entry with a value of **true** in the document’s mark information dictionary (see Table 321). This entry allows conforming readers to quickly determine whether it is necessary to search the structure tree for elements containing user properties.

EXAMPLE The following example shows a structure element containing user properties called Part Name, Part Number, Supplier, and Price.

```
100 0 obj
   << /Type /StructElem
      /S /Figure                                          % Structure type
      /P 50 0 R                                           % Parent in structure tree
      /A << /O /UserProperties                            % Attribute object
            /P [                                          % Array of user properties
               << /N (Part Name) /V (Framostat) >>
               << /N (Part Number) /V 11603 >>
               << /N (Supplier) /V (Just Framostats) /H true >>      % Hidden attribute
               << /N (Price) /V -37.99 /F ($37.99) >>                % Formatted value
               ]
         >>
   >>
endobj
```

14.7.6 Example of Logical Structure

The next Example shows portions of a PDF file with a simple document structure. The structure tree root (object 300) contains elements with structure types **Chap** (object 301) and **Para** (object 304). The **Chap** element, titled Chapter 1, contains elements with types **Head1** (object 302) and **Para** (object 303).

These elements are mapped to the standard structure types specified in Tagged PDF (see 14.8.4, “Standard Structure Types”) by means of the role map specified in the structure tree root. Objects 302 through 304 have attached attributes (see 14.7.5, “Structure Attributes,” and 14.8.5, “Standard Structure Attributes”).

The example also illustrates the structure of a parent tree (object 400) that maps content items back to their parent structure elements and an ID tree (object 403) that maps element identifiers to the structure elements they denote.

EXAMPLE

```
1  0  obj                                   % Document catalog
    <<  /Type  /Catalog
        /Pages  100 0 R                     % Page tree
        /StructTreeRoot  300 0 R            % Structure tree root
    >>
endobj

100  0  obj                               % Page tree
    <<  /Type  /Pages
        /Kids   [  101 1 R                  % First page object
                102 0 R                     % Second page object
                ]
        /Count  2                           % Page count
    >>
endobj

101  1  obj                               % First page object
    <<  /Type  /Page
        /Parent  100 0 R                    % Parent is the page tree
        /Resources    <<  /Font    <<  /F1  6 0 R    % Font resources
                                     /F12  7 0 R
                                 >>
                        /ProcSet  [/PDF  /Text]     % Procedure sets
                    >>
        /MediaBox  [0  0  612  792]         % Media box
        /Contents  201 0 R                  % Content stream
        /StructParents  0                   % Parent tree key
    >>
endobj

201  0  obj                               % Content stream for first page
    <<  /Length  …  >>
stream
    1  1  1  rg
    0  0  612  792  re  f
    BT                                      % Start of text object

        /Head1  << /MCID  0 >>              % Start of marked-content sequence 0
            BDC
                0  0  0  rg
                /F1  1  Tf
                30  0  0  30  18  732  Tm
                (This is a first level heading. Hello world: )  Tj
                1.1333  TL
                T*
                (goodbye universe.)  Tj
            EMC                             % End of marked-content sequence 0

        /Para  << /MCID  1 >>               % Start of marked-content sequence 1
        BDC
            /F12  1  Tf
            14  0  0  14  18  660.8  Tm
            (This is the first paragraph, which spans pages. It has four fairly short and \
concise sentences. This is the next to last )  Tj
        EMC                                     % End of marked-content sequence 1

    ET                                          % End of text object
endstream
endobj

102  0  obj                                     % Second page object
    <<  /Type  /Page
         /Parent  100 0 R                           % Parent is the page tree
         /Resources   <<  /Font  <<      /F1  6 0 R      % Font resources
                                         /F12  7 0 R
                                     >>
                         /ProcSet  [/PDF  /Text]     % Procedure sets
                      >>
        /MediaBox  [0  0  612  792]                 % Media box
         /Contents  202 0 R                         % Content stream
        /StructParents  1                           % Parent tree key
    >>
endobj

202  0  obj                                     % Content stream for second page
    <<  /Length  …  >>
stream
    1  1  1  rg
    0  0  612  792  re  f
    BT                                          % Start of text object

        /Para  << /MCID  0 >>                       % Start of marked-content sequence 0
            BDC
                0  0  0  rg
                /F12  1  Tf
                14  0  0  14  18  732  Tm
                (sentence. This is the very last sentence of the first paragraph.)  Tj
            EMC                                     % End of marked-content sequence 0

    /Para  << /MCID  1 >>                       % Start of marked-content sequence 1
        BDC
            /F12  1  Tf
            14  0  0  14  18  570.8  Tm
            (This is the second paragraph. It has four fairly short and concise sentences. \ This is the next
to last )  Tj
        EMC                                     % End of marked-content sequence 1

    /Para  << /MCID  2 >>                       % Start of marked-content sequence 2
        BDC
            1.1429  TL
            T*
            (sentence. This is the very last sentence of the second paragraph.)  Tj
        EMC                                     % End of marked-content sequence 2

    ET                                              % End of text object
endstream
endobj

300  0  obj                                     % Structure tree root
    <<  /Type  /StructTreeRoot
         /K  [  301 0 R                             % Two children: a chapter
            304 0 R                                 % and a paragraph
            ]
        /RoleMap   <<  /Chap  /Sect                  % Mapping to standard structure types
                        /Head1  /H
                        /Para  /P
                     >>
        /ClassMap     <<  /Normal 305 0 R >>        % Class map containing one attribute class

        /ParentTree  400 0 R                         % Number tree for parent elements
        /ParentTreeNextKey  2                        % Next key to use in parent tree
        /IDTree  403 0 R                             % Name tree for element identifiers
    >>
endobj

301  0  obj                                       % Structure element for a chapter
    <<  /Type  /StructElem
        /S  /Chap
        /ID  (Chap1)                                 % Element identifier
        /T  (Chapter 1)                              % Human-readable title
        /P  300 0 R                                  % Parent is the structure tree root
        /K  [  302 0 R                               % Two children: a section head
             303 0 R                                 %     and a paragraph
             ]
    >>
endobj

302  0  obj                                       % Structure element for a section head
    <<  /Type  /StructElem
        /S  /Head1
        /ID  (Sec1.1)                                % Element identifier
        /T  (Section 1.1)                            % Human-readable title
        /P  301 0 R                                  % Parent is the chapter
        /Pg  101 1 R                                 % Page containing content items
        /A  << /O  /Layout                           % Attribute owned by Layout
                /SpaceAfter 25
                /SpaceBefore 0
                /TextIndent 12.5
             >>
        /K  0                                        % Marked-content sequence 0
    >>
endobj

303  0  obj                                       % Structure element for a paragraph
    <<  /Type  /StructElem
        /S  /Para
        /ID  (Para1)                                 % Element identifier
        /P  301 0 R                                  % Parent is the chapter
        /Pg  101 1 R                                 % Page containing first content item
        /C  /Normal                                  % Class containing this element’s attributes
        /K  [  1                                     % Marked-content sequence 1
                << /Type  /MCR                       % Marked-content reference to 2nd item
                     /Pg  102 0 R                    % Page containing second item
                     /MCID  0                        % Marked-content sequence 0
                >>
             ]
    >>
endobj

304  0  obj                                       % Structure element for another paragraph
    <<  /Type  /StructElem
        /S  /Para
        /ID  (Para2)                                 % Element identifier
        /P  300 0 R                                  % Parent is the structure tree root
        /Pg  102 0 R                                 % Page containing content items
        /C  /Normal                                  % Class containing this element’s attributes
        /A  << /O  /Layout
                /TextAlign  /Justify                 % Overrides attribute provided by classmap
```

```
            >>
        /K  [1  2]                          % Marked-content sequences 1 and 2
    >>
endobj

305  0  obj                             % Attribute class
    <<  /O  /Layout                         % Owned by Layout
        /EndIndent 0
        /StartIndent 0
        /WritingMode /LrTb
        /TextAlign /Start
    >>
endobj

400  0  obj                             % Parent tree
    <<  /Nums  [   0  401 0 R               % Parent elements for first page
                   1  402 0 R               % Parent elements for second page
               ]
    >>
endobj

401  0  obj                             % Array of parent elements for first page
    [  302 0 R                              % Parent of marked-content sequence 0
     303 0 R                                % Parent of marked-content sequence 1
    ]
endobj

402  0  obj                             % Array of parent elements for second page
    [  303 0 R                              % Parent of marked-content sequence 0
    304 0 R                                 % Parent of marked-content sequence 1
    304 0 R                                 % Parent of marked-content sequence 2
    ]
endobj

403  0  obj                             % ID tree root node
    <<  /Kids  [404 0 R]  >>                % Reference to leaf node
endobj

404  0  obj                             % ID tree leaf node
    <<  /Limits     [  (Chap1)  (Sec1.3)  ]     % Least and greatest keys in tree
         /Names     [  (Chap1)  301 0 R         % Mapping from element identifiers
                    (Sec1.1)  302 0 R           %     to structure elements
                     (Sec1.2)  303 0 R
                     (Sec1.3)  304 0 R
                    ]
    >>
endobj
```

14.8 Tagged PDF

14.8.1 General

Tagged PDF (PDF 1.4) is a stylized use of PDF that builds on the logical structure framework described in 14.7, “Logical Structure.” It defines a set of standard structure types and attributes that allow page content (text, graphics, and images) to be extracted and reused for other purposes. A tagged PDF document is one that conforms to the rules described in this sub-clause. A conforming writer is not required to produce tagged PDF documents; however, if it does, it shall conform to these rules.

NOTE 1 It is intended for use by tools that perform the following types of operations:

- Simple extraction of text and graphics for pasting into other applications

- Automatic reflow of text and associated graphics to fit a page of a different size than was assumed for the original layout
- Processing text for such purposes as searching, indexing, and spell-checking
- Conversion to other common file formats (such as HTML, XML, and RTF) with document structure and basic styling information preserved
- Making content accessible to users with visual impairments (see 14.9, "Accessibility Support")

A tagged PDF document shall conform to the following rules:

- *Page content* (14.8.2, "Tagged PDF and Page Content"). Tagged PDF defines a set of rules for representing text in the page content so that characters, words, and text order can be determined reliably. All text shall be represented in a form that can be converted to Unicode. Word breaks shall be represented explicitly. Actual content shall be distinguished from artifacts of layout and pagination. Content shall be given in an order related to its appearance on the page, as determined by the conforming writer.

- A *basic layout model* (14.8.3, "Basic Layout Model"). A set of rules for describing the arrangement of structure elements on the page.

- *Structure types* (14.8.4, "Standard Structure Types"). A set of standard structure types define the meaning of structure elements, such as paragraphs, headings, articles, and tables.

- *Structure attributes* (14.8.5, "Standard Structure Attributes"). Standard structure attributes preserve styling information used by the conforming writer in laying out content on the page.

A Tagged PDF document shall also contain a mark information dictionary (see Table 321) with a value of **true** for the **Marked** entry.

NOTE 2 The types and attributes defined for Tagged PDF are intended to provide a set of standard fallback roles and minimum guaranteed attributes to enable conforming readers to perform operations such as those mentioned previously. Conforming writers are free to define additional structure types as long as they also provide a role mapping to the nearest equivalent standard types, as described in 14.7.3, "Structure Types." Likewise, conforming writers can define additional structure attributes using any of the available extension mechanisms.

14.8.2 Tagged PDF and Page Content

14.8.2.1 General

Like all PDF documents, a Tagged PDF document consists of a sequence of self-contained pages, each of which shall be described by one or more page content streams (including any subsidiary streams such as form XObjects and annotation appearances). Tagged PDF defines some further rules for organizing and marking content streams so that additional information can be derived from them:

- Distinguishing between the author's original content and artifacts of the layout process (see 14.8.2.2, "Real Content and Artifacts").

- Specifying a content order to guide the layout process if the conforming reader reflows the page content (see 14.8.2.3, "Page Content Order").

- Representing text in a form from which a Unicode representation and information about font characteristics can be unambiguously derived (see 14.8.2.4, "Extraction of Character Properties").

- Representing word breaks unambiguously (see 14.8.2.5, "Identifying Word Breaks").

- Marking text with information for making it accessible to users with visual impairments (see 14.9, "Accessibility Support").

14.8.2.2 Real Content and Artifacts

14.8.2.2.1 General

The graphics objects in a document can be divided into two classes:

- The *real content* of a document comprises objects representing material originally introduced by the document's author.
- *Artifacts* are graphics objects that are not part of the author's original content but rather are generated by the conforming writer in the course of pagination, layout, or other strictly mechanical processes.

NOTE Artifacts may also be used to describe areas of the document where the author uses a graphical background, with the goal of enhancing the visual experience. In such a case, the background is not required for understanding the content.

The document's logical structure encompasses all graphics objects making up the real content and describes how those objects relate to one another. It does not include graphics objects that are mere artifacts of the layout and production process.

A document's real content includes not only the page content stream and subsidiary form XObjects but also associated annotations that meet all of the following conditions:

- The annotation has an appearance stream (see 12.5.5, "Appearance Streams") containing a normal (**N**) appearance.
- The annotation's Hidden flag (see 12.5.3, "Annotation Flags") is not set.
- The annotation is included in the document's logical structure (see 14.7, "Logical Structure").

14.8.2.2.2 Specification of Artifacts

An artifact shall be explicitly distinguished from real content by enclosing it in a marked-content sequence with the tag Artifact:

```
/Artifact                              /Artifact  propertyList
   BMC                                    BDC
    …              or                      …
   EMC                                    EMC
```

The first form shall be used to identify a generic artifact; the second shall be used for those that have an associated property list. Table 330 shows the properties that can be included in such a property list.

NOTE 1 To aid in text reflow, artifacts should be defined with property lists whenever possible. Artifacts lacking a specified bounding box are likely to be discarded during reflow.

Table 330 – Property list entries for artifacts

Key	Type	Value
Type	name	*(Optional)* The type of artifact that this property list describes; if present, shall be one of the names **Pagination**, **Layout**, **Page**, or *(PDF 1.7)* **Background**.
BBox	rectangle	*(Optional; required for background artifacts)* An array of four numbers in default user space units giving the coordinates of the left, bottom, right, and top edges, respectively, of the artifact's bounding box (the rectangle that completely encloses its visible extent).

Table 330 – Property list entries for artifacts (continued)

Key	Type	Value
Attached	array	*(Optional; pagination and full-page background artifacts only)* An array of name objects containing one to four of the names Top, Bottom, Left, and Right, specifying the edges of the page, if any, to which the artifact is logically attached. Page edges shall be defined by the page's crop box (see 14.11.2, "Page Boundaries"). The ordering of names within the array is immaterial. Including both Left and Right or both Top and Bottom indicates a full-width or full-height artifact, respectively. Use of this entry for background artifacts shall be limited to full-page artifacts. Background artifacts that are not full-page take their dimensions from their parent structural element.
Subtype	name	*(Optional; PDF 1.7)* The subtype of the artifact. This entry should appear only when the **Type** entry has a value of Pagination. Standard values are Header, Footer, and Watermark. Additional values may be specified for this entry, provided they comply with the naming conventions described in Annex E.

The following types of artifacts can be specified by the **Type** entry:

- *Pagination artifacts*. Ancillary page features such as running heads and folios (page numbers).
- *Layout artifacts*. Purely cosmetic typographical or design elements such as footnote rules or background screens.
- *Page artifacts*. Production aids extraneous to the document itself, such as cut marks and colour bars.
- *Background artifacts*. Images, patterns or coloured blocks that either run the entire length and/or width of the page or the entire dimensions of a structural element. Background artifacts typically serve as a background for content shown either on top of or placed adjacent to that background.

A background artifact can further be classified as visual content that serves to enhance the user experience, that lies under the actual content, and that is not required except to retain visual fidelity.

NOTE 2 Examples of this include a coloured background, pattern, blend, or image that resides under main body text. In the case of white text on a black background, the black background is absolutely necessary to be able to read the white text; however, the background itself is merely there to enhance the visual experience. However, a draft or other identifying watermark is classified as a pagination artifact because it does not serve to enhance the experience; rather, it serves as a running artifact typically used on every page in the document. As a further example, a Figure differs from a background artifact in that removal of the graphics objects from a Figure would detract from the overall contextual understanding of the Figure as an entity.

- Tagged conforming readers may have their own ideas about what page content to consider relevant. A text-to-speech engine, for instance, probably should not speak running heads or page numbers when the page is turned. In general, conforming readers can do any of the following:
- Disregard elements of page content (for example, specific types of artifacts) that are not of interest
- Treat some page elements as *terminals* that are not to be examined further (for example, to treat an illustration as a unit for reflow purposes)
- Replace an element with alternate text (see 14.9.3, "Alternate Descriptions")

NOTE 3 Depending on their goals, different conforming readers can make different decisions in this regard. The purpose of Tagged PDF is not to prescribe what the conforming reader should do, but to provide sufficient declarative and descriptive information to allow it to make appropriate choices about how to process the content.

To support conforming readers in providing accessibility to users with disabilities, Tagged PDF documents should use the natural language specification (**Lang**), alternate description (**Alt**), replacement text (**ActualText**), and abbreviation expansion text (**E**) facilities described in 14.9, "Accessibility Support."

14.8.2.2.3 Incidental Artifacts

In addition to objects that are explicitly marked as artifacts and excluded from the document's logical structure, the running text of a page may contain other elements and relationships that are not logically part of the document's real content, but merely incidental results of the process of laying out that content into a document. They may include the following elements:

- *Hyphenation*. Among the artifacts introduced by text layout is the hyphen marking the incidental division of a word at the end of a line. In Tagged PDF, such an incidental word division shall be represented by a *soft hyphen* character, which the Unicode mapping algorithm (see "Unicode Mapping in Tagged PDF" in 14.8.2.4, "Extraction of Character Properties") translates to the Unicode value U+00AD. (This character is distinct from an ordinary *hard hyphen*, whose Unicode value is U+002D.) The producer of a Tagged PDF document shall distinguish explicitly between soft and hard hyphens so that the consumer does not have to guess which type a given character represents.

NOTE 1 In some languages, the situation is more complicated: there may be multiple hyphen characters, and hyphenation may change the spelling of words. See the Example in 14.9.4, "Replacement Text."

- *Text discontinuities*. The running text of a page, as expressed in page content order (see 14.8.2.3, "Page Content Order"), may contain places where the normal progression of text suffers a discontinuity. Conforming readers may recognize such discontinuities by examining the document's logical structure.

NOTE 2 For example, the page may contain the beginnings of two separate articles (see 12.4.3, "Articles"), each of which is continued onto a later page of the document. The last words of the first article appearing on the page should not be run together with the first words of the second article.

- *Hidden page elements*. For a variety of reasons, elements of a document's logical content may be invisible on the page: they may be clipped, their colour may match the background, or they may be obscured by other, overlapping objects. For the purposes of Tagged PDF, page content shall be considered to include all text and illustrations in their entirety, regardless of whether they are visible when the document is displayed or printed.

NOTE 3 For example, formerly invisible elements may become visible when a page is reflowed, or a text-to-speech engine may choose to speak text that is not visible to a sighted reader.

14.8.2.3 Page Content Order

14.8.2.3.1 General

When dealing with material on a page-by-page basis, some Tagged PDF conforming readers may choose to process elements in *page content order*, determined by the sequencing of graphics objects within a page's content stream and of characters within a text object, rather than in the *logical structure order* defined by a depth-first traversal of the page's logical structure hierarchy. The two orderings are logically distinct and may or may not coincide. In particular, any artifacts the page may contain shall be included in the page content order but not in the logical structure order, since they are not considered part of the document's logical structure. The conforming writer is responsible for establishing both an appropriate page content order for each page and an appropriate logical structure hierarchy for the entire document.

Because the primary requirement for page content order is to enable reflow to maintain elements in proper reading sequence, it should normally (for Western writing systems) proceed from top to bottom (and, in a multiple-column layout, from column to column), with artifacts in their correct relative places. In general, all parts of an article that appear on a given page should be kept together, even if the article flows to scattered locations on the page. Illustrations or footnotes may be interspersed with the text of the associated article or may appear at the end of its content (or, in the case of footnotes, at the end of the entire page's logical content).

In some situations, conforming writer may be unable to determine correct page content order for part of a document's contents. In such cases, *tag suspects (PDF 1.6)* can be used. The conforming writer shall identify suspect content by using marked content (see 14.6, "Marked Content") with a tag of TagSuspect, as shown in next Example. The marked content shall have a properties dictionary with an entry whose name is **TagSuspect** and whose value is **Ordering**, which indicates that the ordering of the enclosed marked content does not meet Tagged PDF specifications.

NOTE This can occur, for example, if content was extracted from another application, or if there are ambiguities or missing information in text output.

EXAMPLE
```
/TagSuspect <</TagSuspect /Ordering>>
    BDC
        ....                    % Problem page contents
    EMC
```

Documents containing tag suspects shall contain a **Suspects** entry with a value of **true** in the mark information dictionary (see Table 321).

14.8.2.3.2 Sequencing of Annotations

Annotations associated with a page are not interleaved within the page's content stream but shall be placed in the **Annots** array in its page object (see 7.7.3.3, "Page Objects"). Consequently, the correct position of an annotation in the page content order is not readily apparent but shall be determined from the document's logical structure.

Both page content (marked-content sequences) and annotations may be treated as content items that are referenced from structure elements (see 14.7.4, "Structure Content"). Structure elements of type Annot (*PDF 1.5*), Link, or Form (see 14.8.4.4, "Inline-Level Structure Elements," and 14.8.4.5, "Illustration Elements") explicitly specify the association between a marked-content sequence and a corresponding annotation. In other cases, if the structure element corresponding to an annotation immediately precedes or follows (in the logical structure order) a structure element corresponding to a marked-content sequence, the annotation is considered to precede or follow the marked-content sequence, respectively, in the page content order.

NOTE If necessary, a conforming writer may introduce an empty marked-content sequence solely to serve as a structure element for the purpose of positioning adjacent annotations in the page content order.

14.8.2.3.3 Reverse-Order Show Strings

NOTE 1 In writing systems that are read from right to left (such as Arabic or Hebrew), one might expect that the glyphs in a font would have their origins at the lower right and their widths (rightward horizontal displacements) specified as negative. For various technical and historical reasons, however, many such fonts follow the same conventions as those designed for Western writing systems, with glyph origins at the lower left and positive widths, as shown in Figure 39. Consequently, showing text in such right-to-left writing systems requires either positioning each glyph individually (which is tedious and costly) or representing text with show strings (see 9.2, "Organization and Use of Fonts") whose character codes are given in reverse order. When the latter method is used, the character codes' correct page content order is the reverse of their order within the show string.

The marked-content tag ReversedChars informs the conforming reader that show strings within a marked-content sequence contain characters in the reverse of page content order. If the sequence encompasses multiple show strings, only the individual characters within each string shall be reversed; the strings themselves shall be in natural reading order.

EXAMPLE The sequence

```
/ReversedChars
    BMC
        ( olleH) Tj
        −200 0 Td
        (.dlrow) Tj
    EMC
```

represents the text

Hello world.

The show strings may have a SPACE (U+0020) character at the beginning or end to indicate a word break (see 14.8.2.5, “Identifying Word Breaks”) but shall not contain interior SPACEs.

NOTE 2 This limitation is not serious, since a SPACE provides an opportunity to realign the typography without visible effect, and it serves the valuable purpose of limiting the scope of reversals for word-processing conforming readers.

14.8.2.4 Extraction of Character Properties

14.8.2.4.1 General

Tagged PDF enables character codes to be unambiguously converted to Unicode values representing the information content of the text. There are several methods for doing this; a Tagged PDF document shall conform to at least one of them (see “Unicode Mapping in Tagged PDF” in 14.8.2.4, “Extraction of Character Properties”). In addition, Tagged PDF enables some characteristics of the associated fonts to be deduced (see “Font Characteristics” in 14.8.2.4, “Extraction of Character Properties”).

NOTE These Unicode values and font characteristics can then be used for such operations as cut-and-paste editing, searching, text-to-speech conversion, and exporting to other applications or file formats.

14.8.2.4.2 Unicode Mapping in Tagged PDF

Tagged PDF requires that every character code in a document can be mapped to a corresponding Unicode value.

NOTE 1 Unicode defines scalar values for most of the characters used in the world’s languages and writing systems, as well as providing a private use area for application-specific characters. Information about Unicode can be found in the Unicode Standard, by the Unicode Consortium (see the Bibliography).

The methods for mapping a character code to a Unicode value are described in 9.10.2, “Mapping Character Codes to Unicode Values.” A conforming writer shall ensure that the PDF file contains enough information to map all character codes to Unicode by one of the methods described there.

NOTE 2 An **Alt**, **ActualText**, or **E** entry specified in a structure element dictionary or a marked-content property list (see 14.9.3, “Alternate Descriptions,” 14.9.4, “Replacement Text,” and 14.9.5, “Expansion of Abbreviations and Acronyms”) may affect the character stream that some conforming readers actually use. For example, some conforming readers may choose to use the **Alt** or **ActualText** value and ignore all text and other content associated with the structure element and its descendants.

NOTE 3 Some uses of Tagged PDF require characters that may not be available in all fonts, such as the soft hyphen (see 14.8.2.2.3, “Incidental Artifacts”). Such characters may be represented either by adding them to the font’s encoding or CMap and using **ToUnicode** to map them to appropriate Unicode values, or by using an **ActualText** entry in the associated structure element to provide substitute characters.

14.8.2.4.3 Font Characteristics

In addition to a Unicode value, each character code in a content stream has an associated set of font characteristics. These characteristics are not specified explicitly in the PDF file. Instead, the conforming reader derives the characteristics from the font descriptor for the font that is set in the text state at the time the character is shown.

NOTE These characteristics are useful when exporting text to another application or file format that has a limited repertoire of available fonts.

Table 331 lists a common set of font characteristics corresponding to those used in CSS and XSL; the W3C document *Extensible Stylesheet Language (XSL) 1.0* provides more information (see the Bibliography). Each of the characteristics shall be derived from information available in the font descriptor’s **Flags** entry (see 9.8.2, “Font Descriptor Flags”).

Table 331 – Derivation of font characteristics

Characteristic	Type	Derivation
Serifed	boolean	The value of the Serif flag in the font descriptor's **Flags** entry
Proportional	boolean	The complement of the FixedPitch flag in the font descriptor's **Flags** entry
Italic	boolean	The value of the Italic flag in the font descriptor's **Flags** entry
Smallcap	boolean	The value of the SmallCap flag in the font descriptor's **Flags** entry

The characteristics shown in the table apply only to character codes contained in show strings within content streams. They do not exist for alternate description text (**Alt**), replacement text (**ActualText**), or abbreviation expansion text (**E**).

For the standard 14 Type 1 fonts, the font descriptor may be missing; the well-known values for those fonts shall be used.

Tagged PDF in PDF 1.5 defines a wider set of font characteristics, which provide information needed when converting PDF to other files formats such as RTF, HTML, XML, and OEB, and also improve accessibility and reflow of tables. Table 332 lists these *font selector attributes* and shows how their values shall be derived.

If the *FontFamily*, *FontWeight* and *FontStretch* fields are not present in the font descriptor, these values shall be derived from the font name in a manner of the conforming reader's choosing.

Table 332 – Font selector attributes

Attribute	Description
FontFamily	A string specifying the preferred font family name. Derived from the **FontFamily** entry in the font descriptor (see Table 122).
GenericFontFamily	A general font classification, used if FontFamily is not found. Derived from the font descriptor's **Flags** entry as follows: Serif Chosen if the Serif flag is set and the FixedPitch and Script flags are not set SansSerif Chosen if the FixedPitch, Script and Serif flags are all not set Cursive Chosen if the Script flag is set and the FixedPitch flag is not set Monospace Chosen if the FixedPitch flag is set NOTE The values Decorative and Symbol cannot be derived
FontSize	The size of the font: a positive number specifying the height of the typeface in points. Derived from the a, b, c, and d fields of the current text matrix.
FontStretch	The stretch value of the font. Derived from **FontStretch** in the font descriptor (see Table 122).
FontStyle	The italicization value of the font. It shall be Italic if the Italic flag is set in the **Flags** field of the font descriptor; otherwise, it shall be Normal.
FontVariant	The small-caps value of the font. It shall be SmallCaps if the SmallCap flag is set in the **Flags** field of the font descriptor; otherwise, it shall be Normal.
FontWeight	The weight (thickness) value of the font. Derived from **FontWeight** in the font descriptor (see Table 122). The ForceBold flag and the **StemV** field should not be used to set this attribute.

14.8.2.5 Identifying Word Breaks

NOTE 1 A document's text stream defines not only the characters in a page's text but also the words. Unlike a character, the notion of a word is not precisely defined but depends on the purpose for which the text is being processed. A reflow tool needs to determine where it can break the running text into lines; a text-to-speech engine needs to identify the words to be vocalized; spelling checkers and other applications all have their own ideas of what constitutes a word. It is not important for a Tagged PDF document to identify the words within the text stream according to a single, unambiguous definition that satisfies all of these clients. What is important is that there be enough information available for each client to make that determination for itself.

A conforming reader of a Tagged PDF document may find words by sequentially examining the Unicode character stream, perhaps augmented by replacement text specified with **ActualText** (see 14.9.4, “Replacement Text”). For this purpose the spacing characters that would be present to separate words in a pure text representation shall be present in the Tagged PDF representation of the text.

NOTE 2 The conforming reader does not need to guess about word breaks based on information such as glyph positioning on the page, font changes, or glyph sizes.

NOTE 3 The identification of what constitutes a word is unrelated to how the text happens to be grouped into show strings. The division into show strings has no semantic significance. In particular, a SPACE (U+0020) or other word-breaking character is still needed even if a word break happens to fall at the end of a show string.

NOTE 4 Some conforming readers may identify words by simply separating them at every SPACE character. Others may be slightly more sophisticated and treat punctuation marks such as hyphens or em dashes as word separators as well. Still others may identify possible line-break opportunities by using an algorithm similar to the one in Unicode Standard Annex #29, Text Boundaries, available from the Unicode Consortium (see the Bibliography).

14.8.3 Basic Layout Model

The basic layout model begins with the notion of a *reference area*. This is a rectangular region used as a frame or guide in which to place the document's content. Some of the standard structure attributes, such as **StartIndent** and **EndIndent** (see 14.8.5.4.3, “Layout Attributes for BLSEs”), shall be measured from the boundaries of the reference area. Reference areas are not specified explicitly but are inferred from context. Those of interest are generally the column area or areas in a general text layout, the outer bounding box of a table and those of its component cells, and the bounding box of an illustration or other floating element.

NOTE 1 Tagged PDF's standard structure types and attributes shall be interpreted in the context of a basic layout model that describes the arrangement of structure elements on the page. This model is designed to capture the general intent of the document's underlying structure and does not necessarily correspond to the one actually used for page layout by the application creating the document. (The PDF content stream specifies the exact appearance.) The goal is to provide sufficient information for conforming readers to make their own layout decisions while preserving the authoring application's intent as closely as their own layout models allow.

NOTE 2 The Tagged PDF layout model resembles the ones used in markup languages such as HTML, CSS, XSL, and RTF, but does not correspond exactly to any of them. The model is deliberately defined loosely to allow reasonable latitude in the interpretation of structure elements and attributes when converting to other document formats. Some degree of variation in the resulting layout from one format to another is to be expected.

The standard structure types are divided into four main categories according to the roles they play in page layout:

- *Grouping elements* (see 14.8.4.2, “Grouping Elements”) group other elements into sequences or hierarchies but hold no content directly and have no direct effect on layout.
- *Block-level structure elements* (*BLSE*s) (see 14.8.4.3, “Block-Level Structure Elements”) describe the overall layout of content on the page, proceeding in the *block-progression direction*.
- *Inline-level structure elements* (*ILSE*s) (see 14.8.4.4, “Inline-Level Structure Elements”) describe the layout of content within a BLSE, proceeding in the *inline-progression direction*.

- *Illustration elements* (see 14.8.4.5, “Illustration Elements”) are compact sequences of content, in page content order, that are considered to be unitary objects with respect to page layout. An illustration can be treated as either a BLSE or an ILSE.

The meaning of the terms *block-progression direction* and *inline-progression direction* depends on the writing system in use, as specified by the standard attribute **WritingMode** (see 14.8.5.4.2, “General Layout Attributes”). In Western writing systems, the block direction is from top to bottom and the inline direction is from left to right. Other writing systems use different directions for laying out content.

Because the progression directions can vary depending on the writing system, edges of areas and directions on the page are identified by terms that are neutral with respect to the progression order rather than by familiar terms such as *up*, *down*, *left*, and *right*. Block layout proceeds from *before* to *after*, inline from *start* to *end*. Thus, for example, in Western writing systems, the before and after edges of a reference area are at the top and bottom, respectively, and the start and end edges are at the left and right. Another term, *shift direction* (the direction of shift for a superscript), refers to the direction opposite that for block progression—that is, from after to before (in Western writing systems, from bottom to top).

BLSEs shall be *stacked* within a reference area in block-progression order. In general, the first BLSE shall be placed against the before edge of the reference area. Subsequent BLSEs shall be stacked against preceding ones, progressing toward the after edge, until no more BLSEs fit in the reference area. If the overflowing BLSE allows itself to be split—such as a paragraph that can be split between lines of text—a portion of it may be included in the current reference area and the remainder carried over to a subsequent reference area (either elsewhere on the same page or on another page of the document). Once the amount of content that fits in a reference area is determined, the placements of the individual BLSEs may be adjusted to bias the placement toward the before edge, the middle, or the after edge of the reference area, or the spacing within or between BLSEs may be adjusted to fill the full extent of the reference area.

BLSEs may be nested, with child BLSEs stacked within a parent BLSE in the same manner as BLSEs within a reference area. Except in a few instances noted (the BlockAlign and InlineAlign elements), such nesting of BLSEs does not result in the nesting of reference areas; a single reference area prevails for all levels of nested BLSEs.

Within a BLSE, child ILSEs shall be *packed* into *lines*. *Direct content items*—those that are immediate children of a BLSE rather than contained within a child ILSE—shall be implicitly treated as ILSEs for packing purposes. Each line shall be treated as a synthesized BLSE and shall be stacked within the parent BLSE. Lines may be intermingled with other BLSEs within the parent area. This line-building process is analogous to the stacking of BLSEs within a reference area, except that it proceeds in the inline-progression rather than the block-progression direction: a line shall be packed with ILSEs beginning at the start edge of the containing BLSE and continuing until the end edge shall be reached and the line is full. The overflowing ILSE may allow itself to be broken at linguistically determined or explicitly marked break points (such as hyphenation points within a word), and the remaining fragment shall be carried over to the next line.

Certain values of an element’s **Placement** attribute remove the element from the normal stacking or packing process and allow it instead to float to a specified edge of the enclosing reference area or parent BLSE; see “General Layout Attributes” in 14.8.5.4, “Layout Attributes,” for further discussion.

Two enclosing rectangles shall be associated with each BLSE and ILSE (including direct content items that are treated implicitly as ILSEs):

- The *content rectangle* shall be derived from the shape of the enclosed content and defines the bounds used for the layout of any included child elements.

- The *allocation rectangle* includes any additional borders or spacing surrounding the element, affecting how it shall be positioned with respect to adjacent elements and the enclosing content rectangle or reference area.

The definitions of these rectangles shall be determined by layout attributes associated with the structure element; see 14.8.5.4.5, “Content and Allocation Rectangles” for further discussion.

14.8.4 Standard Structure Types

14.8.4.1 General

Tagged PDF's *standard structure types* characterize the role of a content element within the document and, in conjunction with the standard structure attributes (described in 14.8.5, "Standard Structure Attributes"), how that content is laid out on the page. As discussed in 14.7.3, "Structure Types," the structure type of a logical structure element shall be specified by the **S** entry in its structure element dictionary. To be considered a standard structure type, this value shall be either:

- One of the standard structure type names described in 14.8.4.2, "Grouping Elements."
- An arbitrary name that shall be mapped to one of the standard names by the document's role map (see 14.7.3, "Structure Types"), possibly through multiple levels of mapping.

NOTE 1 Beginning with PDF 1.5, an element name is always mapped to its corresponding name in the role map, if there is one, even if the original name is one of the standard types. This is done to allow the element, for example, to represent a tag with the same name as a standard role, even though its use differs from the standard role.

Ordinarily, structure elements having standard structure types shall be processed the same way whether the type is expressed directly or is determined indirectly from the role map. However, some conforming readers may ascribe additional semantics to nonstandard structure types, even though the role map associates them with standard ones.

NOTE 2 For instance, the actual values of the **S** entries may be used when exporting to a tagged representation such as XML, and the corresponding role-mapped values shall be used when converting to presentation formats such as HTML or RTF, or for purposes such as reflow or accessibility to users with disabilities.

NOTE 3 Most of the standard element types are designed primarily for laying out text; the terminology reflects this usage. However, a layout may in fact include any type of content, such as path or image objects.

The content items associated with a structure element shall be laid out on the page as if they were blocks of text (for a BLSE) or characters within a line of text (for an ILSE).

14.8.4.2 Grouping Elements

Grouping elements shall be used solely to group other structure elements; they are not directly associated with content items. Table 333 describes the standard structure types for elements in this category. H.8, "Structured Elements That Describe Hierarchical Lists" provides an example of nested table of content items.

In a tagged PDF document, the structure tree shall contain a single top-level element; that is, the structure tree root (identified by the **StructTreeRoot** entry in the document catalogue) shall have only one child in its **K** (kids) array. If the PDF file contains a complete document, the structure type Document should be used for this top-level element in the logical structure hierarchy. If the file contains a well-formed document fragment, one of the structure types Part, Art, Sect, or Div may be used instead.

Table 333 – Standard structure types for grouping elements

Structure type	Description
Document	(Document) A complete document. This is the root element of any structure tree containing multiple parts or multiple articles.
Part	(Part) A large-scale division of a document. This type of element is appropriate for grouping articles or sections.
Art	(Article) A relatively self-contained body of text constituting a single narrative or exposition. Articles should be disjoint; that is, they should not contain other articles as constituent elements.

Table 333 – Standard structure types for grouping elements (continued)

Structure type	Description
Sect	(Section) A container for grouping related content elements. NOTE 1 For example, a section might contain a heading, several introductory paragraphs, and two or more other sections nested within it as subsections.
Div	(Division) A generic block-level element or group of elements.
BlockQuote	(Block quotation) A portion of text consisting of one or more paragraphs attributed to someone other than the author of the surrounding text.
Caption	(Caption) A brief portion of text describing a table or figure.
TOC	(Table of contents) A list made up of table of contents item entries (structure type TOCI) and/or other nested table of contents entries (TOC). A TOC entry that includes only TOCI entries represents a flat hierarchy. A TOC entry that includes other nested TOC entries (and possibly TOCI entries) represents a more complex hierarchy. Ideally, the hierarchy of a top level TOC entry reflects the structure of the main body of the document. NOTE 2 Lists of figures and tables, as well as bibliographies, can be treated as tables of contents for purposes of the standard structure types.
TOCI	(Table of contents item) An individual member of a table of contents. This entry's children may be any of the following structure types: Lbl A label (see "List Elements" in 14.8.4.3, "Block-Level Structure Elements") Reference A reference to the title and the page number (see "Inline-Level Structure Elements" in 14.8.4.4, "Inline-Level Structure Elements") NonStruct Non-structure elements for wrapping a leader artifact (see "Grouping Elements" in 14.8.4.2, "Grouping Elements"). P Descriptive text (see "Paragraphlike Elements" 14.8.4.3, "Block-Level Structure Elements") TOC Table of content elements for hierarchical tables of content, as described for the TOC entry
Index	(Index) A sequence of entries containing identifying text accompanied by reference elements (structure type Reference; see 14.8.4.4, "Inline-Level Structure Elements") that point out occurrences of the specified text in the main body of a document.
NonStruct	(Nonstructural element) A grouping element having no inherent structural significance; it serves solely for grouping purposes. This type of element differs from a division (structure type Div) in that it shall not be interpreted or exported to other document formats; however, its descendants shall be processed normally.
Private	(Private element) A grouping element containing private content belonging to the application producing it. The structural significance of this type of element is unspecified and shall be determined entirely by the conforming writer. Neither the Private element nor any of its descendants shall be interpreted or exported to other document formats.

14.8.4.3 Block-Level Structure Elements

14.8.4.3.1 General

A *block-level structure element (BLSE)* is any region of text or other content that is laid out in the block-progression direction, such as a paragraph, heading, list item, or footnote. A structure element is a BLSE if its structure type (after role mapping, if any) is one of those listed in Table 334. All other standard structure types shall be treated as ILSEs, with the following exceptions:

- TR (Table row), TH (Table header), TD (Table data), THead (Table head), TBody (Table body), and TFoot (Table footer), which shall be used to group elements within a table and shall be considered neither BLSEs nor ILSEs
- Elements with a **Placement** attribute (see “General Layout Attributes” in 14.8.5.4, “Layout Attributes”) other than the default value of Inline

Table 334 – Block-level structure elements

Category	Structure types		
Paragraphlike elements	P H	H1 H2 H3	H4 H5 H6
List elements	L LI	Lbl LBody	
Table element	Table		

In many cases, a BLSE may appear as one compact, contiguous piece of page content; in other cases, it may be discontiguous.

NOTE Examples of the latter include a BLSE that extends across a page boundary or is interrupted in the page content order by another, nested BLSE or a directly included footnote. When necessary, Tagged conforming readers can recognize such fragmented BLSEs from the logical structure and use this information to reassemble them and properly lay them out.

14.8.4.3.2 Paragraphlike Elements

Table 335 describes structure types for *paragraphlike elements* that consist of running text and other content laid out in the form of conventional paragraphs (as opposed to more specialized layouts such as lists and tables).

Table 335 – Standard structure types for paragraphlike elements

Structure Type	Description
H	(Heading) A label for a subdivision of a document’s content. It should be the first child of the division that it heads.
H1–H6	Headings with specific levels, for use in conforming writers that cannot hierarchically nest their sections and thus cannot determine the level of a heading from its level of nesting.
P	(Paragraph) A low-level division of text.

14.8.4.3.3 List Elements

Table 336 describes structure types for organizing the content of lists. H.8, “Structured Elements That Describe Hierarchical Lists” provides an example of nested list entries.

Table 336 – Standard structure types for list elements

Structure Type	Description
L	(List) A sequence of items of like meaning and importance. Its immediate children should be an optional caption (structure type Caption; see 14.8.4.2, “Grouping Elements”) followed by one or more list items (structure type LI).
LI	(List item) An individual member of a list. Its children may be one or more labels, list bodies, or both (structure types Lbl or LBody).

Table 336 – Standard structure types for list elements (continued)

Structure Type	Description
Lbl	(Label) A name or number that distinguishes a given item from others in the same list or other group of like items. NOTE In a dictionary list, for example, it contains the term being defined; in a bulleted or numbered list, it contains the bullet character or the number of the list item and associated punctuation.
LBody	(List body) The descriptive content of a list item. In a dictionary list, for example, it contains the definition of the term. It may either contain the content directly or have other BLSEs, perhaps including nested lists, as children.

14.8.4.3.4 Table Elements

The structure types described in Table 337 shall be used for organizing the content of tables.

NOTE 1 Strictly speaking, the *Table* element is a BLSE; the others in this table are neither BLSEs or ILSEs.

Table 337 – Standard structure types for table elements

Structure Type	Description
Table	(Table) A two-dimensional layout of rectangular data cells, possibly having a complex substructure. It contains either one or more table rows (structure type TR) as children; or an optional table head (structure type THead) followed by one or more table body elements (structure type TBody) and an optional table footer (structure type TFoot). In addition, a table may have a caption (structure type Caption; see 14.8.4.2, “Grouping Elements”) as its first or last child.
TR	(Table row) A row of headings or data in a table. It may contain table header cells and table data cells (structure types TH and TD).
TH	(Table header cell) A table cell containing header text describing one or more rows or columns of the table.
TD	(Table data cell) A table cell containing data that is part of the table’s content.
THead	(Table header row group; *PDF 1.5*) A group of rows that constitute the header of a table. If the table is split across multiple pages, these rows may be redrawn at the top of each table fragment (although there is only one THead element).
TBody	(Table body row group; *PDF 1.5*) A group of rows that constitute the main body portion of a table. If the table is split across multiple pages, the body area may be broken apart on a row boundary. A table may have multiple TBody elements to allow for the drawing of a border or background for a set of rows.
TFoot	(Table footer row group; *PDF 1.5*) A group of rows that constitute the footer of a table. If the table is split across multiple pages, these rows may be redrawn at the bottom of each table fragment (although there is only one TFoot element.)

NOTE 2 The association of headers with rows and columns of data is typically determined heuristically by applications. Such heuristics may fail for complex tables; the standard attributes for tables shown in Table 348 can be used to make the association explicit.

14.8.4.3.5 Usage Guidelines for Block-Level Structure

Because different conforming readers use PDF’s logical structure facilities in different ways, Tagged PDF does not enforce any strict rules regarding the order and nesting of elements using the standard structure types. Furthermore, each export format has its own conventions for logical structure. However, adhering to certain general guidelines helps to achieve the most consistent and predictable interpretation among different Tagged PDF consumers.

As described under 14.8.4.2, “Grouping Elements,” a Tagged PDF document may have one or more levels of grouping elements, such as Document, Part, Art (Article), Sect (Section), and Div (Division). The descendants of these should be BLSEs, such as H (Heading), P (Paragraph), and L (List), that hold the actual content. Their descendants, in turn, should be either content items or ILSEs that further describe the content.

NOTE 1 As noted earlier, elements with structure types that would ordinarily be treated as ILSEs may have a ***Placement*** attribute (see “General Layout Attributes” in 14.8.5.4, “Layout Attributes”) that causes them to be treated as BLSEs instead. Such elements may be included as BLSEs in the same manner as headings and paragraphs.

The block-level structure may follow one of two principal paradigms:

- *Strongly structured*. The grouping elements nest to as many levels as necessary to reflect the organization of the material into articles, sections, subsections, and so on. At each level, the children of the grouping element should consist of a heading (H), one or more paragraphs (P) for content at that level, and perhaps one or more additional grouping elements for nested subsections.

- *Weakly structured*. The document is relatively flat, having perhaps only one or two levels of grouping elements, with all the headings, paragraphs, and other BLSEs as their immediate children. In this case, the organization of the material is not reflected in the logical structure; however, it may be expressed by the use of headings with specific levels (H1–H6).

NOTE 2 The strongly structured paradigm is used by some rich document models based on XML. The weakly structured paradigm is typical of documents represented in HTML.

Lists and tables should be organized using the specific structure types described under “List Elements” in 14.8.4.3, “Block-Level Structure Elements,” and “Table Elements” in 14.8.4.3, “Block-Level Structure Elements”. Likewise, tables of contents and indexes should be structured as described for the TOC and Index structure types under “Grouping Elements” in 14.8.4.2, “Grouping Elements.”

14.8.4.4 Inline-Level Structure Elements

14.8.4.4.1 General

An *inline-level structure element (ILSE)* contains a portion of text or other content having specific styling characteristics or playing a specific role in the document. Within a paragraph or other block defined by a containing BLSE, consecutive ILSEs—possibly intermixed with other content items that are direct children of the parent BLSE—are laid out consecutively in the inline-progression direction (left to right in Western writing systems). The resulting content may be broken into multiple *lines*, which in turn shall be stacked in the block-progression direction. An ILSE may in turn contain a BLSE, which shall be treated as a unitary item of layout in the inline direction. Table 338 lists the standard structure types for ILSEs.

Table 338 – Standard structure types for inline-level structure elements

Structure Type	Description
Span	(Span) A generic inline portion of text having no particular inherent characteristics. It can be used, for example, to delimit a range of text with a given set of styling attributes. NOTE 1 Not all inline style changes need to be identified as a span. Text colour and font changes (including modifiers such as bold, italic, and small caps) need not be so marked, since these can be derived from the PDF content (see “Font Characteristics” in 14.8.2.4, “Extraction of Character Properties”). However, it is necessary to use a span to apply explicit layout attributes such as **LineHeight**, **BaselineShift**, or **TextDecorationType** (see “Layout Attributes for ILSEs” in 14.8.5.4, “Layout Attributes”). NOTE 2 Marked-content sequences having the tag Span are also used to carry certain accessibility properties (**Alt**, **ActualText**, **Lang**, and **E**; see 14.9, “Accessibility Support”). Such sequences lack an **MCID** property and are not associated with any structure element. This use of the Span marked-content tag is distinct from its use as a structure type.
Quote	(Quotation) An inline portion of text attributed to someone other than the author of the surrounding text. The quoted text should be contained inline within a single paragraph. This differs from the block-level element BlockQuote (see 14.8.4.2, “Grouping Elements”), which consists of one or more complete paragraphs (or other elements presented as if they were complete paragraphs).
Note	(Note) An item of explanatory text, such as a footnote or an endnote, that is referred to from within the body of the document. It may have a label (structure type Lbl; see “List Elements” in 14.8.4.3, “Block-Level Structure Elements”) as a child. The note may be included as a child of the structure element in the body text that refers to it, or it may be included elsewhere (such as in an endnotes section) and accessed by means of a reference (structure type Reference). Tagged PDF does not prescribe the placement of footnotes in the page content order. They may be either inline or at the end of the page, at the discretion of the conforming writer.
Reference	(Reference) A citation to content elsewhere in the document.
BibEntry	(Bibliography entry) A reference identifying the external source of some cited content. It may contain a label (structure type Lbl; see “List Elements” in 14.8.4.3, “Block-Level Structure Elements”) as a child. Although a bibliography entry is likely to include component parts identifying the cited content’s author, work, publisher, and so forth, no standard structure types are defined at this level of detail.
Code	(Code) A fragment of computer program text.
Link	(Link) An association between a portion of the ILSE’s content and a corresponding link annotation or annotations (see 12.5.6.5, “Link Annotations”). Its children should be one or more content items or child ILSEs and one or more object references (see 14.7.4.3, “PDF Objects as Content Items”) identifying the associated link annotations. See “Link Elements” in 14.8.4.3, “Block-Level Structure Elements,” for further discussion.
Annot	(Annotation; PDF 1.5) An association between a portion of the ILSE’s content and a corresponding PDF annotation (see 12.5, “Annotations”). Annot shall be used for all PDF annotations except link annotations (see the Link element) and widget annotations (see the Form element in Table 340). See “Annotation Elements” 14.8.4.4, “Inline-Level Structure Elements,” for further discussion.

Table 338 – Standard structure types for inline-level structure elements (continued)

Structure Type	Description
Ruby	(Ruby; PDF 1.5) A side-note (annotation) written in a smaller text size and placed adjacent to the base text to which it refers. A Ruby element may also contain the RB, RT, and RP elements. See “Ruby and Warichu Elements” in14.8.4.4, “Inline-Level Structure Elements,” for more details.
Warichu	(Warichu; PDF 1.5) A comment or annotation in a smaller text size and formatted onto two smaller lines within the height of the containing text line and placed following (inline) the base text to which it refers. A Warichu element may also contain the WT and WP elements. See “Ruby and Warichu Elements” in14.8.4.4, “Inline-Level Structure Elements,” for more details.

14.8.4.4.2 Link Elements

NOTE 1 Link annotations (like all PDF annotations) are associated with a geometric region of the page rather than with a particular object in its content stream. Any connection between the link and the content is based solely on visual appearance rather than on an explicitly specified association. For this reason, link annotations alone are not useful to users with visual impairments or to applications needing to determine which content can be activated to invoke a hypertext link.

Tagged PDF link elements (structure type Link) use PDF’s logical structure facilities to establish the association between content items and link annotations, providing functionality comparable to HTML hypertext links. The following items may be children of a link element:

- One or more content items or other ILSEs (except other links)
- Object references (see 14.7.4.3, “PDF Objects as Content Items”) to one or more link annotations associated with the content

When a **Link** structure element describes a span of text to be associated with a link annotation and that span wraps from the end of one line to the beginning of another, the **Link** structure element shall include a single object reference that associates the span with the associated link annotation. Further, the link annotation shall use the **QuadPoint** entry to denote the active areas on the page.

EXAMPLE 1 The **Link** structure element references a link annotation that includes a **QuadPoint** entry that boxes the strings “with a” and “link”. That is, the **QuadPoint** entry contains 16 numbers: the first 8 numbers describe a quadrilateral for “with a”, and the next 8 describe a quadrilateral for “link.”

Here is some text with a
link inside.

NOTE 2 Beginning with PDF 1.7, use of the **Link** structure element to enclose multiple link annotations is deprecated.

EXAMPLE 2 Consider the following fragment of HTML code, which produces a line of text containing a hypertext link:

```
<html>
    <body>
        <p>
        Here is some text <a href=http://www.adobe.com>with a link</a> inside.
    </body>
</html>
```

This code sample shows an equivalent fragment of PDF using a link element, whose text it displays in blue and underlined.

```
/P << /MCID 0 >>                                   % Marked-content sequence 0 (paragraph)
    BDC                                            % Begin marked-content sequence
```

```
            BT                                          % Begin text object
                /T1_0  1  Tf                            % Set text font and size
                14  0  0  14  10.000  753.976  Tm       % Set text matrix
                0.0  0.0  0.0  rg                       % Set nonstroking colour to black
                (Here is some text )  Tj                % Show text preceding link
            ET                                          % End text object
        EMC                                             % End marked-content sequence

    /Link  << /MCID  1 >>                               % Marked-content sequence 1 (link)
        BDC                                             % Begin marked-content sequence
            0.7  w                                      % Set line width
            [ ]  0  d                                   % Solid dash pattern

            111.094  751.8587  m                        % Move to beginning of underline
            174.486  751.8587  l                        % Draw underline
            0.0  0.0  1.0  RG                           % Set stroking colour to blue
            S                                           % Stroke underline

            BT                                          % Begin text object
                14  0  0  14  111.094  753.976  Tm      % Set text matrix
                0.0  0.0  1.0  rg                       % Set nonstroking colour to blue
                (with a link)  Tj                       % Show text of link
            ET                                          % End text object
        EMC                                             % End marked-content sequence

    /P  << /MCID  2 >>                                  % Marked-content sequence 2 (paragraph)
        BDC                                             % Begin marked-content sequence
            BT                                          % Begin text object
                14  0  0  14  174.486  753.976  Tm      % Set text matrix
                0.0  0.0  0.0  rg                       % Set nonstroking colour to black
                ( inside.)  Tj                          % Show text following link
            ET                                          % End text object
        EMC                                             % End marked-content sequence
```

EXAMPLE 3 This example shows an excerpt from the associated logical structure hierarchy.

```
501  0  obj                         % Structure element for paragraph
    << /Type  /StructElem
        /S  /P
        . . .
        /K  [  0                        % Three children: marked-content sequence 0
            502 0 R                     %     Link
            2                           %     Marked-content sequence 2
            ]
    >>
endobj

502  0  obj                         % Structure element for link
    << /Type  /StructElem
        /S  /Link
        . . .
        /K  [  1                        % Two children: marked-content sequence 1
            503 0 R                     %     Object reference to link annotation
            ]
    >>
endobj

503  0  obj                         % Object reference to link annotation
    << /Type  /OBJR
        /Obj  600 0 R                   % Link annotation (not shown)
    >>
endobj
```

14.8.4.4.3 Annotation Elements

Tagged PDF annotation elements (structure type Annot; *PDF 1.5*) use PDF's logical structure facilities to establish the association between content items and PDF annotations. Annotation elements shall be used for all types of annotations other than links (see "Link Elements" in 14.8.4.3, "Block-Level Structure Elements") and forms (see Table 340).

The following items may be children of an annotation element:

- Object references (see 14.7.4.3, "PDF Objects as Content Items") to one or more annotation dictionaries
- Optionally, one or more content items (such as marked-content sequences) or other ILSEs (except other annotations) associated with the annotations

If an Annot element has no children other than object references, its rendering shall be defined by the appearance of the referenced annotations, and its text content shall be treated as if it were a Span element. It may have an optional **BBox** attribute; if supplied, this attribute overrides the rectangle specified by the annotation dictionary's **Rect** entry.

If the Annot element has children that are content items, those children represent the displayed form of the annotation, and the appearance of the associated annotation may also be applied (for example, with a **Highlight** annotation).

There may be multiple children that are object references to different annotations, subject to the constraint that the annotations shall be the same except for their **Rect** entry. This is much the same as is done for the Link element; it allows an annotation to be associated with discontiguous pieces of content, such as line-wrapped text.

14.8.4.4.4 Ruby and Warichu Elements

Ruby text is a side note, written in a smaller text size and placed adjacent to the base text to which it refers. It is used in Japanese and Chinese to describe the pronunciation of unusual words or to describe such items as abbreviations and logos.

Warichu text is a comment or annotation, written in a smaller text size and formatted onto two smaller lines within the height of the containing text line and placed following (inline) the base text to which it refers. It is used in Japanese for descriptive comments and for ruby annotation text that is too long to be aesthetically formatted as a ruby.

Table 339 – Standard structure types for Ruby and Warichu elements (*PDF 1.5*)

Structure Type	Description
Ruby	(Ruby) The wrapper around the entire ruby assembly. It shall contain one RB element followed by either an RT element or a three-element group consisting of RP, RT, and RP. Ruby elements and their content elements shall not break across multiple lines.
RB	(Ruby base text) The full-size text to which the ruby annotation is applied. RB may contain text, other inline elements, or a mixture of both. It may have the **RubyAlign** attribute.
RT	(Ruby annotation text) The smaller-size text that shall be placed adjacent to the ruby base text. It may contain text, other inline elements, or a mixture of both. It may have the **RubyAlign** and **RubyPosition** attributes.

Table 339 – Standard structure types for Ruby and Warichu elements (*PDF 1.5*) (continued)

Structure Type	Description
RP	(Ruby punctuation) Punctuation surrounding the ruby annotation text. It is used only when a ruby annotation cannot be properly formatted in a ruby style and instead is formatted as a normal comment, or when it is formatted as a warichu. It contains text (usually a single LEFT or RIGHT PARENTHESIS or similar bracketing character).
Warichu	(Warichu) The wrapper around the entire warichu assembly. It may contain a three-element group consisting of WP, WT, and WP. Warichu elements (and their content elements) may wrap across multiple lines, according to the warichu breaking rules described in the Japanese Industrial Standard (JIS) X 4051-1995.
WT	(Warichu text) The smaller-size text of a warichu comment that is formatted into two lines and placed between surrounding WP elements.
WP	(Warichu punctuation) The punctuation that surrounds the WT text. It contains text (usually a single LEFT or RIGHT PARENTHESIS or similar bracketing character). According to JIS X 4051-1995, the parentheses surrounding a warichu may be converted to a SPACE (nominally 1/4 EM in width) at the discretion of the formatter.

14.8.4.5 Illustration Elements

Tagged PDF defines an *illustration element* as any structure element whose structure type (after role mapping, if any) is one of those listed in Table 340. The illustration's content shall consist of one or more complete graphics objects. It shall not appear between the **BT** and **ET** operators delimiting a text object (see 9.4, "Text Objects"). It may include clipping only in the form of a contained marked clipping sequence, as defined in 14.6.3, "Marked Content and Clipping." In Tagged PDF, all such marked clipping sequences shall carry the marked-content tag Clip.

Table 340 – Standard structure types for illustration elements

Structure Type	Description
Figure	(Figure) An item of graphical content. Its placement may be specified with the **Placement** layout attribute (see "General Layout Attributes" in 14.8.5.4, "Layout Attributes").
Formula	(Formula) A mathematical formula. This structure type is useful only for identifying an entire content element as a formula. No standard structure types are defined for identifying individual components within the formula. From a formatting standpoint, the formula shall be treated similarly to a figure (structure type Figure).
Form	(Form) A widget annotation representing an interactive form field (see 12.7, "Interactive Forms"). If the element contains a **Role** attribute, it may contain content items that represent the value of the (non-interactive) form field. If the element omits a **Role** attribute (see Table 348), it shall have only one child: an object reference (see 14.7.4.3, "PDF Objects as Content Items") identifying the widget annotation. The annotations' appearance stream (see 12.5.5, "Appearance Streams") shall describe the appearance of the form element.

An illustration may have logical substructure, including other illustrations. For purposes of reflow, however, it shall be moved (and perhaps resized) as a unit, without examining its internal contents. To be useful for reflow, it shall have a **BBox** attribute. It may also have **Placement**, **Width**, **Height**, and **BaselineShift** attributes (see 14.8.5.4, "Layout Attributes").

Often an illustration is logically part of, or at least attached to, a paragraph or other element of a document. Any such containment or attachment shall be represented through the use of the Figure structure type. The Figure element indicates the point of attachment, and its **Placement** attribute describes the nature of the attachment. An illustration element without a **Placement** attribute shall be treated as an ILSE and laid out inline.

For accessibility to users with disabilities and other text extraction purposes, an illustration element should have an **Alt** entry or an **ActualText** entry (or both) in its structure element dictionary (see 14.9.3, "Alternate Descriptions," and 14.9.4, "Replacement Text"). **Alt** is a description of the illustration, whereas **ActualText** gives the exact text equivalent of a graphical illustration that has the appearance of text.

14.8.5 Standard Structure Attributes

14.8.5.1 General

In addition to the standard structure types, Tagged PDF defines standard layout and styling attributes for structure elements of those types. These attributes enable predictable formatting to be applied during operations such as reflow and export of PDF content to other document formats.

As discussed in 14.7.5, "Structure Attributes," attributes shall be defined in *attribute objects*, which are dictionaries or streams attached to a structure element in either of two ways:

- The **A** entry in the structure element dictionary identifies an attribute object or an array of such objects.
- The **C** entry in the structure element dictionary gives the name of an *attribute class* or an array of such names. The class name is in turn looked up in the *class map*, a dictionary identified by the **ClassMap** entry in the structure tree root, yielding an attribute object or array of objects corresponding to the class.

In addition to the standard structure attributes described in 14.8.5.2, "Standard Attribute Owners," there are several other optional entries—**Lang**, **Alt**, **ActualText**, and **E**—that are described in 14.9, "Accessibility Support," but are useful to other PDF consumers as well. They appear in the following places in a PDF file (rather than in attribute dictionaries):

- As entries in the structure element dictionary (see Table 323)
- As entries in property lists attached to marked-content sequences with a Span tag (see 14.6, "Marked Content")

The Example in 14.7.6, "Example of Logical Structure," illustrates the use of standard structure attributes.

14.8.5.2 Standard Attribute Owners

Each attribute object has an *owner*, specified by the object's **O** entry, which determines the interpretation of the attributes defined in the object's dictionary. Multiple owners may define like-named attributes with different value types or interpretations. Tagged PDF defines a set of standard attribute owners, shown in Table 341.

Table 341 – Standard attribute owners

Owner	Description
Layout	Attributes governing the layout of content
List	Attributes governing the numbering of lists
PrintField	*(PDF 1.7)* Attributes governing Form structure elements for non-interactive form fields
Table	Attributes governing the organization of cells in tables

Table 341 – Standard attribute owners (continued)

Owner	Description
XML-1.00	Additional attributes governing translation to XML, version 1.00
HTML-3.20	Additional attributes governing translation to HTML, version 3.20
HTML-4.01	Additional attributes governing translation to HTML, version 4.01
OEB-1.00	Additional attributes governing translation to OEB, version 1.0
RTF-1.05	Additional attributes governing translation to Microsoft Rich Text Format, version 1.05
CSS-1.00	Additional attributes governing translation to a format using CSS, version 1.00
CSS-2.00	Additional attributes governing translation to a format using CSS, version 2.00

An attribute object owned by a specific export format, such as **XML-1.00**, shall be applied only when exporting PDF content to that format. Such format-specific attributes shall override any corresponding attributes owned by **Layout**, **List**, **PrintField**, or **Table**. There may also be additional format-specific attributes; the set of possible attributes is open-ended and is not explicitly specified or limited by Tagged PDF.

14.8.5.3 Attribute Values and Inheritance

Some attributes are defined as *inheritable*. Inheritable attributes propagate down the structure tree; that is, an attribute that is specified for an element shall apply to all the descendants of the element in the structure tree unless a descendent element specifies an explicit value for the attribute.

NOTE 1 The description of each of the standard attributes in this sub-clause specifies whether their values are inheritable.

An inheritable attribute may be specified for an element for the purpose of propagating its value to child elements, even if the attribute is not meaningful for the parent element. Non-inheritable attributes may be specified only for elements on which they would be meaningful.

The following list shows the priority for determining attribute values. A conforming reader determines an attribute’s value to be the first item in the following list that applies:

a) The value of the attribute specified in the element’s **A** entry, owned by one of the export formats (such as **XML**, **HTML-3.20**, **HTML-4.01**, **OEB-1.0**, **CSS-1.00**, **CSS-2.0**, and **RTF**), if present, and if outputting to that format

b) The value of the attribute specified in the element’s **A** entry, owned by **Layout**, **PrintField**, **Table** or **List**, if present

c) The value of the attribute specified in a class map associated with the element’s **C** entry, if there is one

d) The resolved value of the parent structure element, if the attribute is inheritable

e) The default value for the attribute, if there is one

NOTE 2 The attributes ***Lang***, ***Alt***, ***ActualText***, and ***E*** do not appear in attribute dictionaries. The rules governing their application are discussed in 14.9, “Accessibility Support.”

There is no semantic distinction between attributes that are specified explicitly and ones that are inherited. Logically, the structure tree has attributes fully bound to each element, even though some may be inherited from an ancestor element. This is consistent with the behaviour of properties (such as font characteristics) that are not specified by structure attributes but shall be derived from the content.

14.8.5.4 Layout Attributes

14.8.5.4.1 General

Layout attributes specify parameters of the layout process used to produce the appearance described by a document's PDF content. Attributes in this category shall be defined in attribute objects whose **O** (owner) entry has the value **Layout** (or is one of the format-specific owner names listed in Table 341).

NOTE The intent is that these parameters can be used to reflow the content or export it to some other document format with at least basic styling preserved.

Table 342 summarizes the standard layout attributes and the structure elements to which they apply. The following sub-clauses describe the meaning and usage of these attributes.

As described in 14.8.5.3, "Attribute Values and Inheritance," an inheritable attribute may be specified for any element to propagate it to descendants, regardless of whether it is meaningful for that element.

Table 342 – Standard layout attributes

Structure Elements	Attributes	Inheritable
Any structure element	**Placement** **WritingMode** **BackgroundColor** **BorderColor** **BorderStyle** **BorderThickness** **Color** **Padding**	No Yes No Yes No Yes Yes No
Any BLSE ILSEs with **Placement** other than Inline	**SpaceBefore** **SpaceAfter** **StartIndent** **EndIndent**	No No Yes Yes
BLSEs containing text	**TextIndent** **TextAlign**	Yes Yes
Illustration elements (Figure, Formula, Form) Table	**BBox** **Width** **Height**	No No No
TH (Table header) TD (Table data)	**Width** **Height** **BlockAlign** **InlineAlign** **TBorderStyle** **TPadding**	No No Yes Yes Yes Yes

Table 342 – Standard layout attributes (continued)

Structure Elements	Attributes	Inheritable
Any ILSE BLSEs containing ILSEs or containing direct or nested content items	**LineHeight** **BaselineShift** **TextDecorationType** **TextDecorationColor** **TextDecorationThickness**	Yes No No Yes Yes
Grouping elements Art, Sect, and Div	**ColumnCount** **ColumnWidths** **ColumnGap**	No No No
Vertical text	**GlyphOrientationVertical**	Yes
Ruby text	**RubyAlign** **RubyPosition**	Yes Yes

14.8.5.4.2 General Layout Attributes

The layout attributes described in Table 343 may apply to structure elements of any of the standard types at the block level (BLSEs) or the inline level (ILSEs).

Table 343 – Standard layout attributes common to all standard structure types

Key	Type	Value
Placement	name	*(Optional; not inheritable)* The positioning of the element with respect to the enclosing reference area and other content: Block — Stacked in the block-progression direction within an enclosing reference area or parent BLSE. Inline — Packed in the inline-progression direction within an enclosing BLSE. Before — Placed so that the before edge of the element's allocation rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes") coincides with that of the nearest enclosing reference area. The element may float, if necessary, to achieve the specified placement. The element shall be treated as a block occupying the full extent of the enclosing reference area in the inline direction. Other content shall be stacked so as to begin at the after edge of the element's allocation rectangle. Start — Placed so that the start edge of the element's allocation rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes") coincides with that of the nearest enclosing reference area. The element may float, if necessary, to achieve the specified placement. Other content that would intrude into the element's allocation rectangle shall be laid out as a runaround. End — Placed so that the end edge of the element's allocation rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes") coincides with that of the nearest enclosing reference area. The element may float, if necessary, to achieve the specified placement. Other content that would intrude into the element's allocation rectangle shall be laid out as a runaround. When applied to an ILSE, any value except Inline shall cause the element to be treated as a BLSE instead. Default value: Inline. Elements with **Placement** values of Before, Start, or End shall be removed from the normal stacking or packing process and allowed to float to the specified edge of the enclosing reference area or parent BLSE. Multiple such floating elements may be positioned adjacent to one another against the specified edge of the reference area or placed serially against the edge, in the order encountered. Complex cases such as floating elements that interfere with each other or do not fit on the same page may be handled differently by different conforming readers. Tagged PDF merely identifies the elements as floating and indicates their desired placement.

Table 343 – Standard layout attributes common to all standard structure types (continued)

Key	Type	Value
WritingMode	name	*(Optional; inheritable)* The directions of layout progression for packing of ILSEs (inline progression) and stacking of BLSEs (block progression): LrTb Inline progression from left to right; block progression from top to bottom. This is the typical writing mode for Western writing systems. RlTb Inline progression from right to left; block progression from top to bottom. This is the typical writing mode for Arabic and Hebrew writing systems. TbRl Inline progression from top to bottom; block progression from right to left. This is the typical writing mode for Chinese and Japanese writing systems. The specified layout directions shall apply to the given structure element and all of its descendants to any level of nesting. Default value: LrTb. For elements that produce multiple columns, the writing mode defines the direction of column progression within the reference area: the inline direction determines the stacking direction for columns and the default flow order of text from column to column. For tables, the writing mode controls the layout of rows and columns: table rows (structure type TR) shall be stacked in the block direction, cells within a row (structure type TD) in the inline direction. The inline-progression direction specified by the writing mode is subject to local override within the text being laid out, as described in Unicode Standard Annex #9, The Bidirectional Algorithm, available from the Unicode Consortium (see the Bibliography).
BackgroundColor	array	*(Optional; not inheritable; PDF 1.5)* The colour to be used to fill the background of a table cell or any element's content rectangle (possibly adjusted by the **Padding** attribute). The value shall be an array of three numbers in the range 0.0 to 1.0, representing the red, green, and blue values, respectively, of an RGB colour space. If this attribute is not specified, the element shall be treated as if it were transparent.
BorderColor	array	*(Optional; inheritable; PDF 1.5)* The colour of the border drawn on the edges of a table cell or any element's content rectangle (possibly adjusted by the **Padding** attribute). The value of each edge shall be an array of three numbers in the range 0.0 to 1.0, representing the red, green, and blue values, respectively, of an RGB colour space. There are two forms: A single array of three numbers representing the RGB values to apply to all four edges. An array of four arrays, each specifying the RGB values for one edge of the border, in the order of the before, after, start, and end edges. A value of **null** for any of the edges means that it shall not be drawn. If this attribute is not specified, the border colour for this element shall be the current text fill colour in effect at the start of its associated content.

Table 343 – Standard layout attributes common to all standard structure types (continued)

Key	Type	Value
BorderStyle	array or name	*(Optional; not inheritable; PDF 1.5)* The style of an element's border. Specifies the stroke pattern of each edge of a table cell or any element's content rectangle (possibly adjusted by the **Padding** attribute). There are two forms: • A name from the list below representing the border style to apply to all four edges. • An array of four entries, each entry specifying the style for one edge of the border in the order of the before, after, start, and end edges. A value of **null** for any of the edges means that it shall not be drawn. None — No border. Forces the computed value of **BorderThickness** to be 0. Hidden — Same as None, except in terms of border conflict resolution for table elements. Dotted — The border is a series of dots. Dashed — The border is a series of short line segments. Solid — The border is a single line segment. Double — The border is two solid lines. The sum of the two lines and the space between them equals the value of **BorderThickness**. Groove — The border looks as though it were carved into the canvas. Ridge — The border looks as though it were coming out of the canvas (the opposite of Groove). Inset — The border makes the entire box look as though it were embedded in the canvas. Outset — The border makes the entire box look as though it were coming out of the canvas (the opposite of Inset). Default value: None All borders shall be drawn on top of the box's background. The colour of borders drawn for values of Groove, Ridge, Inset, and Outset shall depend on the structure element's **BorderColor** attribute and the colour of the background over which the border is being drawn. NOTE Conforming HTML applications may interpret Dotted, Dashed, Double, Groove, Ridge, Inset, and Outset to be Solid.
BorderThickness	number or array	*(Optional; inheritable; PDF 1.5)* The thickness of the border drawn on the edges of a table cell or any element's content rectangle (possibly adjusted by the **Padding** attribute). The value of each edge shall be a positive number in default user space units representing the border's thickness (a value of 0 indicates that the border shall not be drawn). There are two forms: A number representing the border thickness for all four edges. An array of four entries, each entry specifying the thickness for one edge of the border, in the order of the before, after, start, and end edges. A value of **null** for any of the edges means that it shall not be drawn.

Table 343 – Standard layout attributes common to all standard structure types (continued)

Key	Type	Value
Padding	number or array	*(Optional; not inheritable; PDF 1.5)* Specifies an offset to account for the separation between the element's content rectangle and the surrounding border (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes"). A positive value enlarges the background area; a negative value trims it, possibly allowing the border to overlap the element's text or graphic. The value shall be either a single number representing the width of the padding, in default user space units, that applies to all four sides or a 4-element array of numbers representing the padding width for the before, after, start, and end edge, respectively, of the content rectangle. Default value: 0.
Color	array	*(Optional; inheritable; PDF 1.5)* The colour to be used for drawing text and the default value for the colour of table borders and text decorations. The value shall be an array of three numbers in the range 0.0 to 1.0, representing the red, green, and blue values, respectively, of an RGB colour space. If this attribute is not specified, the border colour for this element shall be the current text fill colour in effect at the start of its associated content.

14.8.5.4.3 Layout Attributes for BLSEs

Table 344 describes layout attributes that shall apply only to block-level structure elements (BLSEs).

Inline-level structure elements (ILSEs) with a **Placement** attribute other than the default value of *Inline* shall be treated as BLSEs and shall also be subject to the attributes described here.

Table 344 – Additional standard layout attributes specific to block-level structure elements

Key	Type	Value
SpaceBefore	number	*(Optional; not inheritable)* The amount of extra space preceding the before edge of the BLSE, measured in default user space units in the block-progression direction. This value shall be added to any adjustments induced by the **LineHeight** attributes of ILSEs within the first line of the BLSE (see "Layout Attributes for ILSEs" in 14.8.5.4, "Layout Attributes"). If the preceding BLSE has a **SpaceAfter** attribute, the greater of the two attribute values shall be used. Default value: 0. This attribute shall be disregarded for the first BLSE placed in a given reference area.
SpaceAfter	number	*(Optional; not inheritable)* The amount of extra space following the after edge of the BLSE, measured in default user space units in the block-progression direction. This value shall be added to any adjustments induced by the **LineHeight** attributes of ILSEs within the last line of the BLSE (see 14.8.5.4, "Layout Attributes"). If the following BLSE has a **SpaceBefore** attribute, the greater of the two attribute values shall be used. Default value: 0. This attribute shall be disregarded for the last BLSE placed in a given reference area.

Table 344 – Additional standard layout attributes specific to block-level structure elements (continued)

Key	Type	Value
StartIndent	number	*(Optional; inheritable)* The distance from the start edge of the reference area to that of the BLSE, measured in default user space units in the inline-progression direction. This attribute shall apply only to structure elements with a **Placement** attribute of Block or Start (see “General Layout Attributes” in 14.8.5.4, “Layout Attributes”). The attribute shall be disregarded for elements with other **Placement** values. Default value: 0. A negative value for this attribute places the start edge of the BLSE outside that of the reference area. The results are implementation-dependent and may not be supported by all conforming products that process Tagged PDF or by particular export formats. If a structure element with a **StartIndent** attribute is placed adjacent to a floating element with a **Placement** attribute of Start, the actual value used for the element’s starting indent shall be its own **StartIndent** attribute or the inline extent of the adjacent floating element, whichever is greater. This value may be further adjusted by the element’s **TextIndent** attribute, if any.
EndIndent	number	*(Optional; inheritable)* The distance from the end edge of the BLSE to that of the reference area, measured in default user space units in the inline-progression direction. This attribute shall apply only to structure elements with a **Placement** attribute of Block or End (see “General Layout Attributes” in 14.8.5.4, “Layout Attributes”). The attribute shall be disregarded for elements with other **Placement** values. Default value: 0. A negative value for this attribute places the end edge of the BLSE outside that of the reference area. The results are implementation-dependent and may not be supported by all conforming products that process Tagged PDF or by particular export formats. If a structure element with an **EndIndent** attribute is placed adjacent to a floating element with a **Placement** attribute of End, the actual value used for the element’s ending indent shall be its own **EndIndent** attribute or the inline extent of the adjacent floating element, whichever is greater.
TextIndent	number	*(Optional; inheritable; applies only to some BLSEs)* The additional distance, measured in default user space units in the inline-progression direction, from the start edge of the BLSE, as specified by **StartIndent**, to that of the first line of text. A negative value shall indicate a hanging indent. Default value: 0. This attribute shall apply only to paragraphlike BLSEs and those of structure types Lbl (Label), LBody (List body), TH (Table header), and TD (Table data), provided that they contain content other than nested BLSEs.
TextAlign	name	*(Optional; inheritable; applies only to BLSEs containing text)* The alignment, in the inline-progression direction, of text and other content within lines of the BLSE: Start Aligned with the start edge. Center Centered between the start and end edges. End Aligned with the end edge. Justify Aligned with both the start and end edges, with internal spacing within each line expanded, if necessary, to achieve such alignment. The last (or only) line shall be aligned with the start edge only. Default value: Start.

Table 344 – Additional standard layout attributes specific to block-level structure elements (continued)

Key	Type	Value
BBox	rectangle	*(Optional for Annot; required for any figure or table appearing in its entirety on a single page; not inheritable)* An array of four numbers in default user space units that shall give the coordinates of the left, bottom, right, and top edges, respectively, of the element's bounding box (the rectangle that completely encloses its visible content). This attribute shall apply to any element that lies on a single page and occupies a single rectangle.
Width	number or name	*(Optional; not inheritable; illustrations, tables, table headers, and table cells only; should be used for table cells)* The width of the element's content rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes"), measured in default user space units in the inline-progression direction. This attribute shall apply only to elements of structure type Figure, Formula, Form, Table, TH (Table header), or TD (Table data). The name Auto in place of a numeric value shall indicate that no specific width constraint is to be imposed; the element's width shall be determined by the intrinsic width of its content. Default value: Auto.
Height	number or name	*(Optional; not inheritable; illustrations, tables, table headers, and table cells only)* The height of the element's content rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes"), measured in default user space units in the block-progression direction. This attribute shall apply only to elements of structure type Figure, Formula, Form, Table, TH (Table header), or TD (Table data). The name Auto in place of a numeric value shall indicate that no specific height constraint is to be imposed; the element's height shall be determined by the intrinsic height of its content. Default value: Auto.
BlockAlign	name	*(Optional; inheritable; table cells only)* The alignment, in the block-progression direction, of content within the table cell: Before — Before edge of the first child's allocation rectangle aligned with that of the table cell's content rectangle. Middle — Children centered within the table cell. The distance between the before edge of the first child's allocation rectangle and that of the table cell's content rectangle shall be the same as the distance between the after edge of the last child's allocation rectangle and that of the table cell's content rectangle. After — After edge of the last child's allocation rectangle aligned with that of the table cell's content rectangle. Justify — Children aligned with both the before and after edges of the table cell's content rectangle. The first child shall be placed as described for Before and the last child as described for After, with equal spacing between the children. If there is only one child, it shall be aligned with the before edge only, as for Before. This attribute shall apply only to elements of structure type TH (Table header) or TD (Table data) and shall control the placement of all BLSEs that are children of the given element. The table cell's content rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes") shall become the reference area for all of its descendants. Default value: Before.

Table 344 – Additional standard layout attributes specific to block-level structure elements (continued)

Key	Type	Value
InlineAlign	name	*(Optional; inheritable; table cells only)* The alignment, in the inline-progression direction, of content within the table cell: Start Start edge of each child's allocation rectangle aligned with that of the table cell's content rectangle. Center Each child centered within the table cell. The distance between the start edges of the child's allocation rectangle and the table cell's content rectangle shall be the same as the distance between their end edges. End End edge of each child's allocation rectangle aligned with that of the table cell's content rectangle. This attribute shall apply only to elements of structure type TH (Table header) or TD (Table data) and controls the placement of all BLSEs that are children of the given element. The table cell's content rectangle (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes") shall become the reference area for all of its descendants. Default value: Start.
TBorderStyle	name or array	*(Optional; inheritable; PDF 1.5)* The style of the border drawn on each edge of a table cell. Allowed values shall be the same as those specified for **BorderStyle** (see Table 343). If both **TBorderStyle** and **BorderStyle** apply to a given table cell, **BorderStyle** shall supersede **TBorderStyle**. Default value: None.
TPadding	integer or array	*(Optional; inheritable; PDF 1.5)* Specifies an offset to account for the separation between the table cell's content rectangle and the surrounding border (see "Content and Allocation Rectangles" in 14.8.5.4, "Layout Attributes"). If both **TPadding** and **Padding** apply to a given table cell, **Padding** shall supersede **TPadding**. A positive value shall enlarge the background area; a negative value shall trim it, and the border may overlap the element's text or graphic. The value shall be either a single number representing the width of the padding, in default user space units, that applies to all four edges of the table cell or a 4-entry array representing the padding width for the before edge, after edge, start edge, and end edge, respectively, of the content rectangle. Default value: 0.

14.8.5.4.4 Layout Attributes for ILSEs

The attributes described in Table 345 apply to inline-level structure elements (ILSEs). They may also be specified for a block-level element (BLSE) and may apply to any content items that are its immediate children.

Table 345 – Standard layout attributes specific to inline-level structure elements

Key	Type	Value
BaselineShift	number	*(Optional; not inheritable)* The distance, in default user space units, by which the element's baseline shall be shifted relative to that of its parent element. The shift direction shall be the opposite of the block-progression direction specified by the prevailing **WritingMode** attribute (see "General Layout Attributes" in 14.8.5.4, "Layout Attributes"). Thus, positive values shall shift the baseline toward the before edge and negative values toward the after edge of the reference area (upward and downward, respectively, in Western writing systems). Default value: 0. The shifted element may be a superscript, a subscript, or an inline graphic. The shift shall apply to the element, its content, and all of its descendants. Any further baseline shift applied to a child of this element shall be measured relative to the shifted baseline of this (parent) element.
LineHeight	number or name	*(Optional; inheritable)* The element's preferred height, measured in default user space units in the block-progression direction. The height of a line shall be determined by the largest **LineHeight** value for any complete or partial ILSE that it contains. The name Normal or Auto in place of a numeric value shall indicate that no specific height constraint is to be imposed. The element's height shall be set to a reasonable value based on the content's font size: Normal Adjust the line height to include any nonzero value specified for **BaselineShift**. Auto Adjustment for the value of **BaselineShift** shall not be made. Default value: Normal. This attribute applies to all ILSEs (including implicit ones) that are children of this element or of its nested ILSEs, if any. It shall not apply to nested BLSEs. When translating to a specific export format, the values Normal and Auto, if specified, shall be used directly if they are available in the target format. The meaning of the term "reasonable value" is left to the conforming reader to determine. It should be approximately 1.2 times the font size, but this value can vary depending on the export format. NOTE 1 In the absence of a numeric value for **LineHeight** or an explicit value for the font size, a reasonable method of calculating the line height from the information in a Tagged PDF file is to find the difference between the associated font's **Ascent** and **Descent** values (see 9.8, "Font Descriptors"), map it from glyph space to default user space (see 9.4.4, "Text Space Details"), and use the maximum resulting value for any character in the line.
TextDecorationColor	array	*(Optional; inheritable; PDF 1.5)* The colour to be used for drawing text decorations. The value shall be an array of three numbers in the range 0.0 to 1.0, representing the red, green, and blue values, respectively, of an RGB colour space. If this attribute is not specified, the border colour for this element shall be the current fill colour in effect at the start of its associated content.

Table 345 – Standard layout attributes specific to inline-level structure elements (continued)

Key	Type	Value
TextDecorationThickness	number	*(Optional; inheritable; PDF 1.5)* The thickness of each line drawn as part of the text decoration. The value shall be a non-negative number in default user space units representing the thickness (0 is interpreted as the thinnest possible line). If this attribute is not specified, it shall be derived from the current stroke thickness in effect at the start of the element's associated content, transformed into default user space units.
TextDecorationType	name	*(Optional; not inheritable)* The text decoration, if any, to be applied to the element's text. None — No text decoration Underline — A line below the text Overline — A line above the text LineThrough — A line through the middle of the text Default value: None. This attribute shall apply to all text content items that are children of this element or of its nested ILSEs, if any. The attribute shall not apply to nested BLSEs or to content items other than text. The colour, position, and thickness of the decoration shall be uniform across all children, regardless of changes in colour, font size, or other variations in the content's text characteristics.
RubyAlign	name	*(Optional; inheritable; PDF 1.5)* The justification of the lines within a ruby assembly: Start — The content shall be aligned on the start edge in the inline-progression direction. Center — The content shall be centered in the inline-progression direction. End — The content shall be aligned on the end edge in the inline-progression direction. Justify — The content shall be expanded to fill the available width in the inline-progression direction. Distribute — The content shall be expanded to fill the available width in the inline-progression direction. However, space shall also be inserted at the start edge and end edge of the text. The spacing shall be distributed using a 1:2:1 (start:infix:end) ratio. It shall be changed to a 0:1:1 ratio if the ruby appears at the start of a text line or to a 1:1:0 ratio if the ruby appears at the end of the text line. Default value: Distribute. This attribute may be specified on the RB and RT elements. When a ruby is formatted, the attribute shall be applied to the shorter line of these two elements. (If the RT element has a shorter width than the RB element, the RT element shall be aligned as specified in its **RubyAlign** attribute.)

Table 345 – Standard layout attributes specific to inline-level structure elements (continued)

Key	Type	Value
RubyPosition	name	*(Optional; inheritable; PDF 1.5)* The placement of the RT structure element relative to the RB element in a ruby assembly: Before The RT content shall be aligned along the before edge of the element. After The RT content shall be aligned along the after edge of the element. Warichu The RT and associated RP elements shall be formatted as a warichu, following the RB element. Inline The RT and associated RP elements shall be formatted as a parenthesis comment, following the RB element. Default value: Before.
GlyphOrientationVertical	name	*(Optional; inheritable; PDF 1.5)* Specifies the orientation of glyphs when the inline-progression direction is top to bottom or bottom to top. This attribute may take one of the following values: angle A number representing the clockwise rotation in degrees of the top of the glyphs relative to the top of the reference area. Shall be a multiple of 90 degrees between -180 and +360. AutoSpecifies a default orientation for text, depending on whether it is fullwidth (as wide as it is high). Fullwidth Latin and fullwidth ideographic text (excluding ideographic punctuation) shall be set with an angle of 0. Ideographic punctuation and other ideographic characters having alternate horizontal and vertical forms shall use the vertical form of the glyph. Non-fullwidth text shall be set with an angle of 90. Default value: Auto. NOTE 2 This attribute is used most commonly to differentiate between the preferred orientation of alphabetic (non-ideographic) text in vertically written Japanese documents (Auto or 90) and the orientation of the ideographic characters and/or alphabetic (non-ideographic) text in western signage and advertising (90). This attribute shall affect both the alignment and width of the glyphs. If a glyph is perpendicular to the vertical baseline, its horizontal alignment point shall be aligned with the alignment baseline for the script to which the glyph belongs. The width of the glyph area shall be determined from the horizontal width font characteristic for the glyph.

14.8.5.4.5 Content and Allocation Rectangles

As defined in 14.8.3, “Basic Layout Model,” an element’s *content rectangle* is an enclosing rectangle derived from the shape of the element’s content, which shall define the bounds used for the layout of any included child elements. The *allocation rectangle* includes any additional borders or spacing surrounding the element, affecting how it shall be positioned with respect to adjacent elements and the enclosing content rectangle or reference area.

The exact definition of the content rectangle shall depend on the element’s structure type:

- For a table cell (structure type TH or TD), the content rectangle shall be determined from the bounding box of all graphics objects in the cell’s content, taking into account any explicit bounding boxes (such as the **BBox** entry in a form XObject). This implied size may be explicitly overridden by the cell’s **Width** and **Height** attributes. The cell’s height shall be adjusted to equal the maximum height of any cell in its row; its width shall be adjusted to the maximum width of any cell in its column.

- For any other BLSE, the height of the content rectangle shall be the sum of the heights of all BLSEs it contains, plus any additional spacing adjustments between these elements.
- For an ILSE that contains text, the height of the content rectangle shall be set by the **LineHeight** attribute. The width shall be determined by summing the widths of the contained characters, adjusted for any indents, letter spacing, word spacing, or line-end conditions.
- For an ILSE that contains an illustration or table, the content rectangle shall be determined from the bounding box of all graphics objects in the content, and shall take into account any explicit bounding boxes (such as the **BBox** entry in a form XObject). This implied size may be explicitly overridden by the element's **Width** and **Height** attributes.
- For an ILSE that contains a mixture of elements, the height of the content rectangle shall be determined by aligning the child objects relative to one another based on their text baseline (for text ILSEs) or end edge (for non-text ILSEs), along with any applicable **BaselineShift** attribute (for all ILSEs), and finding the extreme top and bottom for all elements.

NOTE Some conforming readers may apply this process to all elements within the block; others may apply it on a line-by-line basis.

The allocation rectangle shall be derived from the content rectangle in a way that also depends on the structure type:

- For a BLSE, the allocation rectangle shall be equal to the content rectangle with its before and after edges adjusted by the element's **SpaceBefore** and **SpaceAfter** attributes, if any, but with no changes to the start and end edges.
- For an ILSE, the allocation rectangle is the same as the content rectangle.

14.8.5.4.6 Illustration Attributes

Particular uses of illustration elements (structure types Figure, Formula, or Form) shall have additional restrictions:

- When an illustration element has a **Placement** attribute of Block, it shall have a **Height** attribute with an explicitly specified numerical value (not Auto). This value shall be the sole source of information about the illustration's extent in the block-progression direction.
- When an illustration element has a **Placement** attribute of Inline, it shall have a **Width** attribute with an explicitly specified numerical value (not Auto). This value shall be the sole source of information about the illustration's extent in the inline-progression direction.
- When an illustration element has a **Placement** attribute of Inline, Start, or End, the value of its **BaselineShift** attribute shall be used to determine the position of its after edge relative to the text baseline; **BaselineShift** shall be ignored for all other values of **Placement**. (An illustration element with a **Placement** value of Start may be used to create a dropped capital; one with a **Placement** value of Inline may be used to create a raised capital.)

14.8.5.4.7 Column Attributes

The attributes described in Table 346 shall be present for the grouping elements Art, Sect, and Div (see 14.8.4.2, "Grouping Elements"). They shall be used when the content in the grouping element is divided into columns.

Table 346 – Standard column attributes

Key	Type	Value
ColumnCount	integer	*(Optional; not inheritable; PDF 1.6)* The number of columns in the content of the grouping element. Default value: 1.
ColumnGap	number or array	*(Optional; not inheritable; PDF 1.6)* The desired space between adjacent columns, measured in default user space units in the inline-progression direction. If the value is a number, it specifies the space between all columns. If the value is an array, it should contain numbers, the first element specifying the space between the first and second columns, the second specifying the space between the second and third columns, and so on. If there are fewer than **ColumnCount** - 1 numbers, the last element shall specify all remaining spaces; if there are more than **ColumnCount** - 1 numbers, the excess array elements shall be ignored.
ColumnWidths	number or array	*(Optional; not inheritable; PDF 1.6)* The desired width of the columns, measured in default user space units in the inline-progression direction. If the value is a number, it specifies the width of all columns. If the value is an array, it shall contain numbers, representing the width of each column, in order. If there are fewer than **ColumnCount** numbers, the last element shall specify all remaining widths; if there are more than **ColumnCount** numbers, the excess array elements shall be ignored.

14.8.5.5 List Attribute

If present, the **ListNumbering** attribute, described in Table 347, shall appear in an L (List) element. It controls the interpretation of the Lbl (Label) elements within the list's LI (List item) elements (see "List Elements" in 14.8.4.3, "Block-Level Structure Elements"). This attribute may only be defined in attribute objects whose **O** (owner) entry has the value **List** or is one of the format-specific owner names listed in Table 341.

Table 347 – Standard list attribute

Key	Type	Value
ListNumbering	name	*(Optional; inheritable)* The numbering system used to generate the content of the Lbl (Label) elements in an autonumbered list, or the symbol used to identify each item in an unnumbered list. The value of the **ListNumbering** shall be one of the following, and shall be applied as described here. None: No autonumbering; Lbl elements (if present) contain arbitrary text not subject to any numbering scheme Disc: Solid circular bullet Circle: Open circular bullet Square: Solid square bullet Decimal: Decimal arabic numerals (1–9, 10–99, …) UpperRoman: Uppercase roman numerals (I, II, III, IV, …) LowerRoman: Lowercase roman numerals (i, ii, iii, iv, …) UpperAlpha: Uppercase letters (A, B, C, …) LowerAlpha: Lowercase letters (a, b, c, …) Default value: None. The alphabet used for UpperAlpha and LowerAlpha shall be determined by the prevailing **Lang** entry (see 14.9.2, "Natural Language Specification"). The set of possible values may be expanded as Unicode identifies additional numbering systems. A conforming reader shall ignore any value not listed in this table; it shall behave as though the value were None.

NOTE This attribute is used to allow a content extraction tool to autonumber a list. However, the *Lbl* elements within the table should nevertheless contain the resulting numbers explicitly, so that the document can be reflowed or printed without the need for autonumbering.

14.8.5.6 PrintField Attributes

(PDF 1.7) The attributes described in Table 348 identify the role of fields in non-interactive PDF forms. Such forms may have originally contained interactive fields such as text fields and radio buttons but were then converted into non-interactive PDF files, or they may have been designed to be printed out and filled in manually. Since the roles of the fields cannot be determined from interactive elements, the roles are defined using PrintField attributes.

NOTE PrintField attributes enable screen readers to identify page content that represents form fields (radio buttons, check boxes, push buttons, and text fields). These attributes enable the controls in print form fields to be represented in the logical structure tree and to be presented to assistive technology as if they were read-only interactive fields.

Table 348 – PrintField attributes

Key	Type	Value
Role	name	*(Optional; not inheritable)* The type of form field represented by this graphic. The value of Role shall be one of the following, and a conforming reader shall interpret its meaning as defined herein. **rb** Radio button **cb** Check box **pb** Push button **tv** Text-value field The **tv** role shall be used for interactive fields whose values have been converted to text in the non-interactive document. The text that is the value of the field shall be the content of the Form element (see Table 340). NOTE 1 Examples include text edit fields, numeric fields, password fields, digital signatures, and combo boxes. Default value: None specified.
checked	name	*(Optional; not inheritable)* The state of a radio button or check box field. The value shall be one of: **on**, **off** (default), or **neutral**. NOTE 2 The case (capitalization) used for this key does not conform to the same conventions used elsewhere in this standard.
Desc	text string	*(Optional; not inheritable)* The alternate name of the field. NOTE 3 Similar to the value supplied in the **TU** entry of the field dictionary for interactive fields (see Table 220).

14.8.5.7 Table Attributes

The value of the **O** (owner) entry of a Table attributes element shall be **Table** or one of the format-specific owner names listed in Table 341.

Table 349 – Standard table attributes

Key	Type	Value
RowSpan	integer	*(Optional; not inheritable)* The number of rows in the enclosing table that shall be spanned by the cell. The cell shall expand by adding rows in the block-progression direction specified by the table's **WritingMode** attribute. If this entry is absent, a conforming reader shall assume a value of 1. This entry shall only be used when the table cell has a structure type of **TH** or **TD** or one that is role mapped to structure type **TH** or **TD** (see Table 337).
ColSpan	integer	*(Optional; not inheritable)* The number of columns in the enclosing table that shall be spanned by the cell. The cell shall expand by adding columns in the inline-progression direction specified by the table's **WritingMode** attribute. If this entry is absent, a conforming reader shall assume a value of 1 This entry shall only be used when the table cell has a structure type of **TH** or **TD** or one that is role mapped to structure types **TH** or **TD** (see Table 337).
Headers	array	*(Optional; not inheritable; PDF 1.5)* An array of byte strings, where each string shall be the element identifier (see the **ID** entry in Table 323) for a **TH** structure element that shall be used as a header associated with this cell. This attribute may apply to header cells (**TH**) as well as data cells (**TD**) (see Table 337). Therefore, the headers associated with any cell shall be those in its **Headers** array plus those in the **Headers** array of any **TH** cells in that array, and so on recursively.
Scope	name	*(Optional; not inheritable; PDF 1.5)* A name whose value shall be one of the following: **Row**, **Column**, or **Both**. This attribute shall only be used when the structure type of the element is **TH.** (see Table 337). It shall reflect whether the header cell applies to the rest of the cells in the row that contains it, the column that contains it, or both the row and the column that contain it.
Summary	text string	*(Optional; not inheritable; PDF 1.7)* A summary of the table's purpose and structure. This entry shall only be used within **Table** structure elements (see Table 337). NOTE For use in non-visual rendering such as speech or braille

14.9 Accessibility Support

14.9.1 General

PDF includes several facilities in support of accessibility of documents to users with disabilities. In particular, many visually computer users with visual impairments use screen readers to read documents aloud. To enable proper vocalization, either through a screen reader or by some more direct invocation of a text-to-speech engine, PDF supports the following features:

- Specifying the natural language used for text in a PDF document—for example, as English or Spanish, or used to hide or reveal optional content (see 14.9.2, "Natural Language Specification")
- Providing textual descriptions for images or other items that do not translate naturally into text (14.9.3, "Alternate Descriptions"), or replacement text for content that does translate into text but is represented in a nonstandard way (such as with a ligature or illuminated character; see 14.9.4, "Replacement Text")
- Specifying the expansion of abbreviations or acronyms (Section 14.9.5, "Expansion of Abbreviations and Acronyms")

The core of this support lies in the ability to determine the logical order of content in a PDF document, independently of the content's appearance or layout, through logical structure and Tagged PDF, as described under 14.8.2.3, "Page Content Order." An accessibility application can extract the content of a document for presentation to users with disabilities by traversing the structure hierarchy and presenting the contents of each node. For this reason, conforming writers ensure that all information in a document is reachable by means of the structure hierarchy, and they should use the facilities described in this sub-clause.

NOTE 1 Text can be extracted from Tagged PDF documents and examined or reused for purposes other than accessibility; see 14.8, "Tagged PDF."

NOTE 2 Additional guidelines for accessibility support of content published on the Web can be found in the W3C document *Web Content Accessibility Guidelines* and the documents it points to (see the Bibliography).

14.9.2 Natural Language Specification

14.9.2.1 General

Natural language may be specified for text in a document or for optional content.

The natural language used for text in a document shall be determined in a hierarchical fashion, based on whether an optional **Lang** entry *(PDF 1.4)* is present in any of several possible locations. At the highest level, the document's default language (which applies to both text strings and text within content streams) may be specified by a **Lang** entry in the document catalogue (see 7.7.2, "Document Catalog"). Below this, the language may be specified for the following items:

- Structure elements of any type (see 14.7.2, "Structure Hierarchy"), through a **Lang** entry in the structure element dictionary.
- Marked-content sequences that are not in the structure hierarchy (see 14.6, "Marked Content"), through a **Lang** entry in a property list attached to the marked-content sequence with a Span tag.

NOTE 1 Although **Span** is also a standard structure type, as described under 14.8.4.4, "Inline-Level Structure Elements," its use here is entirely independent of logical structure.

NOTE 2 The natural language used for optional content allows content to be hidden or revealed, based on the **Lang** entry *(PDF 1.5)* in the **Language** dictionary of an optional content usage dictionary.

NOTE 3 The following sub-clauses provide details on the value of the **Lang** entry and the hierarchical manner in which the language for text in a document is determined.

Text strings encoded in Unicode may include an escape sequence or language tag indicating the language of the text and overriding the prevailing **Lang** entry (see 7.9.2.2, "Text String Type").

14.9.2.2 Language Identifiers

Certain language-related dictionary entries are text strings that specify *language identifiers*. Such text strings may appear as ***Lang*** entries in the following structures or dictionaries:

- Document catalogue, structure element dictionary, or property list
- Optional content usage dictionary's Language dictionary, the hierarchical issues described in 14.9.2.3, "Language Specification Hierarchy," shall not apply to this entry

A language identifier shall either be the empty text string, to indicate that the language is unknown, or a *Language-Tag* as defined in RFC 3066, *Tags for the Identification of Languages*.

Although language codes are commonly represented using lowercase letters and country codes are commonly represented using uppercase letters, all tags shall be treated as case insensitive.

14.9.2.3 Language Specification Hierarchy

The ***Lang*** entry in the document catalogue shall specify the default natural language for all text in the document. Language specifications may appear within structure elements, and they may appear within marked-content sequences that are not in the structure hierarchy. If present, such language specifications override the default.

Language specifications within the structure hierarchy apply in this order:

- A structure element's language specification. If a structure element does not have a **Lang** entry, the element shall inherit its language from any parent element that has one.
- Within a structure element, a language specification for a nested structure element or marked-content sequence

If only part of the page content is contained in the structure hierarchy, and the structured content is nested within nonstructured content for which a different language specification applies, the structure element's language specification shall take precedence.

A language identifier attached to a marked-content sequence with the Span tag specifies the language for all text in the sequence except for nested marked content that is contained in the structure hierarchy (in which case the structure element's language applies) and except where overridden by language specifications for other nested marked content.

NOTE Examples in this sub-clause illustrate the hierarchical manner in which the language for text in a document is determined.

EXAMPLE 1 This example shows how a language specified for the document as a whole could be overridden by one specified for a marked-content sequence within a page's content stream, independent of any logical structure. In this case, the **Lang** entry in the document catalogue (not shown) has the value en-US, meaning U.S. English, and it is overridden by the **Lang** property attached (with the Span tag) to the marked-content sequence Hasta la vista. The **Lang** property identifies the language for this marked content sequence with the value es-MX, meaning Mexican Spanish.

```
2  0  obj                              % Page object
    <<  /Type  /Page
        /Contents  3 0 R                   % Content stream
        …
    >>
endobj

3  0  obj                              % Page's content stream
    <<  /Length  …  >>
stream
    BT
        (See you later, or as Arnold would say, )  Tj
        /Span  << /Lang  (es-MX) >>        % Start of marked-content sequence
            BDC
                (Hasta la vista.)  Tj
            EMC                            % End of marked-content sequence
    ET
endstream
endobj
```

EXAMPLE 2 In the following example, the **Lang** entry in the structure element dictionary (specifying English) applies to the marked-content sequence having an **MCID** (marked-content identifier) value of 0 within the indicated page's content stream. However, nested within that marked-content sequence is another one in which the **Lang** property attached with the Span tag (specifying Spanish) overrides the structure element's language specification.

This example omits required **StructParents** entries in the objects used as content items (see 14.7.4.4, "Finding Structure Elements from Content Items").

```
1  0  obj                                  % Structure element
    <<  /Type  /StructElem
        /S  /P                                 % Structure type
        /P  …                                  % Parent in structure hierarchy
        /K  <<  /Type  /MCR
                /Pg  2 0 R                     % Page containing marked-content sequence
                /MCID  0                       % Marked-content identifier
            >>
        /Lang  (en-US)                         % Language specification for this element
    >>
endobj

2  0  obj                                  % Page object
    <<  /Type  /Page
        /Contents  3 0 R                       % Content stream
        …
    >>
endobj

3  0  obj                                  % Page's content stream
    <<  /Length  …  >>
stream
    BT
        /P  << /MCID  0 >>                     % Start of marked-content sequence
            BDC
                (See you later, or in Spanish you would say, )  Tj
                /Span  << /Lang  (es-MX) >>    % Start of nested marked-content sequence
                    BDC
                        (Hasta la vista.)  Tj
                    EMC                        % End of nested marked-content sequence
            EMC                                % End of marked-content sequence
    ET
endstream
endobj
```

EXAMPLE 3 The page’s content stream consists of a marked-content sequence that specifies Spanish as its language by means of the Span tag with a **Lang** property. Nested within it is content that is part of a structure element (indicated by the **MCID** entry in that property list), and the language specification that applies to the latter content is that of the structure element, English.

This example omits required StructParents entries in the objects used as content items (see 14.7.4.4, “Finding Structure Elements from Content Items”).

```
1  0  obj                              % Structure element
    <<  /Type  /StructElem
        /S  /P                             % Structure type
        /P  …                              % Parent in structure hierarchy
        /K  <<  /Type  /MCR
                /Pg  2 0 R                 % Page containing marked-content sequence
                /MCID  0                   % Marked-content identifier
            >>
        /Lang  (en-US)                     % Language specification for this element
    >>
endobj

2  0  obj                              % Page object
    <<  /Type  /Page
        /Contents  3 0 R                   % Content stream
        …
    >>
endobj

3  0  obj                              % Page's content stream
    <<  /Length  …  >>
stream
```

```
        /Span  << /Lang  (es-MX) >>              % Start of marked-content sequence
            BDC
                (Hasta la vista, )  Tj
                /P  << /MCID  0 >>               % Start of structured marked-content sequence,
                    BDC                          %     to which structure element's language applies
                        (as Arnold would say.)  Tj
                    EMC                          % End of structured marked-content sequence
            EMC                                  % End of marked-content sequence
    endstream
    endobj
```

14.9.2.4 Multi-language Text Arrays

A *multi-language text array (PDF 1.5)* allows multiple text strings to be specified, each in association with a language identifier. (See the **Alt** entry in Tables 274 and 277 for examples of its use.)

A multi-language text array shall contain pairs of strings. The first string in each pair shall be a language identifier (14.9.2.2, “Language Identifiers”). A language identifier shall not appear more than once in the array; any unrecognized language identifier shall be ignored. An empty string specifies default text that may be used when no suitable language identifier is found in the array. The second string is text associated with the language.

EXAMPLE [(en-US) (My vacation) (fr) (mes vacances) ( ) (default text)]

When a conforming reader searches a multi-language text array to find text for a given language, it shall look for an exact (though case-insensitive) match between the given language’s identifier and the language identifiers in the array. If no exact match is found, prefix matching shall be attempted in increasing array order: a match shall be declared if the given identifier is a leading, case-insensitive, substring of an identifier in the array, and the first post-substring character in the array identifier is a hyphen. For example, given identifier en matches array identifier en-US, but given identifier en-US matches neither en nor en-GB. If no exact or prefix match can be found, the default text (if any) should be used.

14.9.3 Alternate Descriptions

PDF documents may be enhanced by providing alternate descriptions for images, formulas, or other items that do not translate naturally into text.

NOTE 1 Alternate descriptions are human-readable text that could, for example, be vocalized by a text-to-speech engine for the benefit of users with visual impairments.

An alternate description may be specified for the following items:

- A structure element (see 14.7.2, “Structure Hierarchy”), through an **Alt** entry in the structure element dictionary
- *(PDF 1.5)* A marked-content sequence (see 14.6, “Marked Content”), through an **Alt** entry in a property list attached to the marked-content sequence with a Span tag.
- Any type of annotation (see 12.5, “Annotations”) that does not already have a text representation, through a **Contents** entry in the annotation dictionary

For annotation types that normally display text, the **Contents** entry of the annotation dictionary shall be used as the source for an alternate description. For annotation types that do not display text, a **Contents** entry *(PDF 1.4)* may be included to specify an alternate description. Sound annotations, which need no alternate description for the purpose of vocalization, may include a **Contents** entry specifying a description that may be displayed for the benefit of users with hearing impairments.

An alternate name may be specified for an interactive form field (see 12.7, “Interactive Forms”) which, if present, shall be used in place of the actual field name when a conforming reader identifies the field in a user-interface. This alternate name, if provided, shall be specified using the **TU** entry of the field dictionary.

NOTE 2 The **TU** entry is useful for vocalization purposes.

Alternate descriptions are text strings, which shall be encoded in either **PDFDocEncoding** or Unicode character encoding.

NOTE 3 As described in 7.9.2.2, “Text String Type,” Unicode defines an escape sequence for indicating the language of the text. This mechanism enables the alternate description to change from the language specified by the prevailing **Lang** entry (as described in the preceding sub-clause). Within alternate descriptions, Unicode escape sequences specifying language shall override the prevailing Lang entry.

When applied to structure elements, the alternate description text shall be considered to be a complete (or whole) word or phrase substitution for the current element. If each of two (or more) elements in a sequence have an **Alt** entry in their dictionaries, they shall be treated as if a word break is present between them. The same applies to consecutive marked-content sequences.

The **Alt** entry in property lists may be combined with other entries.

EXAMPLE This example shows the **Alt** entry combined with a **Lang** entry.

```
/Span << /Lang (en-us) /Alt (six-point star) >> BDC (A) Tj  EMC
```

14.9.4 Replacement Text

NOTE 1 Just as alternate descriptions can be provided for images and other items that do not translate naturally into text (as described in the preceding sub-clause), replacement text can be specified for content that does translate into text but that is represented in a nonstandard way. These nonstandard representations might include, for example, glyphs for ligatures or custom characters, or inline graphics corresponding to letters in an illuminated manuscript or to dropped capitals.

Replacement text may be specified for the following items:

- A structure element (see 14.7.2, “Structure Hierarchy”), by means of the optional **ActualText** entry *(PDF 1.4)* of the structure element dictionary.
- *(PDF 1.5)* A marked-content sequence (see 14.6, “Marked Content”), through an **ActualText** entry in a property list attached to the marked-content sequence with a Span tag.

The **ActualText** value shall be used as a replacement, not a description, for the content, providing text that is equivalent to what a person would see when viewing the content. The value of **ActualText** shall be considered to be a character substitution for the structure element or marked-content sequence. If each of two (or more) consecutive structure or marked-content sequences has an **ActualText** entry, they shall be treated as if no word break is present between them.

NOTE 2 The treatment of **ActualText** as a character replacement is different from the treatment of **Alt**, which is treated as a whole word or phrase substitution.

EXAMPLE This example shows the use of replacement text to indicate the correct character content in a case where hyphenation changes the spelling of a word (in German, up until recent spelling reforms, the word “Drucker” when hyphenated was rendered as “Druk-” and “ker”).

```
(Dru) Tj
/Span
    <</Actual Text (c) >>
    BDC
        (k-) Tj
    EMC
(ker) '
```

Like alternate descriptions (and other text strings), replacement text, if encoded in Unicode, may include an escape sequence for indicating the language of the text. Such a sequence shall override the prevailing **Lang** entry (see 7.9.2.2, "Text String Type").

14.9.5 Expansion of Abbreviations and Acronyms

The expansion of an abbreviation or acronym may be specified for the following items:

- Marked-content sequences, through an **E** property *(PDF 1.4)* in a property list attached to the sequence with a Span tag.

- Structure elements, through an **E** entry *(PDF 1.5)* in the structure element dictionary.

NOTE 1 Abbreviations and acronyms can pose a problem for text-to-speech engines. Sometimes the full pronunciation for an abbreviation can be divined without aid. For example, a dictionary search will probably reveal that "Blvd." is pronounced "boulevard" and that "Ave." is pronounced "avenue." However, some abbreviations are difficult to resolve, as in the sentence "Dr. Healwell works at 123 Industrial Dr.".

EXAMPLE

```
BT
    /Span << /E (Doctor) >>
        BDC
            (Dr. ) Tj
        EMC
    (Healwell works at 123 Industrial ) Tj
    /Span << /E (Drive) >>
        BDC
            (Dr.) Tj
        EMC
ET
```

The **E** value (a text string) shall be considered to be a word or phrase substitution for the tagged text and therefore shall be treated as if a word break separates it from any surrounding text.The expansion text, if encoded in Unicode, may include an escape sequence for indicating the language of the text (see 7.9.2.2, "Text String Type"). Such a sequence shall override the prevailing Lang entry.

NOTE 2 Some abbreviations or acronyms are conventionally not expanded into words. For the text "XYZ," for example, either no expansion should be supplied (leaving its pronunciation up to the text-to-speech engine) or, to be safe, the expansion "X Y Z" should be specified.

14.10 Web Capture

14.10.1 General

The information in the Web Capture data structures enables conforming products to perform the following operations:

- Save locally and preserve the visual appearance of material from the Web

- Retrieve additional material from the Web and add it to an existing PDF file

- Update or modify existing material previously captured from the Web

- Find source information for material captured from the Web, such as the URL (if any) from which it was captured

- Find all material in a PDF file that was generated from a given URL

- Find all material in a PDF file that matches a given digital identifier (MD5 hash)

The information needed to perform these operations shall be recorded in two data structures in the PDF file:

- The *Web Capture information dictionary*, which shall hold document-level information related to Web Capture.
- The *Web Capture content database*, which shall hold a complete registry of the source content resources retrieved by Web Capture and where it came from.

NOTE 3 The *Web Capture content database* enables the capturing process to avoid downloading material that is already present in the file.

14.10.2 Web Capture Information Dictionary

The optional **SpiderInfo** entry in the document catalogue (see 7.7.2, “Document Catalog”), if present, shall hold *Web Capture information dictionary*.

Table 350 – Entries in the Web Capture information dictionary

Key	Type	Value
V	number	*(Required)* The Web Capture version number. The version number shall be 1.0 in a conforming file. This value shall be a single real number, not a major and minor version number. EXAMPLE A version number of 1.2 would be considered greater than 1.15.
C	array	*(Optional)* An array of indirect references to Web Capture command dictionaries (see 14.10.5.3, “Command Dictionaries”) describing commands that were used in building the PDF file. The commands shall appear in the array in the order in which they were executed in building the file.

14.10.3 Content Database

14.10.3.1 General

When a PDF file, or part of a PDF file, is built from a content resource stored in another format, such as an HTML page, the resulting PDF file (or portion thereof) may contain content from more than the single content resources. Conversely, since many content formats do not have static pagination, a single content resource may give rise to multiple PDF pages.

To keep track of the correspondence between PDF content and the resources from which the content was derived, a PDF file may contain a *content database* that maps URLs and digital identifiers to PDF objects such as pages and XObjects.

NOTE 4 By looking up digital identifiers in the database, Web Capture can determine whether newly downloaded content is identical to content already retrieved from a different URL. Thus, it can perform optimizations such as storing only one copy of an image that is referenced by multiple HTML pages.

Web Capture’s content database shall be organized into *content sets*. Each content set shall be a dictionary holding information about a group of related PDF objects generated from the same source data. A content set shall have for the value of its **S** (subtype) entry either the value *SPS*, for a page set, or *SIS*, for an image set.

The mapping from a source content resource to a content set in a PDF document may be saved in the PDF file. The mapping may be an association from the resource's URL to the content set, stored in the PDF document's URLS name tree. The mapping may also be an association from a digital identifier (14.10.3.3, “Digital Identifiers”) generated from resource's data to the content set, stored in the PDF document's IDS name tree. Both associations may be present in the PDF file.

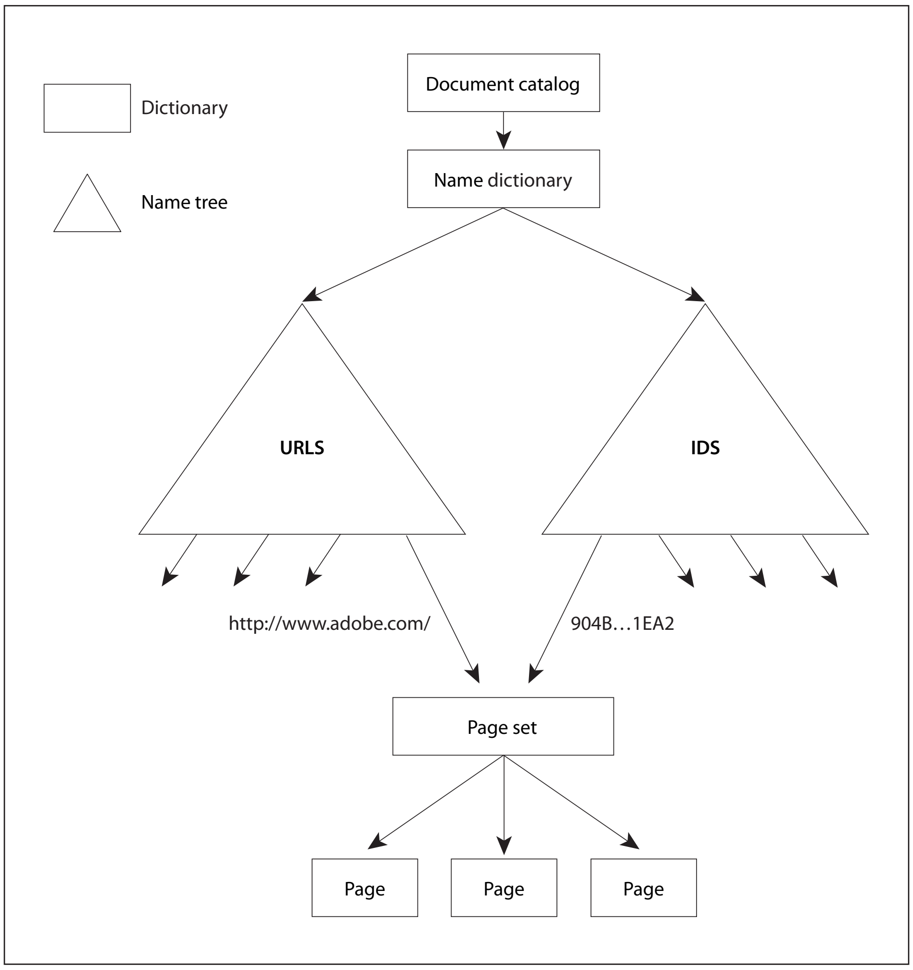


Figure 84 – Simple Web Capture file structure

Entries in the **URLS** and **IDS** name trees may refer to an array of content sets or a single content set. If the entry is an array, the content sets need not have the same subtype; the array may include both page sets and image sets.

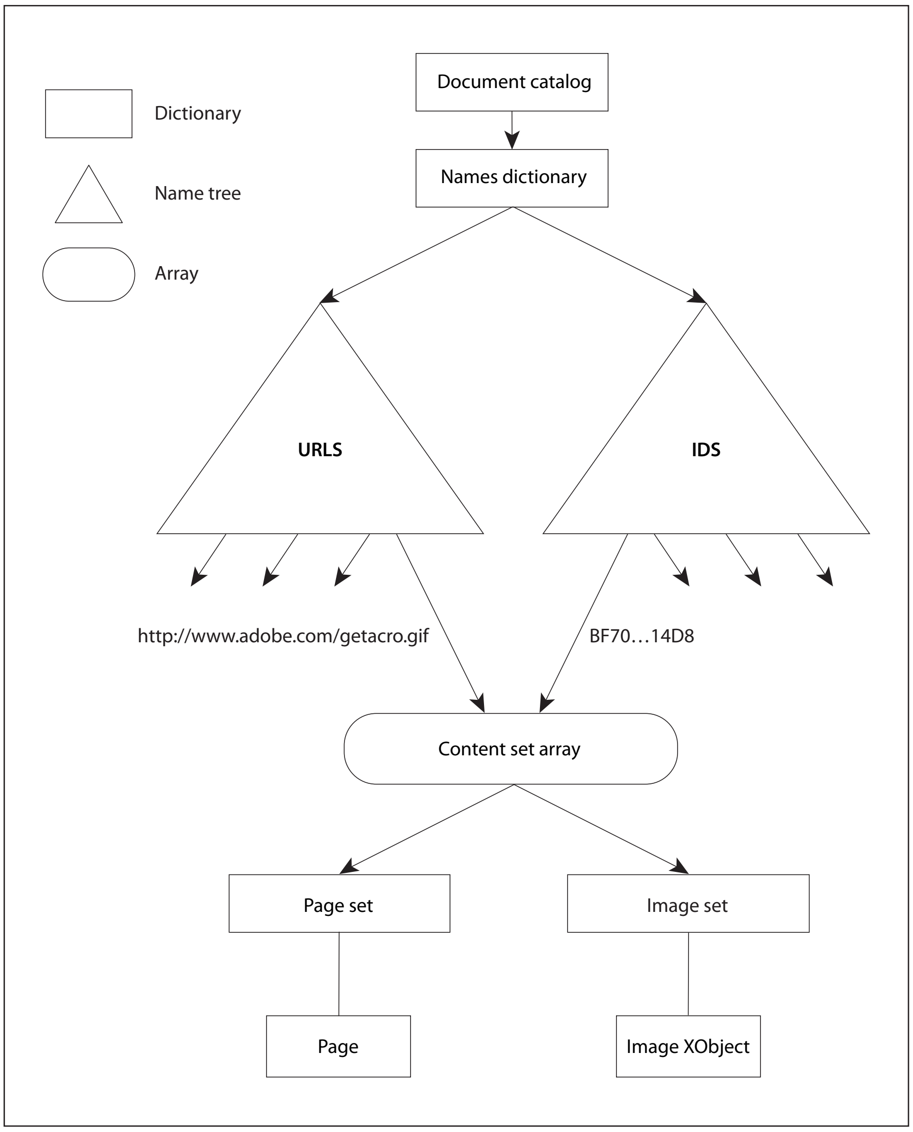


Figure 85 – Complex Web Capture file structure

14.10.3.2 URL Strings

URLs associated with Web Capture content sets shall be reduced to a predictable, canonical form before being used as keys in the **URLS** name tree. The following steps describe how to perform this reduction, using terminology from Internet RFCs 1738, *Uniform Resource Locators*, and 1808, *Relative Uniform Resource Locators* (see the Bibliography). This algorithm shall be applied for HTTP, FTP, and file URLs:

Algorithm: URL strings

a) If the URL is relative, it shall be converted into an absolute URL.

b) If the URL contains one or more NUMBER SIGN (02h3) characters, it shall be truncated before the first NUMBER SIGN.

c) Any uppercase ASCII characters within the scheme section of the URL shall be replaced with the corresponding lowercase ASCII characters.

d) If there is a host section, any uppercase ASCII characters therein shall be converted to lowercase ASCII.

e) If the scheme is file and the host is localhost, the host section shall be removed.

f) If there is a port section and the port is the default port for the given protocol (80 for HTTP or 21 for FTP), the port section shall be removed.

g) If the path section contains PERIOD (2Eh) (.) or DOUBLE PERIOD (..) subsequences, transform the path as described in section 4 of RFC 1808.

NOTE Because the PERCENT SIGN (25h) is unsafe according to RFC 1738 and is also the escape character for encoded characters, it is not possible in general to distinguish a URL with unencoded characters from one with encoded characters. For example, it is impossible to decide whether the sequence *%00* represents a single encoded null character or a sequence of three unencoded characters. Hence, no number of encoding or decoding passes on a URL can ever cause it to reach a stable state. Empirically, URLs embedded in HTML files have unsafe characters encoded with one encoding pass, and Web servers perform one decoding pass on received paths (though CGI scripts can make their own decisions).

Canonical URLs are thus assumed to have undergone one and only one encoding pass. A URL whose initial encoding state is known can be safely transformed into a URL that has undergone only one encoding pass.

14.10.3.3 Digital Identifiers

Digital identifiers, used to associate source content resources with content sets by the **IDS** name tree, shall be generated using the MD5 message-digest algorithm (Internet RFC 1321).

NOTE 1 The exact data passed to the algorithm depends on the type of content set and the nature of the identifier being calculated.

For a page set, the source data shall be passed to the MD5 algorithm first, followed by strings representing the digital identifiers of any auxiliary data files (such as images) referenced in the source data, in the order in which they are first referenced. If an auxiliary file is referenced more than once, its identifier shall be passed only the first time. The resulting string shall be used as the digital identifier for the source content resource.

NOTE 2 This sequence produces a composite identifier representing the visual appearance of the pages in the page set.

NOTE 3 Two HTML source files that are identical, but for which the referenced images contain different data—for example, if they have been generated by a script or are pointed to by relative URLs—do not produce the same identifier.

When the source data is a PDF file, the identifier shall be generated solely from the contents of that file; there shall be no auxiliary data.

A page set may also have a *text identifier*, calculated by applying the MD5 algorithm to just the text present in the source data.

EXAMPLE 1 For an HTML file the text identifier is based solely on the text between markup tags; no images are used in the calculation.

For an image set, the digital identifier shall be calculated by passing the source data for the original image to the MD5 algorithm.

EXAMPLE 2 The identifier for an image set created from a GIF image is calculated from the contents of the GIF.

14.10.3.4 Unique Name Generation

In generating PDF pages from a data source, items such as hypertext links and HTML form fields are converted into corresponding named destinations and interactive form fields. These items shall be given names that do not conflict with those of other such items in the file.

NOTE As used here, the term name refers to a string, not a name object.

Furthermore, when updating an existing file, a conforming processor shall ensure that each destination or field is given a unique name that shall be derived from its original name but constructed so that it avoids conflicts with similarly named items elsewhere.

The unique name shall be formed by appending an encoded form of the page set's digital identifier string to the original name of the destination or field. The identifier string shall be encoded to remove characters that have special meaning in destinations and fields. The characters listed in the first column of Table 351 have special meaning and shall be encoded using the corresponding byte values from second column of Table 351.

Table 351 – Characters with special meaning in destinations and fields and their byte values

Character	Byte value	Escape sequence
(nul)	0x00	\0 (0x5c 0x30)
. (PERIOD)	0x2e	\p (0x5c 0x70)
\ (backslash)	0x5c	\\ (0x5c 0x5c)

EXAMPLE Since the PERIOD character (2Eh) is used as the field separator in interactive form field names, it does not appear in the identifier portion of the unique name.

If the name is used for an interactive form field, there is an additional encoding to ensure uniqueness and compatibility with interactive forms. Each byte in the source string, encoded as described previously, shall be replaced by two bytes in the destination string. The first byte in each pair is 65 (corresponding to the ASCII character A) plus the high-order 4 bits of the source byte; the second byte is 65 plus the low-order 4 bits of the source byte.

14.10.4 Content Sets

14.10.4.1 General

A Web Capture *content set* is a dictionary describing a set of PDF objects generated from the same source data. It may include information common to all the objects in the set as well as about the set itself. Table 352 defines the contents of this type of dictionary.

14.10.4.2 Page Sets

A *page set* is a content set containing a group of PDF page objects generated from a common source, such as an HTML file. The pages shall be listed in the **O** array of the page set dictionary (see Table 352) in the same order in which they were initially added to the file. A single page object shall not belong to more than one page set. Table 353 defines the content set dictionary entries specific to Page Sets.

The **TID** (text identifier) entry may be used to store an identifier generated from the text of the pages belonging to the page set (see 14.10.3.3, "Digital Identifiers"). A text identifier may not be appropriate for some page sets (such as those with no text) and may be omitted in these cases.

EXAMPLE This identifier may be used to determine whether the text of a document has changed.

Table 352 – Entries common to all Web Capture content sets

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **SpiderContentSet** for a Web Capture content set.
S	name	*(Required)* The subtype of content set that this dictionary describes. The value shall be one of: **SPS** ("Spider page set") A page set **SIS** ("Spider image set") An image set

Table 352 – Entries common to all Web Capture content sets (continued)

Key	Type	Value
ID	byte string	*(Required)* The digital identifier of the content set (see 14.10.3.3, “Digital Identifiers”).
O	array	*(Required)* An array of indirect references to the objects belonging to the content set. The order of objects in the array is restricted when the content set subtype (**S** entry) is SPS (see 14.10.4.2, “Page Sets”).
SI	dictionary or array	*(Required)* A source information dictionary (see 14.10.5, “Source Information”) or an array of such dictionaries, describing the sources from which the objects belonging to the content set were created.
CT	ASCII string	*(Optional)* The content type, an ASCII string characterizing the source from which the objects belonging to the content set were created. The string shall conform to the content type specification described in Internet RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (see the Bibliography). EXAMPLE for a page set consisting of a group of PDF pages created from an HTML file, the content type would be text/html.
TS	date	*(Optional)* A time stamp giving the date and time at which the content set was created.

Table 353 – Additional entries specific to a Web Capture page set

Key	Type	Value
S	name	*(Required)* The subtype of content set that this dictionary describes; shall be **SPS**.
T	text string	*(Optional)* The title of the page set, a human-readable text string.
TID	byte string	*(Optional)* A text identifier generated from the text of the page set, as described in 14.10.3.3, “Digital Identifiers.”

14.10.4.3 Image Sets

An *image set* is a content set containing a group of image XObjects generated from a common source, such as multiple frames of an animated GIF image. A single XObject shall not belong to more than one image set. Table 354 shows the content set dictionary entries specific to Image Sets.

Table 354 – Additional entries specific to a Web Capture image set

Key	Type	Value
S	name	*(Required)* The subtype of content set that this dictionary describes; shall be **SIS**.
R	integer or array	*(Required)* The reference counts for the image XObjects belonging to the image set. For an image set containing a single XObject, the value shall be the integer reference count for that XObject. For an image set containing multiple XObjects, the value shall be an array of reference counts parallel to the **O** array (see Table 352); that is, each element in the **R** array shall hold the reference count for the image XObject at the corresponding position in the **O** array.

Each image XObject in an image set has a reference count indicating the number of PDF pages referring to that XObject. The reference count shall be incremented whenever Web Capture creates a new page referring to the XObject (including copies of already existing pages) and decremented whenever such a page is destroyed. The reference count shall be incremented or decremented only once per page, regardless of the number of times the XObject may be referenced by that page. If the reference count reaches 0, it shall be assumed that there are no remaining pages referring to the XObject and that the XObject can be removed from the image set's **O** array. When removing an XObject from the O array of an image set, the corresponding entry in the R array shall be removed also.

14.10.5 Source Information

14.10.5.1 General

The **SI** entry in a content set dictionary (see Table 352) shall contain one or more *source information dictionaries*, each containing information about the locations from which the source data for the content set was retrieved.

Table 355 – Entries in a source information dictionary

Key	Type	Value
AU	ASCII string or dictionary	*(Required)* An ASCII string or URL alias dictionary (see 14.10.5.2, "URL Alias Dictionaries") which shall identify the URLs from which the source data was retrieved.
TS	date	*(Optional)* A time stamp which, if present, shall contain the most recent date and time at which the content set's contents were known to be up to date with the source data.
E	date	*(Optional)* An expiration stamp which, if present, shall contain the date and time at which the content set's contents shall be considered out of date with the source data.
S	integer	*(Optional)* A code which, if present, shall indicate the type of form submission, if any, by which the source data was accessed (see 12.7.5.2, "Submit-Form Action"). If present, the value of the S entry shall be 0, 1, or 2, in accordance with the following meanings: 0 Not accessed by means of a form submission 1 Accessed by means of an HTTP GET request 2 Accessed by means of an HTTP POST request This entry may be present only in source information dictionaries associated with page sets. Default value: 0.
C	dictionary	*(Optional; if present, shall be an indirect reference)* A command dictionary (see 14.10.5.3, "Command Dictionaries") describing the command that caused the source data to be retrieved. This entry may be present only in source information dictionaries associated with page sets.

A content set's **SI** entry may contain a single source information dictionary. However, a PDF processor may attempt to detect situations in which the same source data has been located via two or more distinct URLs. If a processor detects such a situation, it may generate a single content set from the source data, containing a single copy of the relevant PDF pages or image XObjects. In this case, the **SI** entry shall be an array containing one source information dictionary for each distinct URL from which the original source content was found.

The determination that distinct URLs produce the same source data shall be made by comparing digital identifiers for the source data.

A source information dictionary's **AU** (aliased URLs) entry shall identify the URLs from which the source data was retrieved. If there is only one such URL, the *v* value of this entry may be a string. If multiple URLs map to the same location through redirection, the **AU** value shall be a URL alias dictionary (see 14.10.5.2, "URL Alias Dictionaries").

NOTE 1 For file size efficiency, the entire URL alias dictionary (excluding the URL strings) should be represented as a direct object because its internal structure should never be shared or externally referenced.

The **TS** (time stamp) entry allows each source location associated with a content set to have its own time stamp.

NOTE 2 This is necessary because the time stamp in the content set dictionary (see Table 352) merely refers to the creation date of the content set. A hypothetical "Update Content Set" command might reset the time stamp in the source information dictionary to the current time if it found that the source data had not changed since the time stamp was last set.

The **E** (expiration) entry specifies an expiration date for each source location associated with a content set. If the current date and time are later than those specified, the contents of the content set shall be considered out of date with respect to the original source.

14.10.5.2 URL Alias Dictionaries

When a URL is accessed via HTTP, a response header may be returned indicating that the requested data is at a different URL. This *redirection* process may be repeated in turn at the new URL and can potentially continue indefinitely. It is not uncommon to find multiple URLs that all lead eventually to the same destination through one or more redirections. A URL alias dictionary represents such a set of URL chains leading to a common destination. Table 356 shows the contents of this type of dictionary.

Table 356 – Entries in a URL alias dictionary

Key	Type	Value
U	ASCII string	*(Required)* The destination URL to which all of the chains specified by the **C** entry lead.
C	array	*(Optional)* An array of one or more arrays of strings, each representing a chain of URLs leading to the common destination specified by **U**.

The **C** (chains) entry may be omitted if the URL alias dictionary contains only one URL. If **C** is present, its value shall be an array of arrays, each representing a chain of URLs leading to the common destination. Within each chain, the URLs shall be stored as ASCII strings in the order in which they occur in the redirection sequence. The common destination (the last URL in a chain) may be omitted, since it is already identified by the **U** entry.

14.10.5.3 Command Dictionaries

A Web Capture *command dictionary* represents a command executed by Web Capture to retrieve one or more pieces of source data that were used to create new pages or modify existing pages. The entries in this dictionary represent parameters that were originally specified interactively by the user who requested that the Web content be captured. This information is recorded so that the command can subsequently be repeated to update the captured content. Table 357 shows the contents of this type of dictionary.

Table 357 – Entries in a Web Capture command dictionary

Key	Type	Value
URL	ASCII string	*(Required)* The initial URL from which source data was requested.
L	integer	*(Optional)* The number of levels of pages retrieved from the initial URL. Default value: 1.
F	integer	*(Optional)* A set of flags specifying various characteristics of the command (see Table 357). Defaut value: 0.
P	string or stream	*(Optional)* Data that was posted to the URL.
CT	ASCII string	*(Optional)* A content type describing the data posted to the URL. Default value: application/x-www-form-urlencoded.
H	string	*(Optional)* Additional HTTP request headers sent to the URL.

Table 357 – Entries in a Web Capture command dictionary (continued)

Key	Type	Value
S	dictionary	*(Optional)* A command settings dictionary containing settings used in the conversion process (see 14.10.5.4, "Command Settings").

The **URL** entry shall contain the initial URL for the retrieval command. The **L** (levels) entry shall contain the number of levels of the hyperlinked URL hierarchy to follow from this URL, creating PDF pages from the retrieved material. If the **L** entry is omitted, its value shall be assumed to be 1, denoting retrieval of the initial URL only.

The value of the command dictionary's **F** entry shall be an integer that shall be interpreted as an array of flags specifying various characteristics of the command. The flags shall be interpreted as defined in Table 358. Only those flags defined in Table 358 may be set to 1; all other flags shall be 0. Flags not defined in Table 358 are reserved for future use, and shall not be used by a conforming reader.

NOTE 3 The low-order bit of the flags value is referred to as being at bit-position 1.

Table 358 – Web Capture command flags

Bit position	Name	Meaning
1	SameSite	If set, pages were retrieved only from the host specified in the initial URL.
2	SamePath	If set, pages were retrieved only from the path specified in the initial URL.
3	Submit	If set, the command represents a form submission.

The SamePath flag shall be set if the retrieval of source content was restricted to source content in the same path as specified in the initial URL. Source content shall be considered to be in the same path if its scheme and network location components (as defined in Internet RFC 1808, *Relative Uniform Resource Locators*) match those of the initial URL and its path component matches up to and including the last forward slash (/) character in the initial URL.

EXAMPLE 1 the URL

http://www.adobe.com/fiddle/faddle/foo.html

is considered to be in the same path as the initial URL

http://www.adobe.com/fiddle/initial.html

The comparison shall be case-insensitive for the scheme and network location components and case-sensitive for the path component.

The Submit flag shall be set when the command represents a form submission. If no **P** (posted data) entry is present, the submitted data shall be encoded in the URL (an HTTP GET request). If **P** is present, the command shall be an HTTP POST request. In this case, the value of the Submit flag shall be ignored.

NOTE 4 If the posted data is small enough, it may be represented by a string. For large amounts of data, a stream should be used because it can be compressed.

The **CT** (content type) entry shall only be present for POST requests. It shall describe the content type of the posted data, as described in Internet RFC 2045, *Multipurpose Internet Mail Extensions (MIME), Part One: Format of Internet Message Bodies* (see the Bibliography).

The **H** (headers) entry, if present, shall specify additional HTTP request headers that were sent in the request for the URL. Each header line in the string shall be terminated with a CARRIAGE RETURN and a LINE FEED, as in this example:

EXAMPLE 2 (Referer: http://frumble.com\015\012From:veeble@frotz.com\015\012)

The HTTP request header format is specified in Internet RFC 2616, *Hypertext Transfer Protocol—HTTP/1.1* (see the Bibliography).

The **S** (settings) entry specifies a command settings dictionary (see 14.10.5.4, "Command Settings"). Holding settings specific to the conversion engines.

14.10.5.4 Command Settings

The **S** (settings) entry in a command dictionary, if present, shall contain a *command settings dictionary*, which holds settings for conversion engines that shall be used in converting the results of the command to PDF. Table 359 shows the contents of this type of dictionary. If this entry is omitted, default values are assumed. Command settings dictionaries may be shared by any command dictionaries that use the same settings.

Table 359 – Entries in a Web Capture command settings dictionary

Key	Type	Value
G	dictionary	*(Optional)* A dictionary containing global conversion engine settings relevant to all conversion engines. If this entry is absent, default settings shall be used.
C	dictionary	*(Optional)* Settings for specific conversion engines. Each key in this dictionary is the internal name of a conversion engine. The associated value is a dictionary containing the settings associated with that conversion engine. If the settings for a particular conversion engine are not found in the dictionary, default settings shall be used.

Each key in the **C** dictionary represents the internal name of a conversion engine, which shall be a name object of the following form:

/*company*:*product*:*version*:*contentType*

where

company denotes the name (or abbreviation) of the company that created the conversion engine.

product denotes the name of the conversion engine. This field may be left blank, but the trailing COLON character (3Ah) is still required.

version denotes the version of the conversion engine.

contentType denotes an identifier for the content type the associated settings. shall be used because some converters may handle multiple content types.

EXAMPLE /ADBE:H2PDF:1.0:HTML

All fields in the internal name are case-sensitive. The company field shall conform to the naming guidelines described in Annex E. The values of the other fields shall be unrestricted, except that they shall not contain a COLON.

The directed graph of PDF objects rooted by the command settings dictionary shall be entirely self-contained; that is, it shall not contain any object referred to from elsewhere in the PDF file.

NOTE This facilitates the operation of making a deep copy of a command settings dictionary without explicit knowledge of the settings it may contain.

14.10.6 Object Attributes Related to Web Capture

A given page object or image XObject may belong to at most one Web Capture content set, called its *parent content set*. However, the object shall not have direct pointer to its parent content set. Such a pointer may present problems for an application that traces all pointers from an object to determine what resources the object depends on. Instead, the object's **ID** entry (see Table 30 and Table 89) contains the digital identifier of the parent content set, which shall be used to locate the parent content set via the **IDS** name tree in the document's name dictionary. (If the **IDS** entry for the identifier contains an array of content sets, the parent may be found by searching the array for the content set whose **O** entry includes the child object.)

In the course of creating PDF pages from HTML files, Web Capture frequently scales the contents down to fit on fixed-sized pages. The **PZ** (preferred zoom) entry in a page object (see 7.7.3.3, "Page Objects") specifies a magnification factor by which the page may be scaled to undo the downscaling and view the page at its original size. That is, when the page is viewed at the preferred magnification factor, one unit in default user space corresponds to one original source pixel.

14.11 Prepress Support

14.11.1 General

This sub-clause describes features of PDF that support prepress production workflows:

- The specification of *page boundaries* governing various aspects of the prepress process, such as cropping, bleed, and trimming (14.11.2, "Page Boundaries")
- Facilities for including *printer's marks*, such as registration targets, gray ramps, colour bars, and cut marks to assist in the production process (14.11.3, "Printer's Marks")
- Information for generating *colour separations* for pages in a document (14.11.4, "Separation Dictionaries")
- *Output intents* for matching the colour characteristics of a document with those of a target output device or production environment in which it will be printed (14.11.5, "Output Intents")
- Support for the generation of *traps* to minimize the visual effects of misregistration between multiple colorants (14.11.6, "Trapping Support")
- The *Open Prepress Interface (OPI)* for creating low-resolution proxies for high-resolution images (14.11.7, "Open Prepress Interface (OPI)")

14.11.2 Page Boundaries

14.11.2.1 General

A PDF page may be prepared either for a finished medium, such as a sheet of paper, or as part of a prepress process in which the content of the page is placed on an intermediate medium, such as film or an imposed reproduction plate. In the latter case, it is important to distinguish between the intermediate page and the finished page. The intermediate page may often include additional production-related content, such as bleeds or printer marks, that falls outside the boundaries of the finished page. To handle such cases, a PDF page may define as many as five separate boundaries to control various aspects of the imaging process:

- The *media box* defines the boundaries of the physical medium on which the page is to be printed. It may include any extended area surrounding the finished page for bleed, printing marks, or other such purposes. It may also include areas close to the edges of the medium that cannot be marked because of physical limitations of the output device. Content falling outside this boundary may safely be discarded without affecting the meaning of the PDF file.

- The *crop box* defines the region to which the contents of the page shall be clipped (cropped) when displayed or printed. Unlike the other boxes, the crop box has no defined meaning in terms of physical page geometry or intended use; it merely imposes clipping on the page contents. However, in the absence of additional information (such as imposition instructions specified in a JDF or PJTF job ticket), the crop box determines how the page's contents shall be positioned on the output medium. The default value is the page's media box.

- The *bleed box (PDF 1.3)* defines the region to which the contents of the page shall be clipped when output in a production environment. This may include any extra bleed area needed to accommodate the physical limitations of cutting, folding, and trimming equipment. The actual printed page may include printing marks that fall outside the bleed box. The default value is the page's crop box.

- The *trim box (PDF 1.3)* defines the intended dimensions of the finished page after trimming. It may be smaller than the media box to allow for production-related content, such as printing instructions, cut marks, or colour bars. The default value is the page's crop box.

- The *art box (PDF 1.3)* defines the extent of the page's meaningful content (including potential white space) as intended by the page's creator. The default value is the page's crop box.

The page object dictionary specifies these boundaries in the **MediaBox**, **CropBox**, **BleedBox**, **TrimBox**, and **ArtBox** entries, respectively (see Table 30). All of them are rectangles expressed in default user space units. The crop, bleed, trim, and art boxes shall not ordinarily extend beyond the boundaries of the media box. If they do, they are effectively reduced to their intersection with the media box. Figure 86 illustrates the relationships among these boundaries. (The crop box is not shown in the figure because it has no defined relationship with any of the other boundaries.)

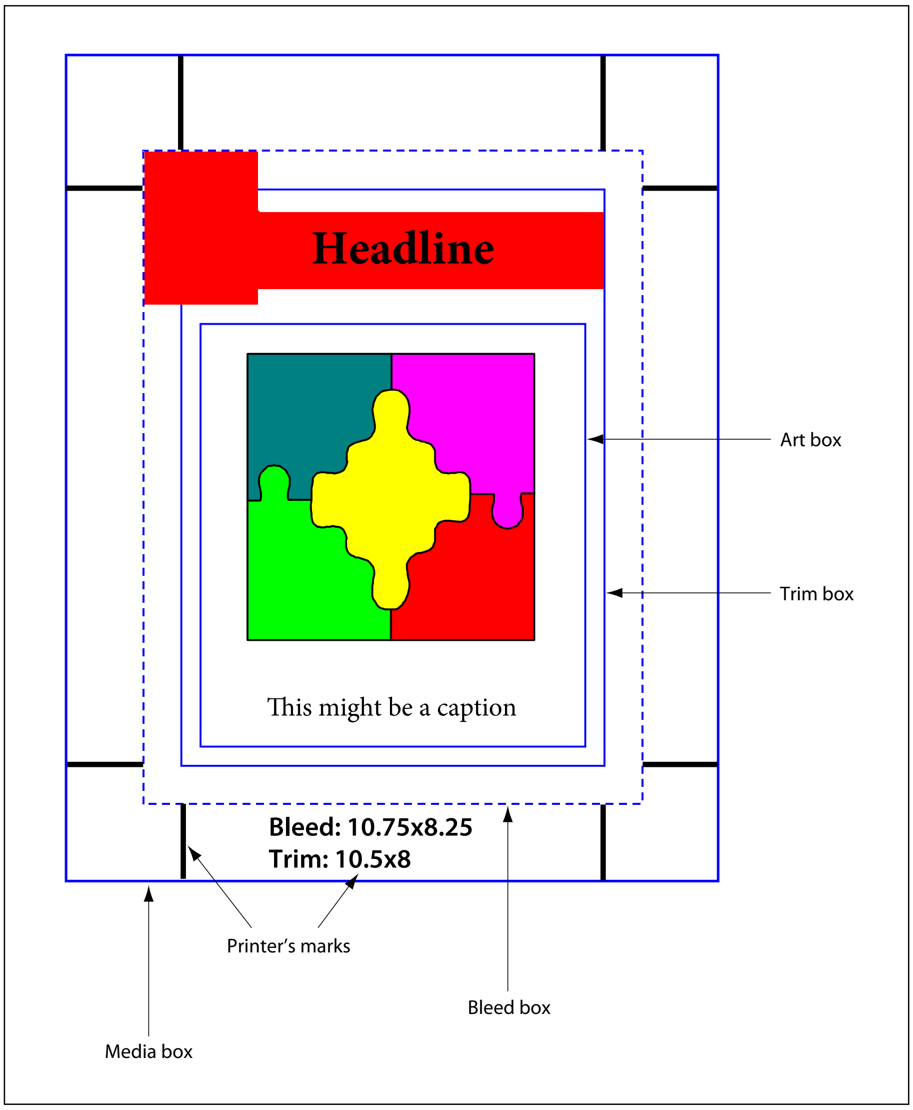


Figure 86 – Page boundaries

NOTE 1 How the various boundaries are used depends on the purpose to which the page is being put. The following are typical purposes:

Placing the content of a page in another application. The art box determines the boundary of the content that is to be placed in the application. Depending on the applicable usage conventions, the placed content may be clipped to either the art box or the bleed box. For example, a quarter-page advertisement to be placed on a magazine page might be clipped to the art box on the two sides of the ad that face into the middle of the page and to the bleed box on the two sides that bleed over the edge of the page. The media box and trim box are ignored.

Printing a finished page. This case is typical of desktop or shared page printers, in which the page content is positioned directly on the final output medium. The art box and bleed box are ignored. The media box may be used as advice for selecting media of the appropriate size. The crop box and trim box, if present, should be the same as the media box.

Printing an intermediate page for use in a prepress process. The art box is ignored. The bleed box defines the boundary of the content to be imaged. The trim box specifies the positioning of the content on the medium; it may also be used to generate cut or fold marks outside the bleed box. Content falling within the media box but outside the bleed box may or may not be imaged, depending on the specific production process being used.

Building an imposition of multiple pages on a press sheet. The art box is ignored. The bleed box defines the clipping boundary of the content to be imaged; content outside the bleed box is ignored. The trim box specifies the positioning of the page's content within the imposition. Cut and fold marks are typically generated for the imposition as a whole.

NOTE 2 In the preceding scenarios, an application that interprets the bleed, trim, and art boxes for some purpose typically alters the crop box so as to impose the clipping that those boxes prescribe.

14.11.2.2 Display of Page Boundaries

Conforming readers may offer the ability to display guidelines on the screen for the various page boundaries. The optional **BoxColorInfo** entry in a page object (see 7.7.3.3, "Page Objects") holds a *box colour information dictionary (PDF 1.4)* specifying the colours and other visual characteristics to be used for such display. Conforming readers typically provide a user interface to allow the user to set these characteristics interactively.

NOTE This information is page-specific and may vary from one page to another.

As shown in Table 360, the box colour information dictionary contains an optional entry for each of the possible page boundaries other than the media box. The value of each entry is a *box style dictionary*, whose contents are shown in Table 361. If a given entry is absent, the conforming reader shall use its own current default settings instead.

14.11.3 Printer's Marks

Printer's marks are graphic symbols or text added to a page to assist production personnel in identifying components of a multiple-plate job and maintaining consistent output during production. Examples commonly used in the printing industry include:

- Registration targets for aligning plates
- Gray ramps and colour bars for measuring colours and ink densities
- Cut marks showing where the output medium is to be trimmed

Although conforming writers traditionally include such marks in the content stream of a document, they are logically separate from the content of the page itself and typically appear outside the boundaries (the crop box, trim box, and art box) defining the extent of that content (see 14.11.2, "Page Boundaries").

Printer's mark annotations (PDF 1.4) provide a mechanism for incorporating printer's marks into the PDF representation of a page, while keeping them separate from the actual page content. Each page in a PDF document may contain any number of such annotations, each of which represents a single printer's mark.

NOTE 1 Because printer's marks typically fall outside the page's content boundaries, each mark is represented as a separate annotation. Otherwise—if, for example, the cut marks at the four corners of the page were defined in a single annotation—the annotation rectangle would encompass the entire contents of the page and could interfere with the user's ability to select content or interact with other annotations on the page. Defining printer's marks in separate annotations also facilitates the implementation of a drag-and-drop user interface for specifying them.

Table 360 – Entries in a box colour information dictionary

Key	Type	Value
CropBox	dictionary	*(Optional)* A box style dictionary (see Table 361) specifying the visual characteristics for displaying guidelines for the page's crop box. This entry shall be ignored if no crop box is defined in the page object.
BleedBox	dictionary	*(Optional)* A box style dictionary (see Table 361) specifying the visual characteristics for displaying guidelines for the page's bleed box. This entry shall be ignored if no bleed box is defined in the page object.

Table 360 – Entries in a box colour information dictionary (continued)

Key	Type	Value
TrimBox	dictionary	*(Optional)* A box style dictionary (see Table 361) specifying the visual characteristics for displaying guidelines for the page's trim box. This entry shall be ignored if no trim box is defined in the page object.
ArtBox	dictionary	*(Optional)* A box style dictionary (see Table 361) specifying the visual characteristics for displaying guidelines for the page's art box. This entry shall be ignored if no art box is defined in the page object.

Table 361 – Entries in a box style dictionary

Key	Type	Value
C	array	*(Optional)* An array of three numbers in the range 0.0 to 1.0, representing the components in the **DeviceRGB** colour space of the colour to be used for displaying the guidelines. Default value: [0.0 0.0 0.0].
W	number	*(Optional)* The guideline width in default user space units. Default value: 1.
S	name	*(Optional)* The guideline style: S (Solid) A solid rectangle. D (Dashed) A dashed rectangle. The dash pattern shall be specified by the **D** entry. Other guideline styles may be defined in the future. Default value: S.
D	array	*(Optional)* A dash array defining a pattern of dashes and gaps to be used in drawing dashed guidelines (guideline style D). The dash array shall be specified in default user space units, in the same format as in the line dash pattern parameter of the graphics state (see 8.4.3.6, "Line Dash Pattern"). The dash phase shall not be specified and shall be assumed to be 0. EXAMPLE A **D** entry of [3 2] specifies guidelines drawn with 3-point dashes alternating with 2-point gaps. Default value: [3].

The visual presentation of a printer's mark shall be defined by a form XObject specified as an appearance stream in the **N** (normal) entry of the printer's mark annotation's appearance dictionary (see 12.5.5, "Appearance Streams"). More than one appearance may be defined for the same printer's mark to meet the requirements of different regions or production facilities. In this case, the appearance dictionary's **N** entry holds a subdictionary containing the alternate appearances, each identified by an arbitrary key. The **AS** (appearance state) entry in the annotation dictionary designates one of them to be displayed or printed.

NOTE 2 The printer's mark annotation's appearance dictionary may include ***R*** (rollover) or ***D*** (down) entries, but appearances defined in either of these entries are never displayed or printed.

Like all annotations, a printer's mark annotation shall be defined by an annotation dictionary (see 12.5.2, "Annotation Dictionaries"); its annotation type is **PrinterMark**. The **AP** (appearances) and **F** (flags) entries (which ordinarily are optional) shall be present, as shall the **AS** (appearance state) entry if the appearance dictionary **AP** contains more than one appearance stream. The Print and ReadOnly flags in the **F** entry shall be set and all others clear (see 12.5.3, "Annotation Flags"). Table 362 shows an additional annotation dictionary entry specific to this type of annotation.

Table 362 – Additional entries specific to a printer’s mark annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **PrinterMark** for a printer’s mark annotation.
MN	name	*(Optional)* An arbitrary name identifying the type of printer’s mark, such as ColorBar or RegistrationTarget.

The form dictionary defining a printer’s mark may contain the optional entries shown in Table 363 in addition to the standard ones common to all form dictionaries (see 8.10.2, “Form Dictionaries”).

Table 363 – Additional entries specific to a printer’s mark form dictionary

Key	Type	Value
MarkStyle	text string	*(Optional; PDF 1.4)* A text string representing the printer’s mark in human-readable form and suitable for presentation to the user.
Colorants	dictionary	*(Optional; PDF 1.4)* A dictionary identifying the individual colorants associated with a printer’s mark, such as a colour bar. For each entry in this dictionary, the key is a colorant name and the value is an array defining a **Separation** colour space for that colorant (see 8.6.6.4, “Separation Colour Spaces”). The key shall match the colorant name given in that colour space.

14.11.4 Separation Dictionaries

In high-end printing workflows, pages are ultimately produced as sets of *separations*, one per colorant (see 8.6.6.4, “Separation Colour Spaces”). Ordinarily, each page in a PDF file shall be treated as a composite page that paints graphics objects using all the process colorants and perhaps some spot colorants as well. In other words, all separations for a page shall be generated from a single PDF description of that page.

In some workflows, however, pages are *preseparated* before generating the PDF file. In a preseparated PDF file, the separations for a page shall be described as separate page objects, each painting only a single colorant (usually specified in the **DeviceGray** colour space). In this case, additional information is needed to identify the actual colorant associated with each separation and to group together the page objects representing all the separations for a given page. This information shall be contained in a *separation dictionary (PDF 1.3)* in the **SeparationInfo** entry of each page object (see 7.7.3.3, “Page Objects”). Table 364 shows the contents of this type of dictionary.

Table 364 – Entries in a separation dictionary

Key	Type	Value
Pages	array	*(Required)* An array of indirect references to page objects representing separations of the same document page. One of the page objects in the array shall be the one with which this separation dictionary is associated, and all of them shall have separation dictionaries (**SeparationInfo** entries) containing **Pages** arrays identical to this one.
DeviceColorant	name or string	*(Required)* The name of the device colorant to be used in rendering this separation, such as Cyan or PANTONE 35 CV.

Table 364 – Entries in a separation dictionary (continued)

Key	Type	Value
ColorSpace	array	*(Optional)* An array defining a **Separation** or **DeviceN** colour space (see 8.6.6.4, "Separation Colour Spaces" and 8.6.6.5, "DeviceN Colour Spaces"). It provides additional information about the colour specified by **DeviceColorant**—in particular, the alternate colour space and tint transformation function that shall be used to represent the colorant as a process colour. This information enables a conforming reader to preview the separation in a colour that approximates the device colorant. The value of **DeviceColorant** shall match the space's colorant name (if it is a **Separation** space) or be one of the space's colorant names (if it is a **DeviceN** space).

14.11.5 Output Intents

Output intents (PDF 1.4) provide a means for matching the colour characteristics of a PDF document with those of a target output device or production environment in which the document will be printed. The optional **OutputIntents** entry in the document catalogue (see 7.7.2, "Document Catalog") holds an array of *output intent dictionaries*, each describing the colour reproduction characteristics of a possible output device or production condition. The contents of these dictionaries may vary for different devices and conditions. The dictionary's **S** entry specifies an *output intent subtype* that determines the format and meaning of the remaining entries.

NOTE 1 This use of multiple output intents allows the production process to be customized to the expected workflow and the specific tools available. For example, one production facility might process files conforming to a recognized standard such as PDF/X-1, while another uses the PDF/A standard to produce RGB output for document distribution on the Web. Each of these workflows would require different sets of output intent information. Multiple output intents also allow the same PDF file to be distributed unmodified to multiple production facilities. The choice of which output intent to use in a given production environment is a matter for agreement between the purchaser and provider of production services. PDF intentionally does not include a selector for choosing a particular output intent from within the PDF file.

At the time of publication, three output intent subtypes have been defined: **GTS_PDFX** corresponding to the PDF/X format standard specified in ISO 15930, **GTS_PDFA1** corresponding to the PDF/A standard as defined by ISO 19005, and ISO_PDFE1 corresponding to the PDF/E standard as defined by ISO 24517. Table 365 shows the contents of this type of output intent dictionary. Other subtypes may be added in the future; the names of any such additional subtypes shall conform to the naming guidelines described in Annex E.

Table 365 – Entries in an output intent dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **OutputIntent** for an output intent dictionary.
S	name	*(Required)* The output intent subtype; shall be either one of *GTS_PDFX*, *GTS_PDFA1*, *ISO_PDFE1* or a key defined by an ISO 32000 extension.
OutputCondition	text string	*(Optional)* A text string concisely identifying the intended output device or production condition in human-readable form. This is the preferred method of defining such a string for presentation to the user.

Table 365 – Entries in an output intent dictionary (continued)

Key	Type	Value
OutputConditionIdentifier	text string	*(Required)* A text string identifying the intended output device or production condition in human- or machine-readable form. If human-readable, this string may be used in lieu of an **OutputCondition** string for presentation to the user. A typical value for this entry may be the name of a production condition maintained in an industry-standard registry such as the ICC Characterization Data Registry (see the Bibliography). If the designated condition matches that in effect at production time, the production software is responsible for providing the corresponding ICC profile as defined in the registry. If the intended production condition is not a recognized standard, the value of this entry may be Custom or an application-specific, machine-readable name. The **DestOutputProfile** entry defines the ICC profile, and the **Info** entry shall be used for further human-readable identification.
RegistryName	text string	*(Optional)* An text string (conventionally a uniform resource identifier, or URI) identifying the registry in which the condition designated by **OutputConditionIdentifier** is defined.
Info	text string	*(Required if **OutputConditionIdentifier** does not specify a standard production condition; optional otherwise)* A human-readable text string containing additional information or comments about the intended target device or production condition.
DestOutputProfile	stream	*(Required if **OutputConditionIdentifier** does not specify a standard production condition; optional otherwise)* An ICC profile stream defining the transformation from the PDF document's source colours to output device colorants. The format of the profile stream is the same as that used in specifying an **ICCBased** colour space (see 8.6.5.5, "ICCBased Colour Spaces"). The output transformation uses the profile's "from CIE" information (BToA in ICC terminology); the "to CIE" (AToB) information may optionally be used to remap source colour values to some other destination colour space, such as for screen preview or hardcopy proofing.

NOTE 2 PDF/X is actually a family of standards representing varying levels of conformance. The standard for a given conformance level may prescribe further restrictions on the usage and meaning of entries in the output intent dictionary. Any such restrictions take precedence over the descriptions given in Table 364.

The ICC profile information in an output intent dictionary supplements rather than replaces that in an **ICCBased** or default colour space (see 8.6.5.5, "ICCBased Colour Spaces," and 8.6.5.6, "Default Colour Spaces"). Those mechanisms are specifically intended for describing the characteristics of source colour component values. An output intent can be used in conjunction with them to convert source colours to those required for a specific production condition or to enable the display or proofing of the intended output.

The data in an output intent dictionary shall be provided for informational purposes only, and conforming readers are free to disregard it. In particular, there is no expectation that PDF production tools automatically convert colours expressed in the same source colour space to the specified target space before generating output. (In some workflows, such conversion may, in fact, be undesirable).

NOTE When working with *CMYK* source colours tagged with a source ICC profile solely for purposes of characterization, converting such colours from four components to three and back is unnecessary and will result in a loss of fidelity in the values of the black component; see 8.6.5.7, "Implicit Conversion of CIE-Based Colour Spaces" for further discussion.) On the other hand, when source colours are expressed in different base colour spaces—for example, when combining separately generated images on the same PDF page—it is possible (though not required) to use the destination profile specified in the output intent dictionary to convert source colours to the same target colour space.

EXAMPLE 1 This Example shows a PDF/X output intent dictionary based on an industry-standard production condition (CGATS TR 001) from the *ICC Characterization Data Registry*. Example 2 shows one for a custom production condition.

```
<<  /Type  /OutputIntent                          % Output intent dictionary
    /S  /GTS_PDFX
    /OutputCondition  (CGATS  TR  001  (SWOP))
    /OutputConditionIdentifier  (CGATS  TR  001)
    /RegistryName  (http://www.color.org)
    /DestOutputProfile  100 0 R
>>

100  0  obj                                       % ICC profile stream
    <<  /N  4
        /Length  1605
        /Filter  /ASCIIHexDecode
    >>

stream
00  00  02  0C  61  70  …  >
endstream
endobj
```

EXAMPLE 2

```
<<   /Type  /OutputIntent                          % Output intent dictionary
    /S  /GTS_PDFX
    /OutputCondition  (Coated)
    /OutputConditionIdentifier  (Custom)
    /Info  (Coated 150lpi)
    /DestOutputProfile  100 0 R
>>

100  0  obj                                        % ICC profile stream
    <<  /N  4
        /Length  1605
        /Filter  /ASCIIHexDecode
    >>

stream
00  00  02  0C  61  70  …  >
endstream
endobj
```

14.11.6 Trapping Support

14.11.6.1 General

On devices such as offset printing presses, which mark multiple colorants on a single sheet of physical medium, mechanical limitations of the device can cause imprecise alignment, or *misregistration*, between colorants. This can produce unwanted visual artifacts such as brightly coloured gaps or bands around the edges of printed objects. In high-quality reproduction of colour documents, such artifacts are commonly avoided by creating an overlap, called a *trap*, between areas of adjacent colour.

NOTE Figure 87 shows an example of trapping. The light and medium grays represent two different colorants, which are used to paint the background and the glyph denoting the letter A. The first figure shows the intended result, with the two colorants properly registered. The second figure shows what happens when the colorants are misregistered. In the third figure, traps have been overprinted along the boundaries, obscuring the artifacts caused by the misregistration. (For emphasis, the traps are shown here in dark gray; in actual practice, their colour will be similar to one of the adjoining colours.)

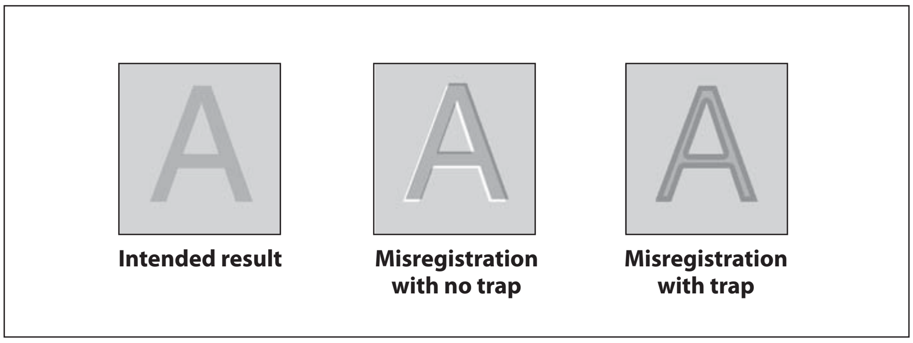


Figure 87 – Trapping example

Trapping may be implemented by the application generating a PDF file, by some intermediate application that adds traps to a PDF document, or by the raster image processor (RIP) that produces final output. In the last two cases, the trapping process is controlled by a set of *trapping instructions*, which define two kinds of information:

- *Trapping zones* within which traps should be created
- *Trapping parameters* specifying the nature of the traps within each zone

Trapping zones and trapping parameters are discussed fully in Sections 6.3.2 and 6.3.3, respectively, of the *PostScript Language Reference*, Third Edition. Trapping instructions are not directly specified in a PDF file (as they are in a PostScript file). Instead, they shall be specified in a *job ticket* that accompanies the PDF file or is embedded within it. Various standards exist for the format of job tickets; two of them, JDF (Job Definition Format) and PJTF (Portable Job Ticket Format), are described in the CIP4 document *JDF Specification* and in Adobe Technical Note #5620, *Portable Job Ticket Format* (see the Bibliography).

When trapping is performed before the production of final output, the resulting traps shall be placed in the PDF file for subsequent use. The traps themselves shall be described as a content stream in a trap network annotation (see 14.11.6.2, “Trap Network Annotations”). The stream dictionary may include additional entries describing the method that was used to produce the traps and other information about their appearance.

14.11.6.2 Trap Network Annotations

A complete set of traps generated for a given page under a specified set of trapping instructions is called a *trap network (PDF 1.3)*. It is a form XObject containing graphics objects for painting the required traps on the page. A page may have more than one trap network based on different trapping instructions, presumably intended for different output devices. All of the trap networks for a given page shall be contained in a single *trap network annotation* (see 12.5, “Annotations”). There may be at most one trap network annotation per page, which shall be the last element in the page’s **Annots** array (see 7.7.3.3, “Page Objects”). This ensures that the trap network shall be printed after all of the page’s other contents.

The form XObject defining a trap network shall be specified as an appearance stream in the **N** (normal) entry of the trap network annotation’s appearance dictionary (see 12.5.5, “Appearance Streams”). If more than one trap network is defined for the same page, the **N** entry holds a subdictionary containing the alternate trap networks, each identified by an arbitrary key. The **AS** (appearance state) entry in the annotation dictionary designates one of them as the *current trap network* to be displayed or printed.

NOTE 1 The trap network annotation’s appearance dictionary may include ***R*** (rollover) or ***D*** (down) entries, but appearances defined in either of these entries are never printed.

Like all annotations, a trap network annotation shall be defined by an annotation dictionary (see 12.5.2, “Annotation Dictionaries”); its annotation type is **TrapNet**. The **AP** (appearances), **AS** (appearance state), and **F** (flags) entries (which ordinarily are optional) shall be present, with the Print and ReadOnly flags set and all others clear (see 12.5.3, “Annotation Flags”). Table 366 shows the additional annotation dictionary entries specific to this type of annotation.

The **Version** and **AnnotStates** entries, if present, shall be used to detect changes in the content of a page that might require regenerating its trap networks. The **Version** array identifies elements of the page’s content that might be changed by an editing application and thus invalidate its trap networks. Because there is at most one **Version** array per trap network annotation (and thus per page), any conforming writer that generates a new trap network shall also verify the validity of existing trap networks by enumerating the objects identified in the array and verifying that the results exactly match the array’s current contents. Any trap networks found to be invalid shall be regenerated.

The **LastModified** entry may be used in place of the **Version** array to track changes to a page’s trap network. (The trap network annotation shall include either a **LastModified** entry or the combination of **Version** and **AnnotStates**, but not all three.) If the modification date in the **LastModified** entry of the page object (see 7.7.3.3, “Page Objects”) is more recent than the one in the trap network annotation dictionary, the page’s trap networks are invalid and shall be regenerated.

NOTE 2 Not all editing applications correctly maintain these modification dates.

This method of tracking trap network modifications may be used reliably only in a controlled workflow environment where the integrity of the modification dates is assured.

Table 366 – Additional entries specific to a trap network annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be **TrapNet** for a trap network annotation.
LastModified	date	*(Required if **Version** and **AnnotStates** are absent; shall be absent if **Version** and **AnnotStates** are present; PDF 1.4)* The date and time (see 7.9.4, “Dates”) when the trap network was most recently modified.
Version	array	*(Required if **AnnotStates** is present; shall be absent if **LastModified** is present)* An unordered array of all objects present in the page description at the time the trap networks were generated and that, if changed, could affect the appearance of the page. If present, the array shall include the following objects: • All content streams identified in the page object’s **Contents** entry (see 7.7.3.3, “Page Objects”) • All resource objects (other than procedure sets) in the page’s resource dictionary (see 7.8.3, “Resource Dictionaries”) • All resource objects (other than procedure sets) in the resource dictionaries of any form XObjects on the page (see 8.10, “Form XObjects”) • All OPI dictionaries associated with XObjects on the page (see 14.11.7, “Open Prepress Interface (OPI)”)
AnnotStates	array	*(Required if **Version** is present; shall be absent if **LastModified** is present)* An array of name objects representing the appearance states (value of the **AS** entry) for annotations associated with the page. The appearance states shall be listed in the same order as the annotations in the page’s **Annots** array (see 7.7.3.3, “Page Objects”). For an annotation with no **AS** entry, the corresponding array element should be **null**. No appearance state shall be included for the trap network annotation itself.

Table 366 – Additional entries specific to a trap network annotation (continued)

Key	Type	Value
FontFauxing	array	*(Optional)* An array of font dictionaries representing fonts that were fauxed (replaced by substitute fonts) during the generation of trap networks for the page.

14.11.6.3 Trap Network Appearances

Each entry in the **N** (normal) subdictionary of a trap network annotation's appearance dictionary holds an appearance stream defining a trap network associated with the given page. Like all appearances, a trap network is a stream object defining a form XObject (see 8.10, "Form XObjects"). The body of the stream contains the graphics objects needed to paint the traps making up the trap network. Its dictionary entries include, besides the standard entries for a form dictionary, the additional entries shown in Table 367.

Table 367 – Additional entries specific to a trap network appearance stream

Key	Type	Value
PCM	name	*(Required)* The name of the process colour model that was assumed when this trap network was created; equivalent to the PostScript page device parameter **ProcessColorModel** (see Section 6.2.5 of the PostScript Language Reference, Third Edition). Valid values are **DeviceGray**, **DeviceRGB**, **DeviceCMYK**, **DeviceCMY**, **DeviceRGBK**, and **DeviceN**.
SeparationColorNames	array	*(Optional)* An array of names identifying the colorants that were assumed when this network was created; equivalent to the PostScript page device parameter of the same name (see Section 6.2.5 of the PostScript Language Reference, Third Edition). Colourants implied by the process colour model **PCM** are available automatically and need not be explicitly declared. If this entry is absent, the colorants implied by **PCM** shall be assumed.
TrapRegions	array	*(Optional)* An array of indirect references to **TrapRegion** objects defining the page's trapping zones and the associated trapping parameters, as described in Adobe Technical Note #5620, Portable Job Ticket Format. These references refer to objects comprising portions of a PJTF job ticket that shall be embedded in the PDF file. When the trapping zones and parameters are defined by an external job ticket (or by some other means, such as JDF), this entry shall be absent.
TrapStyles	text string	*(Optional)* A human-readable text string that applications may use to describe this trap network to the user. EXAMPLE To allow switching between trap networks).

NOTE Preseparated PDF files (see 14.11.4, "Separation Dictionaries") may not be trapped because traps are defined along the borders between different colours and a preseparated file uses only one colour. Therefore, preseparation shall occur after trapping, not before. An conforming writer that preseparates a trapped PDF file is responsible for calculating new ***Version*** arrays for the separated trap networks.

14.11.7 Open Prepress Interface (OPI)

The workflow in a prepress environment often involves multiple applications in areas such as graphic design, page layout, word processing, photo manipulation, and document construction. As pieces of the final document are moved from one application to another, it is useful to separate the data of high-resolution images, which can be quite large—in some cases, many times the size of the rest of the document combined—from that of the document itself. The *Open Prepress Interface (OPI)* is a mechanism, originally developed by Aldus Corporation, for creating low-resolution placeholders, or *proxies*, for such high-resolution images. The proxy typically consists of a downsampled version of the full-resolution image, to be used for screen display and proofing. Before the document is printed, it passes through a filter known as an *OPI server*, which replaces the proxies with the original full-resolution images.

NOTE 1 In PostScript programs, OPI proxies are defined by PostScript code surrounded by special OPI comments, which specify such information as the placement and cropping of the image and adjustments to its size, rotation, colour, and other attributes.

In PDF, proxies shall be embedded in a document as image or form XObjects with an associated *OPI dictionary (PDF 1.2).* This dictionary contains the same information that the OPI comments convey in PostScript. Two versions of OPI shall be supported, versions 1.3 and 2.0. In OPI 1.3, a proxy consisting of a single image, with no changes in the graphics state, may be represented as an image XObject; otherwise it shall be a form XObject. In OPI 2.0, the proxy always entails changes in the graphics state and hence shall be represented as a form XObject.

An XObject representing an OPI proxy shall contain an **OPI** entry in its image or form dictionary (see Table 89 and Table 95). The value of this entry is an *OPI version dictionary* (Table 368) identifying the version of OPI to which the proxy corresponds. This dictionary consists of a single entry, whose key is the name **1.3** or **2.0** and whose value is the OPI dictionary defining the proxy's OPI attributes.

Table 368 – Entry in an OPI version dictionary

Key	Type	Value
version number	dictionary	*(Required; PDF 1.2)* An OPI dictionary specifying the attributes of this proxy (see Tables 369 and 370). The key for this entry shall be the name **1.3** or **2.0**, identifying the version of OPI to which the proxy corresponds.

NOTE 2 As in any other PDF dictionary, the key in an OPI version dictionary is a name object. The OPI version dictionary would thus be written in the PDF file in either the form

<< /1.3 *d* 0 R >>% OPI 1.3 dictionary

or

<< /2.0 *d* 0 R >>% OPI 2.0 dictionary

where *d* is the object number of the corresponding OPI dictionary.

Table 369 and Table 370 describe the contents of the OPI dictionaries for OPI 1.3 and OPI 2.0, respectively, along with the corresponding PostScript OPI comments. The dictionary entries shall be listed in the order in which the corresponding OPI comments appear in a PostScript program. Complete details on the meanings of these entries and their effects on OPI servers can be found in *OPI: Open Prepress Interface Specification 1.3* and Adobe Technical Note #5660, *Open Prepress Interface (OPI) Specification, Version 2.0*.

Table 369 – Entries in a version 1.3 OPI dictionary

Key	Type	OPI Comment	Value
Type	name		*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **OPI** for an OPI dictionary.
Version	number		*(Required)* The version of OPI to which this dictionary refers; shall be the number 1.3 (not the name 1.3, as in an OPI version dictionary).
F	file specification	%ALDImageFilename	*(Required)* The external file containing the image corresponding to this proxy.

Table 369 – Entries in a version 1.3 OPI dictionary (continued)

Key	Type	OPI Comment	Value
ID	byte string	%ALDImageID	*(Optional)* An identifying string denoting the image.
Comments	text string	%ALDObjectComments	*(Optional)* A human-readable comment, typically containing instructions or suggestions to the operator of the OPI server on how to handle the image.
Size	array	%ALDImageDimensions	*(Required)* An array of two integers of the form [pixelsWide pixelsHigh] specifying the dimensions of the image in pixels.
CropRect	rectangle	%ALDImageCropRect	*(Required)* An array of four integers of the form [left top right bottom] specifying the portion of the image to be used.
CropFixed	array	%ALDImageCropFixed	*(Optional)* An array with the same form and meaning as **CropRect**, but expressed in real numbers instead of integers. Default value: the value of **CropRect**.
Position	array	%ALDImagePosition	*(Required)* An array of eight numbers of the form $[ll_x\ ll_y\ ul_x\ ul_y\ ur_x\ ur_y\ lr_x\ lr_y]$ specifying the location on the page of the cropped image, where (ll_x, ll_y) are the user space coordinates of the lower-left corner, (ul_x, ul_y) are those of the upper-left corner, (ur_x, ur_y) are those of the upper-right corner, and (lr_x, lr_y) are those of the lower-right corner. The specified coordinates shall define a parallelogram; that is, they shall satisfy the conditions $ul_x - ll_x = ur_x - lr_x$ and $ul_y - ll_y = ur_y - lr_y$ The combination of **Position** and **CropRect** determines the image's scaling, rotation, reflection, and skew.
Resolution	array	%ALDImageResolution	*(Optional)* An array of two numbers of the form [horizRes vertRes] specifying the resolution of the image in samples per inch.
ColorType	name	%ALDImageColorType	*(Optional)* The type of colour specified by the **Color** entry. Valid values are Process, Spot, and Separation. Default value: Spot.

Table 369 – Entries in a version 1.3 OPI dictionary (continued)

Key	Type	OPI Comment	Value
Color	array	%ALDImageColor	*(Optional)* An array of four numbers and a byte string of the form [C M Y K colorName] specifying the value and name of the colour in which the image is to be rendered. The values of C, M, Y, and K shall all be in the range 0.0 to 1.0. Default value: [0.0 0.0 0.0 1.0 (Black)].
Tint	number	%ALDImageTint	*(Optional)* A number in the range 0.0 to 1.0 specifying the concentration of the colour specified by **Color** in which the image is to be rendered. Default value: 1.0.
Overprint	boolean	%ALDImageOverprint	*(Optional)* A flag specifying whether the image is to overprint (**true**) or knock out (**false**) underlying marks on other separations. Default value: **false**.
ImageType	array	%ALDImageType	*(Optional)* An array of two integers of the form [samples bits] specifying the number of samples per pixel and bits per sample in the image.
GrayMap	array	%ALDImageGrayMap	*(Optional)* An array of 2n integers in the range 0 to 65,535 (where n is the number of bits per sample) recording changes made to the brightness or contrast of the image.
Transparency	boolean	%ALDImageTransparency	*(Optional)* A flag specifying whether white pixels in the image shall be treated as transparent. Default value: **true**.
Tags	array	%ALDImageAsciiTag<NNN>	*(Optional)* An array of pairs of the form [tagNum1 tagText1 … tagNumn tagTextn] where each tagNum is an integer representing a TIFF tag number and each tagText is an ASCII string representing the corresponding ASCII tag value.

Table 370 – Entries in a version 2.0 OPI dictionary

Key	Type	OPI Comment	Value
Type	name		*(Optional)* The type of PDF object that this dictionary describes; if present, shall be **OPI** for an OPI dictionary.
Version	number		*(Required)* The version of OPI to which this dictionary refers; shall be the number 2 or 2.0 (not the name 2.0, as in an OPI version dictionary).
F	file specification	%%ImageFilename	*(Required)* The external file containing the low- resolution proxy image.
MainImage	byte string	%%MainImage	*(Optional)* The pathname of the file containing the full-resolution image corresponding to this proxy, or any other identifying string that uniquely identifies the full-resolution image.

Table 370 – Entries in a version 2.0 OPI dictionary (continued)

Key	Type	OPI Comment	Value
Tags	array	%%TIFFASCIITag	*(Optional)* An array of pairs of the form [tagNum1 tagText1 … tagNumn tagTextn] where each tagNum is an integer representing a TIFF tag number and each tagText is an ASCII string or an array of ASCII strings representing the corresponding ASCII tag value.
Size	array	%%ImageDimensions	*(Optional)* An array of two numbers of the form [width height] specifying the dimensions of the image in pixels.
CropRect	rectangle	%%ImageCropRect	*(Optional)* An array of four numbers of the form [left top right bottom] specifying the portion of the image to be used. The **Size** and **CropRect** entries shall either both be present or both be absent. If present, they shall satisfy the conditions $0 \leq \text{left} < \text{right} \leq \text{width}$ and $0 \leq \text{top} < \text{bottom} \leq \text{height}$ In this coordinate space, the positive y axis extends vertically downward; hence, the requirement that top < bottom.
Overprint	boolean	%%ImageOverprint	*(Optional)* A flag specifying whether the image is to overprint (**true**) or knock out (**false**) underlying marks on other separations. Default value: **false**.
Inks	name or array	%%ImageInks	*(Optional)* A name object or array specifying the colorants to be applied to the image. The value may be the name full_color or registration or an array of the form [/monochrome name1 tint1 … namen tintn] where each name is a string representing the name of a colorant and each tint is a real number in the range 0.0 to 1.0 specifying the concentration of that colorant to be applied.
IncludedImageDimensions			
	array	%%IncludedImageDimensions	*(Optional)* An array of two integers of the form [pixelsWide pixelsHigh] specifying the dimensions of the included image in pixels.
IncludedImageQuality			
	number	%%IncludedImageQuality	*(Optional)* A number indicating the quality of the included image. Valid values are 1, 2, and 3.

Annex A
(informative)

Operator Summary

A.1 General

This annex lists, in alphabetical order, all the operators that may be used in PDF content streams.

A.2 PDF Content Stream Operators

Table A.1 lists each operator, its corresponding PostScript language operators (when it is an exact or near-exact equivalent of the PDF operator), a description of the operator, and references to the table and page where each operator is introduced.

Table A.1 – PDF content stream operators

Operator	PostScript Equivalent	Description	Table
b	**closepath**, **fill**, **stroke**	Close, fill, and stroke path using nonzero winding number rule	60
B	**fill, stroke**	Fill and stroke path using nonzero winding number rule	60
b*	**closepath, eofill, stroke**	Close, fill, and stroke path using even-odd rule	60
B*	**eofill, stroke**	Fill and stroke path using even-odd rule	60
BDC		*(PDF 1.2)* Begin marked-content sequence with property list	320
BI		Begin inline image object	92
BMC		*(PDF 1.2)* Begin marked-content sequence	320
BT		Begin text object	107
BX		*(PDF 1.1)* Begin compatibility section	32
c	**curveto**	Append curved segment to path (three control points)	59
cm	**concat**	Concatenate matrix to current transformation matrix	57
CS	**setcolorspace**	*(PDF 1.1)* Set color space for stroking operations	74
cs	**setcolorspace**	*(PDF 1.1)* Set color space for nonstroking operations	74
d	**setdash**	Set line dash pattern	57
d0	**setcharwidth**	Set glyph width in Type 3 font	113
d1	**setcachedevice**	Set glyph width and bounding box in Type 3 font	113
Do		Invoke named XObject	87
DP		*(PDF 1.2)* Define marked-content point with property list	320
EI		End inline image object	92
EMC		*(PDF 1.2)* End marked-content sequence	320

Table A.1 – PDF content stream operators (continued)

Operator	PostScript Equivalent	Description	Table
ET		End text object	107
EX		*(PDF 1.1)* End compatibility section	32
f	**fill**	Fill path using nonzero winding number rule	60
F	**fill**	Fill path using nonzero winding number rule (obsolete)	60
f*	**eofill**	Fill path using even-odd rule	60
G	**setgray**	Set gray level for stroking operations	74
g	**setgray**	Set gray level for nonstroking operations	74
gs		*(PDF 1.2)* Set parameters from graphics state parameter dictionary	57
h	**closepath**	Close subpath	59
i	**setflat**	Set flatness tolerance	57
ID		Begin inline image data	92
j	**setlinejoin**	Set line join style	57
J	**setlinecap**	Set line cap style	57
K	**setcmykcolor**	Set *CMYK* color for stroking operations	74
k	**setcmykcolor**	Set *CMYK* color for nonstroking operations	74
l	**lineto**	Append straight line segment to path	59
m	**moveto**	Begin new subpath	59
M	**setmiterlimit**	Set miter limit	57
MP		*(PDF 1.2)* Define marked-content point	320
n		End path without filling or stroking	60
q	**gsave**	Save graphics state	57
Q	**grestore**	Restore graphics state	57
re		Append rectangle to path	59
RG	**setrgbcolor**	Set *RGB* color for stroking operations	74
rg	**setrgbcolor**	Set *RGB* color for nonstroking operations	74
ri		Set color rendering intent	57
s	**closepath, stroke**	Close and stroke path	60
S	**stroke**	Stroke path	60
SC	**setcolor**	*(PDF 1.1)* Set color for stroking operations	74
sc	**setcolor**	*(PDF 1.1)* Set color for nonstroking operations	74
SCN	**setcolor**	*(PDF 1.2)* Set color for stroking operations (**ICCBased** and special colour spaces)	74
scn	**setcolor**	*(PDF 1.2)* Set color for nonstroking operations (**ICCBased** and special colour spaces)	74

Table A.1 – PDF content stream operators (continued)

Operator	PostScript Equivalent	Description	Table
sh	**shfill**	*(PDF 1.3)* Paint area defined by shading pattern	77
T*		Move to start of next text line	108
Tc		Set character spacing	
Td		Move text position	108
TD		Move text position and set leading	108
Tf	**selectfont**	Set text font and size	
Tj	**show**	Show text	109
TJ		Show text, allowing individual glyph positioning	109
TL		Set text leading	
Tm		Set text matrix and text line matrix	108
Tr		Set text rendering mode	
Ts		Set text rise	
Tw		Set word spacing	
Tz		Set horizontal text scaling	
v	**curveto**	Append curved segment to path (initial point replicated)	59
w	**setlinewidth**	Set line width	57
W	**clip**	Set clipping path using nonzero winding number rule	61
W*	**eoclip**	Set clipping path using even-odd rule	61
y	**curveto**	Append curved segment to path (final point replicated)	59
'		Move to next line and show text	109
"		Set word and character spacing, move to next line, and show text	109

THIS PAGE BLANK

Annex B
(normative)

Operators in Type 4 Functions

B.1 General

This annex summarizes the PostScript operators that may appear in a type 4 function, as discussed in 7.10.5, "Type 4 (PostScript Calculator) Functions". For details on these operators, see the *PostScript Language Reference*, Third Edition.

B.2 Arithmetic Operators

num_1 num_2	**add** *sum*	Return num_1 plus num_2
num_1 num_2	**sub** *difference*	Return num_1 minus num_2
num_1 num_2	**mul** *product*	Return num_1 times num_2
num_1 num_2	**div** *quotient*	Return num_1 divided by num_2
int_1 int_2	**idiv** *quotient*	Return int_1 divided by int_2 as an integer
int_1 int_2	**mod** *remainder*	Return remainder after dividing int_1 by int_2
num_1	**neg** num_2	Return negative of num_1
num_1	**abs** num_2	Return absolute value of num_1
num_1	**ceiling** num_2	Return ceiling of num_1
num_1	**floor** num_2	Return floor of num_1
num_1	**round** num_2	Round num_1 to nearest integer
num_1	**truncate** num_2	Remove fractional part of num_1
num	**sqrt** *real*	Return square root of *num*
angle	**sin** *real*	Return sine of *angle* degrees
angle	**cos** *real*	Return cosine of *angle* degrees
num den	**atan** *angle*	Return arc tangent of *num*/*den* in degrees
base exponent	**exp** *real*	Raise *base* to *exponent* power
num	**ln** *real*	Return natural logarithm (base *e*)
num	**log** *real*	Return common logarithm (base 10)
num	**cvi** *int*	Convert to integer
num	**cvr** *real*	Convert to real

B.3 Relational, Boolean, and Bitwise Operators

any_1 any_2	**eq** *bool*	Test equal
any_1 any_2	**ne** *bool*	Test not equal
num_1 num_2	**gt** *bool*	Test greater than
num_1 num_2	**ge** *bool*	Test greater than or equal
num_1 num_2	**lt** *bool*	Test less than
num_1 num_2	**le** *bool*	Test less than or equal
$bool_1 \| int_1$ $bool_2 \| int_2$	**and** $bool_3 \| int_3$	Perform logical|bitwise and
$bool_1 \| int_1$ $bool_2 \| int_2$	**or** $bool_3 \| int_3$	Perform logical|bitwise inclusive or
$bool_1 \| int_1$ $bool_2 \| int_2$	**xor** $bool_3 \| int_3$	Perform logical|bitwise exclusive or
$bool_1 \| int_1$	**not** $bool_2 \| int_2$	Perform logical|bitwise not

int_1 shift	**bitshift** *int_2*	Perform bitwise shift of *int_1* (positive is left)
–	**true** *true*	Return boolean value *true*
–	**false** *false*	Return boolean value *false*

B.4 Conditional Operators

bool {*expr*}	**if** –	Execute *expr* if *bool* is *true*
bool {*$expr_1$*} {*$expr_2$*}	*ifelse* –	Execute *$expr_1$* if *bool* is *true*, *$expr_2$* if *false*

B.5 Stack Operators

any	**pop** –	Discard top element
any_1 any_2	**exch** any_2 any_1	Exchange top two elements
any	**dup** *any any*	Duplicate top element
any_1 … any_n n	**copy** any_1 … any_n any_1 … any_n	Duplicate top *n* elements
any_n … any_0 n	**index** any_n… any_0 any_n	Duplicate arbitrary element
$any_{n\text{-}1}$ … any_0 n j	**roll** $any_{(j\text{-}1) \bmod n}$ … any_0 $any_{n\text{-}1}$ … $any_{j \bmod n}$	Roll *n* elements up *j* times

Annex C
(normative)

Implementation Limits

C.1 General

In general, PDF does not restrict the size or quantity of things described in the file format, such as numbers, arrays, images, and so on. However, a conforming reader running on a particular processor and in a particular operating environment does have such limits. If a conforming reader encounters a PDF construct that exceeds one of these limits or performs a computation whose intermediate results exceed a limit, an error occurs.

NOTE PostScript interpreters also have implementation limits, listed in Appendix B of the *PostScript Language Reference*, Third Edition. It is possible to construct a PDF file that does not violate PDF implementation limits but fails to print on a PostScript printer. Keep in mind that these limits vary according to the PostScript LanguageLevel, interpreter version, and the amount of memory available to the interpreter.

This annex describes typical limits for a conforming PDF application (readers and writers). These limits fall into two main classes:

- *Architectural limits*. The hardware on which a conforming reader executes imposes certain constraints. For example, an integer is usually represented in 32 bits, limiting the range of allowed integers. In addition, the design of the software imposes other constraints, such as a limit to the number of elements in an array or string.

- *Memory limits*. The amount of memory available to a conforming reader limits the number of memory-consuming objects that can be held simultaneously.

C.2 Architectural limits

PDF itself has one architectural limit: Because a cross-reference table (see 7.5.4, "Cross-Reference Table") allocates ten digits to represent byte offsets, the size of a file shall be limited to 10^{10} bytes (approximately 10 gigabytes). This limit does not apply in a PDF file that uses a cross-reference stream (see 7.5.8, "Cross-Reference Streams") instead of a cross reference table.

Table C.1 describes the minimum architectural limits that should be accommodated by conforming readers running on 32-bit machines. Because conforming readers may be subject to these limits, conforming writers producing PDF files should remain within them.

NOTE Memory limits are often exceeded before architectural limits (such as the limit on the number of indirect objects) are reached.

Table C.1 – Architectural limits

Quantity	Limit	Description
integer	2,147,483,647	Largest integer value; equal to $2^{31} - 1$.
	-2,147,483,648	Smallest integer value; equal to -2^{31}.
real	$\pm 3.403 \times 10^{38}$	Largest and smallest real values (approximate).
	$\pm 1.175 \times 10^{-38}$	Nonzero real values closest to 0 (approximate). Values closer than these are automatically converted to 0.

Table C.1 – Architectural limits (continued)

Quantity	Limit	Description
	5	Number of significant decimal digits of precision in fractional part (approximate).
string (in content stream)	32,767	Maximum length of a string, in bytes. This restriction applies only to strings in content streams. There is no effective restriction on other strings in PDF files.
name	127	Maximum length of a name, in bytes.
indirect object	8,388,607	Maximum number of indirect objects in a PDF file.
q/**Q** nesting	28	NOTE Maximum depth of graphics state nesting by **q** and **Q** operators. This is not a limit as such, but arises from the fact that **q** and **Q** are implemented by the PostScript **gsave** and **grestore** operators when generating PostScript output.
DeviceN components	32	Maximum number of colorants or tint components in a **DeviceN** colour space.
CID	65,535	Maximum value of a CID (character identifier).

Additionally, conforming writers should adhere to the following constraints, and conforming readers should accommodate PDF files that obey the constraints.

- Thumbnail images should be no larger than 106 by 106 samples, and should be created at one-eighth scale for 8.5-by-11-inch and A4-size pages.
- The minimum page size should be 3 by 3 units in default user space; the maximum should be 14,400 by 14,400 units. In versions of PDF earlier than 1.6, the size of the default user space unit was fixed at 1⁄72 inch, yielding a minimum of approximately 0.04 by 0.04 inch and a maximum of 200 by 200 inches. Beginning with PDF 1.6, the size of the unit may be set on a page-by-page basis; the default remains at 1/72 inch.
- The magnification factor of a view should be constrained to be between approximately 8 percent and 6400 percent.
- When a conforming reader reads a PDF file with a damaged or missing cross-reference table, it may attempt to rebuild the table by scanning all the objects in the file. However, the generation numbers of deleted entries are lost if the cross-reference table is missing or severely damaged. To facilitate such reconstruction, object identifiers, the **endobj** keyword, and the endstream keyword should appear at the start of a line. Also, the data within a stream should not contain a line beginning with the word **endstream**, aside from the required **endstream** that delimits the end of the stream.

C.3 Memory limits

Memory limits cannot be characterized as precisely as architectural limits because the amount of available memory and the ways in which it is allocated vary from one conforming product to another.

NOTE Memory is automatically reallocated from one use to another when necessary: when more memory is needed for a particular purpose, it can be taken from memory allocated to another purpose if that memory is currently unused or its use is nonessential (a cache, for example). Also, data is often saved to a temporary file when memory is limited. Because of this behaviour, it is not possible to state limits for such items as the number of pages in a document, number of text annotations or hypertext links on a page, number of graphics objects on a page, or number of fonts on a page or in a document.

Annex D
(normative)

Character Sets and Encodings

D.1 General

This annex lists the character sets and encodings that shall be predefined in any conforming reader. Simple fonts, encompassing Latin text and some symbols, are described here. See 9.7.5.2, "Predefined CMaps" for a list of predefined CMaps for CID-keyed fonts.

D.2, "Latin Character Set and Encodings", describes the entire character set for the Adobe standard Latin-text fonts. This character set shall be supported by the Times, Helvetica, and Courier font families, which are among the standard 14 predefined fonts; see 9.6.2.2, "Standard Type 1 Fonts (Standard 14 Fonts)". For each named character, an octal character code is defined for four different encodings: **StandardEncoding**, **MacRomanEncoding**, **WinAnsiEncoding**, and **PDFDocEncoding** (see Table D.1). Unencoded characters are indicated by a dash (—).

D.3, "PDFDocEncoding Character Set", describes the entire set of characters that can be represented using PDFDocEncoding. It presents these characters in numerical order and it describes the Unicode representation of each character. This table overlaps the information presented in D.2, "Latin Character Set and Encodings", with respect to the presented octal character codes.

D.4, "Expert Set and MacExpertEncoding", describes the “expert” character set, which contains additional characters useful for sophisticated typography, such as small capitals, ligatures, and fractions. For each named character, an octal character code is given in **MacExpertEncoding**.

NOTE The built-in encoding in an expert font program may be different from **MacExpertEncoding**.

D.5, "Symbol Set and Encoding", and D.6, "ZapfDingbats Set and Encoding", describe the character sets and built-in encodings for the Symbol and ZapfDingbats (ITC Zapf Dingbats) font programs, which shall be among the standard 14 predefined fonts. These fonts have built-in encodings that are unique to each font. The characters for ZapfDingbats are ordered by code instead of by name, since the names in that font are meaningless.

Table D.1 – Latin-text encodings

Encoding	Description
StandardEncoding	Adobe standard Latin-text encoding. This is the built-in encoding defined in Type 1 Latin-text font programs (but generally not in TrueType font programs). Conforming readers shall not have a predefined encoding named **StandardEncoding**. However, it is necessary to describe this encoding, since a font’s built-in encoding can be used as the base encoding from which differences may be specified in an encoding dictionary.
MacRomanEncoding	Mac OS standard encoding for Latin text in Western writing systems. Conforming readers shall have a predefined encoding named **MacRomanEncoding** that may be used with both Type 1 and TrueType fonts.
WinAnsiEncoding	Windows Code Page 1252, often called the “Windows ANSI” encoding. This is the standard Windows encoding for Latin text in Western writing systems. Conforming readers shall have a predefined encoding named **WinAnsiEncoding** that may be used with both Type 1 and TrueType fonts.

Table D.1 – Latin-text encodings (continued)

Encoding	**Description**
PDFDocEncoding	Encoding for text strings in a PDF document *outside* the document's content streams. This is one of two encodings (the other being Unicode) that may be used to represent text strings; see 7.9.2.2, "Text String Type". PDF does not have a predefined encoding named **PDFDocEncoding**; it is not customary to use this encoding to show text from fonts.
MacExpertEncoding	An encoding for use with expert fonts—ones containing the expert character set. Conforming readers shall have a predefined encoding named **MacExpertEncoding**. Despite its name, it is not a platform-specific encoding; however, only certain fonts have the appropriate character set for use with this encoding. No such fonts are among the standard 14 predefined fonts.

D.2 Latin Character Set and Encodings

		CHAR CODE (OCTAL)						CHAR CODE (OCTAL)			
CHAR	NAME	STD	MAC	WIN	PDF	CHAR	NAME	STD	MAC	WIN	PDF
A	A	101	101	101	101	Œ	OE	352	316	214	226
Æ	AE	341	256	306	306	Ó	Oacute	—	356	323	323
Á	Aacute	—	347	301	301	Ô	Ocircumflex	—	357	324	324
Â	Acircumflex	—	345	302	302	Ö	Odieresis	—	205	326	326
Ä	Adieresis	—	200	304	304	Ò	Ograve	—	361	322	322
À	Agrave	—	313	300	300	Ø	Oslash	351	257	330	330
Å	Aring	—	201	305	305	Õ	Otilde	—	315	325	325
Ã	Atilde	—	314	303	303	P	P	120	120	120	120
B	B	102	102	102	102	Q	Q	121	121	121	121
C	C	103	103	103	103	R	R	122	122	122	122
Ç	Ccedilla	—	202	307	307	S	S	123	123	123	123
D	D	104	104	104	104	Š	Scaron	—	—	212	227
E	E	105	105	105	105	T	T	124	124	124	124
É	Eacute	—	203	311	311	Þ	Thorn	—	—	336	336
Ê	Ecircumflex	—	346	312	312	U	U	125	125	125	125
Ë	Edieresis	—	350	313	313	Ú	Uacute	—	362	332	332
È	Egrave	—	351	310	310	Û	Ucircumflex	—	363	333	333
Ð	Eth	—	—	320	320	Ü	Udieresis	—	206	334	334
€	Euro[1]	—	—	200	240	Ù	Ugrave	—	364	331	331
F	F	106	106	106	106	V	V	126	126	126	126
G	G	107	107	107	107	W	W	127	127	127	127
H	H	110	110	110	110	X	X	130	130	130	130
I	I	111	111	111	111	Y	Y	131	131	131	131
Í	Iacute	—	352	315	315	Ý	Yacute	—	—	335	335
Î	Icircumflex	—	353	316	316	Ÿ	Ydieresis	—	331	237	230
Ï	Idieresis	—	354	317	317	Z	Z	132	132	132	132
Ì	Igrave	—	355	314	314	Ž	Zcaron[2]	—	—	216	231
J	J	112	112	112	112	a	a	141	141	141	141
K	K	113	113	113	113	á	aacute	—	207	341	341
L	L	114	114	114	114	â	acircumflex	—	211	342	342
Ł	Lslash	350	—	—	225	´	acute	302	253	264	264
M	M	115	115	115	115	ä	adieresis	—	212	344	344
N	N	116	116	116	116	æ	ae	361	276	346	346
Ñ	Ntilde	—	204	321	321	à	agrave	—	210	340	340
O	O	117	117	117	117	&	ampersand	046	046	046	046

		CHAR CODE (OCTAL)			
CHAR	NAME	STD	MAC	WIN	PDF
å	aring	—	214	345	345
^	asciicircum	136	136	136	136
~	asciitilde	176	176	176	176
*	asterisk	052	052	052	052
@	at	100	100	100	100
ã	atilde	—	213	343	343
b	b	142	142	142	142
\	backslash	134	134	134	134
|	bar	174	174	174	174
{	braceleft	173	173	173	173
}	braceright	175	175	175	175
[	bracketleft	133	133	133	133
]	bracketright	135	135	135	135
˘	breve	306	371	—	030
¦	brokenbar	—	—	246	246
•	bullet[3]	267	245	225	200
c	c	143	143	143	143
ˇ	caron	317	377	—	031
ç	ccedilla	—	215	347	347
¸	cedilla	313	374	270	270
¢	cent	242	242	242	242
ˆ	circumflex	303	366	210	032
:	colon	072	072	072	072
,	comma	054	054	054	054
©	copyright	—	251	251	251
¤	currency[1]	250	333	244	244
d	d	144	144	144	144
†	dagger	262	240	206	201
‡	daggerdbl	263	340	207	202
°	degree	—	241	260	260
¨	dieresis	310	254	250	250
÷	divide	—	326	367	367
$	dollar	044	044	044	044
˙	dotaccent	307	372	—	033
ı	dotlessi	365	365	—	232
e	e	145	145	145	145
é	eacute	—	216	351	351
ê	ecircumflex	—	220	352	352
ë	edieresis	—	221	353	353
è	egrave	—	217	350	350
8	eight	070	070	070	070
…	ellipsis	274	311	205	203
—	emdash	320	321	227	204
–	endash	261	320	226	205
=	equal	075	075	075	075
ð	eth	—	—	360	360
!	exclam	041	041	041	041
¡	exclamdown	241	301	241	241
f	f	146	146	146	146
ﬁ	fi	256	336	—	223
5	five	065	065	065	065
ﬂ	fl	257	337	—	224
ƒ	florin	246	304	203	206
4	four	064	064	064	064
⁄	fraction	244	332	—	207
g	g	147	147	147	147
ß	germandbls	373	247	337	337
`	grave	301	140	140	140
>	greater	076	076	076	076
«	guillemotleft[4]	253	307	253	253
»	guillemotright[4]	273	310	273	273
‹	guilsinglleft	254	334	213	210
›	guilsinglright	255	335	233	211
h	h	150	150	150	150
˝	hungarumlaut	315	375	—	034
-	hyphen[5]	055	055	055	055
i	i	151	151	151	151
í	iacute	—	222	355	355
î	icircumflex	—	224	356	356
ï	idieresis	—	225	357	357
ì	igrave	—	223	354	354
j	j	152	152	152	152
k	k	153	153	153	153
l	l	154	154	154	154

		CHAR CODE (OCTAL)						CHAR CODE (OCTAL)			
CHAR	NAME	STD	MAC	WIN	PDF	CHAR	NAME	STD	MAC	WIN	PDF
<	less	074	074	074	074	q	q	161	161	161	161
¬	logicalnot	—	302	254	254	?	question	077	077	077	077
ł	lslash	370	—	—	233	¿	questiondown	277	300	277	277
m	m	155	155	155	155	"	quotedbl	042	042	042	042
¯	macron	305	370	257	257	„	quotedblbase	271	343	204	214
−	minus	—	—	—	212	“	quotedblleft	252	322	223	215
µ	mu	—	265	265	265	”	quotedblright	272	323	224	216
×	multiply	—	—	327	327	‘	quoteleft	140	324	221	217
n	n	156	156	156	156	’	quoteright	047	325	222	220
9	nine	071	071	071	071	‚	quotesinglbase	270	342	202	221
ñ	ntilde	—	226	361	361	'	quotesingle	251	047	047	047
#	numbersign	043	043	043	043	r	r	162	162	162	162
o	o	157	157	157	157	®	registered	—	250	256	256
ó	oacute	—	227	363	363	˚	ring	312	373	—	036
ô	ocircumflex	—	231	364	364	s	s	163	163	163	163
ö	odieresis	—	232	366	366	š	scaron	—	—	232	235
œ	oe	372	317	234	234	§	section	247	244	247	247
˛	ogonek	316	376	—	035	;	semicolon	073	073	073	073
ò	ograve	—	230	362	362	7	seven	067	067	067	067
1	one	061	061	061	061	6	six	066	066	066	066
½	onehalf	—	—	275	275	/	slash	057	057	057	057
¼	onequarter	—	—	274	274		space[6]	040	040	040	040
¹	onesuperior	—	—	271	271	£	sterling	243	243	243	243
ª	ordfeminine	343	273	252	252	t	t	164	164	164	164
º	ordmasculine	353	274	272	272	þ	thorn	—	—	376	376
ø	oslash	371	277	370	370	3	three	063	063	063	063
õ	otilde	—	233	365	365	¾	threequarters	—	—	276	276
p	p	160	160	160	160	³	threesuperior	—	—	263	263
¶	paragraph	266	246	266	266	˜	tilde	304	367	230	037
(	parenleft	050	050	050	050	™	trademark	—	252	231	222
)	parenright	051	051	051	051	2	two	062	062	062	062
%	percent	045	045	045	045	²	twosuperior	—	—	262	262
.	period	056	056	056	056	u	u	165	165	165	165
·	periodcentered	264	341	267	267	ú	uacute	—	234	372	372
‰	perthousand	275	344	211	213	û	ucircumflex	—	236	373	373
+	plus	053	053	053	053	ü	udieresis	—	237	374	374
±	plusminus	—	261	261	261	ù	ugrave	—	235	371	371

		CHAR CODE (OCTAL)						CHAR CODE (OCTAL)			
CHAR	NAME	STD	MAC	WIN	PDF	CHAR	NAME	STD	MAC	WIN	PDF
_	underscore	137	137	137	137	ÿ	ydieresis	—	330	377	377
v	v	166	166	166	166	¥	yen	245	264	245	245
w	w	167	167	167	167	z	z	172	172	172	172
x	x	170	170	170	170	ž	zcaron[2]	—	—	236	236
y	y	171	171	171	171	0	zero	060	060	060	060
ý	yacute	—	—	375	375						

1. In PDF 1.3, the euro character was added to the Adobe standard Latin character set. It shall be encoded as 200 in **WinAnsiEncoding** and 240 in **PDFDocEncoding**, assigning codes that were previously unused. Apple changed the Mac OS Latin-text encoding for code 333 from the currency character to the euro character. However, this incompatible change has *not* been reflected in PDF's **MacRomanEncoding**, which shall continue to map code 333 to currency. If the euro character is desired, an encoding dictionary may be used to specify this single difference from **MacRomanEncoding**.
2. In PDF 1.3, the existing Zcaron and zcaron characters were added to **WinAnsiEncoding** as the previously unused codes 216 and 236.
3. In **WinAnsiEncoding**, all unused codes greater than 40 map to the bullet character. However, only code 225 shall be specifically assigned to the bullet character; other codes are subject to future reassignment.
4. The character names guillemotleft and guillemotright are misspelled. The correct spelling for this punctuation character is *guillemet*. However, the misspelled names are the ones actually used in the fonts and encodings containing these characters.
5. The hyphen character is also encoded as 255 in **WinAnsiEncoding**. The meaning of this duplicate code shall be "soft hyphen," but it shall be typographically the same as hyphen.
6. The SPACE character shall also be encoded as 312 in **MacRomanEncoding** and as 240 in **WinAnsiEncoding**. This duplicate code shall signify a nonbreaking space; it shall be typographically the same as (U+003A) SPACE.

D.3 PDFDocEncoding Character Set

The column titled Notes uses the following abbreviations:

U	*Undefined* code point in PDFDocEncoding
SR	Unicode codepoint that may require *special representation* in XML in some contexts.

Table D.2 – PDFDocEncoding Character Set

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
^@	0	0x00	0000	U+0000	(NULL)	U
^A	1	0x01	0001	U+0001	(START OF HEADING)	U
^B	2	0x02	0002	U+0002	(START OF TEXT)	U
^C	3	0x03	0003	U+0003	(END OF TEXT)	U
^D	4	0x04	0004	U+0004	(END OF TEXT)	U
^E	5	0x05	0005	U+0005	(END OF TRANSMISSION)	U

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
^F	6	0x06	0006	U+0006	(ACKNOWLEDGE)	U
^G	7	0x07	0007	U+0007	(BELL)	U
^H	8	0x08	0010	U+0008	(BACKSPACE)	U
^I	9	0x09	0011	U+0009	(CHARACTER TABULATION)	SR
^J	10	0x0a	0012	U+000A	(LINE FEED)	SR
^K	11	0x0b	0013	U+000B	(LINE TABULATION)	U
^L	12	0x0c	0014	U+000C	(FORM FEED)	U
^M	13	0x0d	0015	U+000D	(CARRIAGE RETURN)	SR
^N	14	0x0e	0016	U+000E	(SHIFT OUT)	U
^O	15	0x0f	0017	U+000F	(SHIFT IN)	U
^P	16	0x10	0020	U+0010	(DATA LINK ESCAPE)	U
^Q	17	0x11	0021	U+0011	(DEVICE CONTROL ONE)	U
^R	18	0x12	0022	U+0012	(DEVICE CONTROL TWO)	U
^S	19	0x13	0023	U+0013	(DEVICE CONTROL THREE)	U
^T	20	0x14	0024	U+0014	(DEVICE CONTROL FOUR)	U
^U	21	0x15	0025	U+0015	(NEGATIVE ACKNOWLEDGE)	U
^V	22	0x16	0026	U+0017	(SYNCRONOUS IDLE)	U
^W	23	0x17	0027	U+0017	(END OF TRANSMISSION BLOCK)	U
u	24	0x18	0030	U+02D8	BREVE	
v	25	0x19	0031	U+02C7	CARON	
^	26	0x1a	0032	U+02C6	MODIFIER LETTER CIRCUMFLEX ACCENT	
·	27	0x1b	0033	U+02D9	DOT ABOVE	
”	28	0x1c	0034	U+02DD	DOUBLE ACUTE ACCENT	
,	29	0x1d	0035	U+02DB	OGONEK	
°	30	0x1e	0036	U+02DA	RING ABOVE	
~	31	0x1f	0037	U+02DC	SMALL TILDE	
	32	0x20	0040	U+0020	SPACE (&#32;)	
!	33	0x21	0041	U+0021	EXCLAMATION MARK	SR
"	34	0x22	0042	U+0022	QUOTATION MARK (&quot;)	SR
#	35	0x23	0043	U+0023	NUMBER SIGN	
$	36	0x24	0044	U+0024	DOLLAR SIGN	
%	37	0x25	0045	U+0025	PERCENT SIGN	
&	38	0x26	0046	U+0026	AMPERSAND (&amp;)	

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
'	39	0x27	0047	U+0027	APOSTROPHE (&apos;)	
(	40	0x28	0050	U+0028	LEFT PARENTHESIS	
)	41	0x29	0051	U+0029	RIGHT PARENTHESIS	
*	42	0x2a	0052	U+002A	ASTERISK	
+	43	0x2b	0053	U+002B	PLUS SIGN	
,	44	0x2c	0054	U+002C	COMMA	
-	45	0x2d	0055	U+002D	HYPHEN-MINUS	
.	46	0x2e	0056	U+002E	FULL STOP (PERIOD)	
/	47	0x2f	0057	U+002F	SOLIDUS (slash)	
0	48	0x30	0060	U+0030	DIGIT ZERO	
1	49	0x31	0061	U+0031	DIGIT ONE	
2	50	0x32	0062	U+0032	DIGIT TWO	
3	51	0x33	0063	U+0033	DIGIT THREE	
4	52	0x34	0064	U+0034	DIGIT FOUR	
5	53	0x35	0065	U+0035	DIGIT FIVE	
6	54	0x36	0066	U+0036	DIGIT SIX	
7	55	0x37	0067	U+0037	DIGIT SEVEN	
8	56	0x38	0070	U+0038	DIGIT EIGJT	
9	57	0x39	0071	U+0039	DIGIT NINE	
:	58	0x3a	0072	U+003A	COLON	
;	59	0x3b	0073	U+003B	SEMICOLON	
<	60	0x3c	0074	U+003C	LESS THAN SIGN (&lt;)	SR
=	61	0x3d	0075	U+003D	EQUALS SIGN	
>	62	0x3e	0076	U+003E	GREATER THAN SIGN (&gt;)	
?	63	0x3f	0077	U+003F	QUESTION MARK	
@	64	0x40	0100	U+0040	COMMERCIAL AT	
A	65	0x41	0101	U+0041		
B	66	0x42	0102	U+0042		
C	67	0x43	0103	U+0043		
D	68	0x44	0104	U+0044		
E	69	0x45	0105	U+0045		
F	70	0x46	0106	U+0046		
G	71	0x47	0107	U+0047		

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
H	72	0x48	0110	U+0048		
I	73	0x49	0111	U+0049		
J	74	0x4a	0112	U+004A		
K	75	0x4b	0113	U+004B		
L	76	0x4c	0114	U+004C		
M	77	0x4d	0115	U+004D		
N	78	0x4e	0116	U+004E		
O	79	0x4f	0117	U+004F		
P	80	0x50	0120	U+0050		
Q	81	0x51	0121	U+0051		
R	82	0x52	0122	U+0052		
S	83	0x53	0123	U+0053		
T	84	0x54	0124	U+0054		
U	85	0x55	0125	U+0055		
V	86	0x56	0126	U+0056		
W	87	0x57	0127	U+0057		
X	88	0x58	0130	U+0058		
Y	89	0x59	0131	U+0059		
Z	90	0x5a	0132	U+005A		
[	91	0x5b	0133	U+005B	LEFT SQUARE BRACKET	
\	92	0x5c	0134	U+005C	REVERSE SOLIDUS (backslash)	
]	93	0x5d	0135	U+005D	RIGHT SQUARE BRACKET	
^	94	0x5e	0136	U+005E	CIRCUMFLEX ACCENT (hat)	
_	95	0x5f	0137	U+005F	LOW LINE (SPACING UNDERSCORE)	
`	96	0x60	0140	U+0060	GRAVE ACCENT	
a	97	0x61	0141	U+0061		
b	98	0x62	0142	U+0062		
c	99	0x63	0143	U+0063		
d	100	0x64	0144	U+0064		
e	101	0x65	0145	U+0065		
f	102	0x66	0146	U+0066		
g	103	0x67	0147	U+0067		
h	104	0x68	0150	U+0068		

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
i	105	0x69	0151	U+0069		
j	106	0x6a	0152	U+006A		
k	107	0x6b	0153	U+006B		
l	108	0x6c	0154	U+006C		
m	109	0x6d	0155	U+006D		
n	110	0x6e	0156	U+006E		
o	111	0x6f	0157	U+006F		
p	112	0x70	0160	U+0070		
q	113	0x71	0161	U+0071		
r	114	0x72	0162	U+0072		
s	115	0x73	0163	U+0073		
t	116	0x74	0164	U+0074		
u	117	0x75	0165	U+0075		
v	118	0x76	0166	U+0076		
w	119	0x77	0167	U+0077		
x	120	0x78	0170	U+0078		
y	121	0x79	0171	U+0079		
z	122	0x7a	0172	U+007A		
{	123	0x7b	0173	U+007B	LEFT CURLY BRACKET	
|	124	0x7c	0174	U+007C	VERTICAL LINE	
}	125	0x7d	0175	U+007D	RIGHT CURLY BRACKET	
~	126	0x7e	0176	U+007E	TILDE	
	127	0x7f	0177		Undefined	U
•	128	0x80	0200	U+2022	BULLET	
†	129	0x81	0201	U+2020	DAGGER	
‡	130	0x82	0202	U+2021	DOUBLE DAGGER	
…	131	0x83	0203	U+2026	HORIZONTAL ELLIPSIS	
—	132	0x84	0204	U+2014	EM DASH	
–	133	0x85	0205	U+2013	EN DASH	
ƒ	134	0x86	0206	U+0192		
	135	0x87	0207	U+2044	FRACTION SLASH (solidus)	
‹	136	0x88	0210	U+2039	SINGLE LEFT-POINTING ANGLE QUOTATION MARK	

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
›	137	0x89	0211	U+203A	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK	
Š	138	0x8a	0212	U+2212		
‰	139	0x8b	0213	U+2030	PER MILLE SIGN	
„	140	0x8c	0214	U+201E	DOUBLE LOW-9 QUOTATION MARK (quotedblbase)	
“	141	0x8d	0215	U+201C	LEFT DOUBLE QUOTATION MARK (double quote left)	
”	142	0x8e	0216	U+201D	RIGHT DOUBLE QUOTATION MARK (quotedblright)	
‘	143	0x8f	0217	U+2018	LEFT SINGLE QUOTATION MARK (quoteleft)	
’	144	0x90	0220	U+2019	RIGHT SINGLE QUOTATION MARK (quoteright)	
‚	145	0x91	0221	U+201A	SINGLE LOW-9 QUOTATION MARK (quotesinglbase)	
™	146	0x92	0222	U+2122	TRADE MARK SIGN	
fi	147	0x93	0223	U+FB01	LATIN SMALL LIGATURE FI	
fl	148	0x94	0224	U+FB02	LATIN SMALL LIGATURE FL	
	149	0x95	0225	U+0141	LATIN CAPITAL LETTER L WITH STROKE	
OE	150	0x96	0226	U+0152	LATIN CAPITAL LIGATURE OE	
Š	151	0x97	0227	U+0160	LATIN CAPITAL LETTER S WITH CARON	
Ÿ	152	0x98	0230	U+0178	LATIN CAPITAL LETTER Y WITH DIAERESIS	
Z hat	153	0x99	0231	U+017D	LATIN CAPITAL LETTER Z WITH CARON	
i	154	0x9a	0232	U+0131	LATIN SMALL LETTER DOTLESS I	
l/	155	0x9b	0233	U+0142	LATIN SMALL LETTER L WITH STROKE	
œ	156	0x9c	0234	U+0153	LATIN SMALL LIGATURE OE	
š	157	0x9d	0235	U+0161	LATIN SMALL LETTER S WITH CARON	
ž	158	0x9e	0236	U+017E	LATIN SMALL LETTER Z WITH CARON	
	159	0x9f	0237		Undefined	U
€	160	0xa0	0240	U+20AC	EURO SIGN	
¡	161	0xa1	0241	U+00A1	INVERTED EXCLAMATION MARK	
¢	162	0xa2	0242	U+00A2	CENT SIGN	
£	163	0xa3	0243	U+00A3	POUND SIGN (sterling)	
¤	164	0xa4	0244	U+00A4	CURRENCY SIGN	
¥	165	0xa5	0245	U+00A5	YEN SIGN	

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
¦	166	0xa6	0246	U+00A6	BROKEN BAR	
§	167	0xa7	0247	U+00A7	SECTION SIGN	
¨	168	0xa8	0250	U+00A8	DIAERESIS	
©	169	0xa9	0251	U+00A9	COPYRIGHT SIGN	
ª	170	0xaa	0252	U+00AA	FEMININE ORDINAL INDICATOR	
«	171	0xab	0253	U+00AB	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK	
¬	172	0xac	0254	U+00AC	NOT SIGN	
	173	0xad	0255		Undefined	U
®	174	0xae	0256	U+00AE	REGISTERED SIGN	
¯	175	0xaf	0257	U+00AF	MACRON	
°	176	0xb0	0260	U+00B0	DEGREE SIGN	
±	177	0xb1	0261	U+00B1	PLUS-MINUS SIGN	
²	178	0xb2	0262	U+00B2	SUPERSCRIPT TWO	
³	179	0xb3	0263	U+00B3	SUPERSCRIPT THREE	
´	180	0xb4	0264	U+00B4	ACUTE ACCENT	
µ	181	0xb5	0265	U+00B5	MICRO SIGN	
¶	182	0xb6	0266	U+00B6	PILCROW SIGN	
·	183	0xb7	0267	U+00B7	MIDDLE DOT	
¸	184	0xb8	0270	U+00B8	CEDILLA	
¹	185	0xb9	0271	U+00B9	SUPERSCRIPT ONE	
º	186	0xba	0272	U+00BA	MASCULINE ORDINAL INDICATOR	
»	187	0xbb	0273	U+00BB	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK	
¼	188	0xbc	0274	U+00BC	VULGAR FRACTION ONE QUARTER	
½	189	0xbd	0275	U+00BD	VULGAR FRACTION ONE HALF	
¾	190	0xbe	0276	U+00BE	VULGAR FRACTION THREE QUARTERS	
¿	191	0xbf	0277	U+00BF	INVERTED QUESTION MARK	
À	192	0xc0	0300	U+00C0		
Á	193	0xc1	0301	U+00C1		
Â	194	0xc2	0302	U+00C2		
Ã	195	0xc3	0303	U+00C3		
Ä	196	0xc4	0304	U+00C4		
Å	197	0xc5	0305	U+00C5		

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
Æ	198	0xc6	0306	U+00C6		
Ç	199	0xc7	0307	U+00C7		
È	200	0xc8	0310	U+00C8		
É	201	0xc9	0311	U+00C9		
Ê	202	0xca	0312	U+00CA		
Ë	203	0xcb	0313	U+00CB		
Ì	204	0xcc	0314	U+00CC		
Í	205	0xcd	0315	U+00CD		
Î	206	0xce	0316	U+00CE		
Ï	207	0xcf	0317	U+00CF		
Ð	208	0xd0	0320	U+00D0		
Ñ	209	0xd1	0321	U+00D1		
Ò	210	0xd2	0322	U+00D2		
Ó	211	0xd3	0323	U+00D3		
Ô	212	0xd4	0324	U+00D4		
Õ	213	0xd5	0325	U+00D5		
Ö	214	0xd6	0326	U+00D6		
×	215	0xd7	0327	U+00D7		
Ø	216	0xd8	0330	U+00D8		
Ù	217	0xd9	0331	U+00D9		
Ú	218	0xda	0332	U+00DA		
Û	219	0xdb	0333	U+00DB		
Ü	220	0xdc	0334	U+00DC		
Ý	221	0xdd	0335	U+00DD		
Þ	222	0xde	0336	U+00DE		
ß	223	0xdf	0337	U+00DF		
à	224	0xe0	0340	U+00E0		
á	225	0xe1	0341	U+00E1		
â	226	0xe2	0342	U+00E2		
ã	227	0xe3	0343	U+00E3		
ä	228	0xe4	0344	U+00E4		
å	229	0xe5	0345	U+00E5		
æ	230	0xe6	0346	U+00E6		

Table D.2 – PDFDocEncoding Character Set (continued)

Character	Dec	Hex	Octal	Unicode	Unicode character name or (alternative alias)	Notes
ç	231	0xe7	0347	U+00E7		
è	232	0xe8	0350	U+00E8		
é	233	0xe9	0351	U+00E9		
ê	234	0xea	0352	U+00EA		
ë	235	0xeb	0353	U+00EB		
ì	236	0xec	0354	U+00EC		
í	237	0xed	0355	U+00ED		
î	238	0xee	0356	U+00EE		
ï	239	0xef	0357	U+00EF		
ð	240	0xf0	0360	U+00F0		
ñ	241	0xf1	0361	U+00F1		
ò	242	0xf2	0362	U+00F2		
ó	243	0xf3	0363	U+00F3		
ô	244	0xf4	0364	U+00F4		
õ	245	0xf5	0365	U+00F5		
ö	246	0xf6	0366	U+00F6		
÷	247	0xf7	0367	U+00F7		
ø	248	0xf8	0370	U+00F8		
ù	249	0xf9	0371	U+00F9		
ú	250	0xfa	0372	U+00FA		
û	251	0xfb	0373	U+00FB		
ü	252	0xfc	0374	U+00FC		
ý	253	0xfd	0375	U+00FD		
þ	254	0xfe	0376	U+00FE		
ÿ	255	0xff	0377	U+00FF		

D.4 Expert Set and MacExpertEncoding

CHAR	NAME	CODE	CHAR	NAME	CODE
Æ	AEsmall	276	J	Jsmall	152
Á	Aacutesmall	207	K	Ksmall	153
Â	Acircumflexsmall	211	Ł	Lslashsmall	302
´	Acutesmall	047	L	Lsmall	154
Ä	Adieresissmall	212	¯	Macronsmall	364
À	Agravesmall	210	M	Msmall	155
Å	Aringsmall	214	N	Nsmall	156
A	Asmall	141	Ñ	Ntildesmall	226
Ã	Atildesmall	213	Œ	OEsmall	317
˘	Brevesmall	363	Ó	Oacutesmall	227
B	Bsmall	142	Ô	Ocircumflexsmall	231
ˇ	Caronsmall	256	Ö	Odieresissmall	232
Ç	Ccedillasmall	215	˛	Ogoneksmall	362
¸	Cedillasmall	311	Ò	Ogravesmall	230
ˆ	Circumflexsmall	136	Ø	Oslashsmall	277
C	Csmall	143	O	Osmall	157
¨	Dieresissmall	254	Õ	Otildesmall	233
˙	Dotaccentsmall	372	P	Psmall	160
D	Dsmall	144	Q	Qsmall	161
É	Eacutesmall	216	˚	Ringsmall	373
Ê	Ecircumflexsmall	220	R	Rsmall	162
Ë	Edieresissmall	221	Š	Scaronsmall	247
È	Egravesmall	217	S	Ssmall	163
E	Esmall	145	Þ	Thornsmall	271
Ð	Ethsmall	104	˜	Tildesmall	176
F	Fsmall	146	T	Tsmall	164
`	Gravesmall	140	Ú	Uacutesmall	234
G	Gsmall	147	Û	Ucircumflexsmall	236
H	Hsmall	150	Ü	Udieresissmall	237
˝	Hungarumlautsmall	042	Ù	Ugravesmall	235
Í	Iacutesmall	222	U	Usmall	165
Î	Icircumflexsmall	224	V	Vsmall	166
Ï	Idieresissmall	225	W	Wsmall	167
Ì	Igravesmall	223	X	Xsmall	170
I	Ismall	151	Ý	Yacutesmall	264

CHAR	NAME	CODE	CHAR	NAME	CODE
ÿ	Ydieresissmall	330	4	fouroldstyle	064
y	Ysmall	171	4	foursuperior	335
ž	Zcaronsmall	275	⁄	fraction	057
z	Zsmall	172	-	hyphen	055
&	ampersandsmall	046	-	hypheninferior	137
a	asuperior	201	-	hyphensuperior	321
b	bsuperior	365	i	isuperior	351
¢	centinferior	251	l	lsuperior	361
¢	centoldstyle	043	m	msuperior	367
¢	centsuperior	202	9	nineinferior	273
:	colon	072	9	nineoldstyle	071
₡	colonmonetary	173	9	ninesuperior	341
,	comma	054	n	nsuperior	366
,	commainferior	262	.	onedotenleader	053
,	commasuperior	370	⅛	oneeighth	112
$	dollarinferior	266	1	onefitted	174
$	dollaroldstyle	044	½	onehalf	110
$	dollarsuperior	045	1	oneinferior	301
d	dsuperior	353	1	oneoldstyle	061
8	eightinferior	245	¼	onequarter	107
8	eightoldstyle	070	1	onesuperior	332
8	eightsuperior	241	⅓	onethird	116
e	esuperior	344	o	osuperior	257
¡	exclamdownsmall	326	(	parenleftinferior	133
!	exclamsmall	041	(	parenleftsuperior	050
ff	ff	126	)	parenrightinferior	135
ffi	ffi	131	)	parenrightsuperior	051
ffl	ffl	132	.	period	056
fi	fi	127	.	periodinferior	263
‒	figuredash	320	.	periodsuperior	371
⅝	fiveeighths	114	¿	questiondownsmall	300
5	fiveinferior	260	?	questionsmall	077
5	fiveoldstyle	065	r	rsuperior	345
5	fivesuperior	336	Rp	rupiah	175
fl	fl	130	;	semicolon	073
4	fourinferior	242	⅞	seveneighths	115

CHAR	NAME	CODE	CHAR	NAME	CODE
$_7$	seveninferior	246	—	threequartersemdash	075
7	sevenoldstyle	067	^{3}	threesuperior	334
^{7}	sevensuperior	340	^{t}	tsuperior	346
$_6$	sixinferior	244	..	twodotenleader	052
6	sixoldstyle	066	$_2$	twoinferior	252
^{6}	sixsuperior	337	2	twooldstyle	062
	space	040	^{2}	twosuperior	333
^{s}	ssuperior	352	⅔	twothirds	117
⅜	threeeighths	113	$_0$	zeroinferior	274
$_3$	threeinferior	243	0	zerooldstyle	060
3	threeoldstyle	063	^{0}	zerosuperior	342
¾	threequarters	111			

D.5 Symbol Set and Encoding

CHAR	NAME	CODE	CHAR	NAME	CODE
Α	Alpha	101	↔	arrowboth	253
Β	Beta	102	⇔	arrowdblboth	333
Χ	Chi	103	⇓	arrowdbldown	337
Δ	Delta	104	⇐	arrowdblleft	334
Ε	Epsilon	105	⇒	arrowdblright	336
Η	Eta	110	⇑	arrowdblup	335
€	Euro	240	↓	arrowdown	257
Γ	Gamma	107	⎯	arrowhorizex	276
ℑ	Ifraktur	301	←	arrowleft	254
Ι	Iota	111	→	arrowright	256
Κ	Kappa	113	↑	arrowup	255
Λ	Lambda	114	⏐	arrowvertex	275
Μ	Mu	115	∗	asteriskmath	052
Ν	Nu	116	|	bar	174
Ω	Omega	127	β	beta	142
Ο	Omicron	117	{	braceleft	173
Φ	Phi	106	}	braceright	175
Π	Pi	120	⎧	bracelefttp	354
Ψ	Psi	131	⎨	braceleftmid	355
ℜ	Rfraktur	302	⎩	braceleftbt	356
Ρ	Rho	122	⎫	bracerighttp	374
Σ	Sigma	123	⎬	bracerightmid	375
Τ	Tau	124	⎭	bracerightbt	376
Θ	Theta	121	⎪	braceex	357
Υ	Upsilon	125	[	bracketleft	133
ϒ	Upsilon1	241	]	bracketright	135
Ξ	Xi	130	⎡	bracketlefttp	351
Ζ	Zeta	132	⎢	bracketleftex	352
ℵ	aleph	300	⎣	bracketleftbt	353
α	alpha	141	⎤	bracketrighttp	371
&	ampersand	046	⎥	bracketrightex	372
∠	angle	320	⎦	bracketrightbt	373
〈	angleleft	341	•	bullet	267
〉	angleright	361	↵	carriagereturn	277
≈	approxequal	273	χ	chi	143

CHAR	NAME	CODE	CHAR	NAME	CODE
⊗	circlemultiply	304	⌡	integralbt	365
⊕	circleplus	305	∩	intersection	307
♣	club	247	ι	iota	151
:	colon	072	κ	kappa	153
,	comma	054	λ	lambda	154
≅	congruent	100	<	less	074
©	copyrightsans	343	≤	lessequal	243
©	copyrightserif	323	∧	logicaland	331
°	degree	260	¬	logicalnot	330
δ	delta	144	∨	logicalor	332
♦	diamond	250	◊	lozenge	340
÷	divide	270	−	minus	055
·	dotmath	327	′	minute	242
8	eight	070	μ	mu	155
∈	element	316	×	multiply	264
…	ellipsis	274	9	nine	071
∅	emptyset	306	∉	notelement	317
ε	epsilon	145	≠	notequal	271
=	equal	075	⊄	notsubset	313
≡	equivalence	272	ν	nu	156
η	eta	150	#	numbersign	043
!	exclam	041	ω	omega	167
∃	existential	044	ϖ	omega1	166
5	five	065	ο	omicron	157
ƒ	florin	246	1	one	061
4	four	064	(	parenleft	050
⁄	fraction	244	)	parenright	051
γ	gamma	147	⎛	parenlefttp	346
∇	gradient	321	⎜	parenleftex	347
>	greater	076	⎝	parenleftbt	350
≥	greaterequal	263	⎞	parenrighttp	366
♥	heart	251	⎟	parenrightex	367
∞	infinity	245	⎠	parenrightbt	370
∫	integral	362	∂	partialdiff	266
⌠	integraltp	363	%	percent	045
⎮	integralex	364	.	period	056

CHAR	NAME	CODE	CHAR	NAME	CODE
⊥	perpendicular	136	~	similar	176
ϕ	phi	146	6	six	066
φ	phi1	152	/	slash	057
π	pi	160		space	040
+	plus	053	♠	spade	252
±	plusminus	261	∍	suchthat	047
∏	product	325	∑	summation	345
⊂	propersubset	314	τ	tau	164
⊃	propersuperset	311	∴	therefore	134
∝	proportional	265	θ	theta	161
ψ	psi	171	ϑ	theta1	112
?	question	077	3	three	063
√	radical	326	™	trademarksans	344
‾	radicalex	140	™	trademarkserif	324
⊆	reflexsubset	315	2	two	062
⊇	reflexsuperset	312	_	underscore	137
®	registersans	342	∪	union	310
®	registerserif	322	∀	universal	042
ρ	rho	162	υ	upsilon	165
″	second	262	℘	weierstrass	303
;	semicolon	073	ξ	xi	170
7	seven	067	0	zero	060
σ	sigma	163	ζ	zeta	172
ς	sigma1	126			

D.6 ZapfDingbats Set and Encoding

CHAR	NAME	CODE	CHAR	NAME	CODE	CHAR	NAME	CODE	CHAR	NAME	CODE
	space	040	✣	a30	103	❆	a65	146	♠	a109	253
✁	a1	041	✤	a31	104	❇	a66	147	①	a120	254
✂	a2	042	✥	a32	105	❈	a67	150	②	a121	255
✃	a202	043	✦	a33	106	❉	a68	151	③	a122	256
✄	a3	044	✧	a34	107	❊	a69	152	④	a123	257
☎	a4	045	★	a35	110	❋	a70	153	⑤	a124	260
✆	a5	046	✩	a36	111	●	a71	154	⑥	a125	261
✇	a119	047	✪	a37	112	❍	a72	155	⑦	a126	262
✈	a118	050	✫	a38	113	■	a73	156	⑧	a127	263
✉	a117	051	✬	a39	114	❏	a74	157	⑨	a128	264
☛	a11	052	✭	a40	115	❐	a203	160	⑩	a129	265
☞	a12	053	✮	a41	116	❑	a75	161	❶	a130	266
✌	a13	054	✯	a42	117	❒	a204	162	❷	a131	267
✍	a14	055	✰	a43	120	▲	a76	163	❸	a132	270
✎	a15	056	✱	a44	121	▼	a77	164	❹	a133	271
✏	a16	057	✲	a45	122	◆	a78	165	❺	a134	272
✐	a105	060	✳	a46	123	❖	a79	166	❻	a135	273
✑	a17	061	✴	a47	124	◗	a81	167	❼	a136	274
✒	a18	062	✵	a48	125	❘	a82	170	❽	a137	275
✓	a19	063	✶	a49	126	❙	a83	171	❾	a138	276
✔	a20	064	✷	a50	127	❚	a84	172	❿	a139	277
✕	a21	065	✸	a51	130	❛	a97	173	➀	a140	300
✖	a22	066	✹	a52	131	❜	a98	174	➁	a141	301
✗	a23	067	✺	a53	132	❝	a99	175	➂	a142	302
✘	a24	070	✻	a54	133	❞	a100	176	➃	a143	303
✙	a25	071	✼	a55	134	❡	a101	241	➄	a144	304
✚	a26	072	✽	a56	135	❢	a102	242	➅	a145	305
✛	a27	073	✾	a57	136	❣	a103	243	➆	a146	306
✜	a28	074	✿	a58	137	❤	a104	244	➇	a147	307
✝	a6	075	❀	a59	140	❥	a106	245	➈	a148	310
✞	a7	076	❁	a60	141	❦	a107	246	➉	a149	311
✟	a8	077	❂	a61	142	❧	a108	247	➊	a150	312
✠	a9	100	❃	a62	143	♣	a112	250	➋	a151	313
✡	a10	101	❄	a63	144	♦	a111	251	➌	a152	314
✢	a29	102	❅	a64	145	♥	a110	252	➍	a153	315

CHAR	NAME	CODE	CHAR	NAME	CODE	CHAR	NAME	CODE	CHAR	NAME	CODE
❺	a154	316	➚	a192	332	➦	a176	346	➳	a184	363
❻	a155	317	➛	a166	333	➧	a177	347	➴	a197	364
❼	a156	320	➜	a167	334	➨	a178	350	➵	a185	365
❽	a157	321	➝	a168	335	➩	a179	351	➶	a194	366
❾	a158	322	➞	a169	336	➪	a193	352	➷	a198	367
❿	a159	323	➟	a170	337	➫	a180	353	➸	a186	370
➔	a160	324	➠	a171	340	➬	a199	354	➹	a195	371
→	a161	325	➡	a172	341	➭	a181	355	➺	a187	372
↔	a163	326	➢	a173	342	➮	a200	356	➻	a188	373
↕	a164	327	➣	a162	343	➯	a182	357	➼	a189	374
➘	a196	330	➤	a174	344	➱	a201	361	➽	a190	375
➙	a165	331	➥	a175	345	➲	a183	362	➾	a191	376

Annex E
(normative)

PDF Name Registry

E.1 General

This annex discusses a registry for developers, controlled by ISO and currently provided by Adobe on behalf of ISO. The registry contains private names and formats that may be used by conforming writers. Developer prefixes shall be used to identify extensions to PDF that use First Class names (see below) and that are intended for public use. (See 7.12.2, “Developer Extensions Dictionary.) “Developers” means any entity including individuals, companies, non-profits, standards bodies, open source groups, etc., who are developing standards or software to use and extend ISO 32000-1.

Private data may be added to PDF documents that enable conforming reader’s to change behavior based on this data. At the same time, users have certain expectations when opening a PDF document, no matter which conforming reader is being used. PDF enforces certain restrictions on private data in order to meet these expectations.

A conforming writer or conforming reader may define new types of actions, destinations, annotations, security, and file system handlers. If a user opens a PDF document using a conforming reader for which the new type of object is not supported, the conforming reader shall behave as described in Annex I.

A conforming writer may also add keys to any PDF object that is implemented as a dictionary, except the file trailer dictionary (see 7.5.5, "File Trailer"). In addition, a conforming writer may create tags that indicate the role of marked-content operators *(PDF 1.2)*, as described in 14.6, "Marked Content".

E.2 Name Registry

To avoid conflicts with third-party names and with future versions of PDF, ISO maintains a registry for certain private names and formats. Developers shall only add private data that conforms to the registry rules. The registry includes three classes:

- *First class*. Names and data formats that are of value to a wide range of developers. All names defined in this ISO 32000 specification are first-class names. Conforming readers that are publicly available shall use first-class names for their private data. First-class names and data formats shall be registered with ISO and shall be made available for all developers to use. To submit a private name and format for consideration as first-class, see the link on registering a private PDF extension, at the following Web page:

  <http://adobe.com/go/ISO32000Registry>

  Data format descriptions shall follow the style of ISO 32000-1 and give a complete specification of the intended function of the extended information.

- *Second class*. Names that are applicable to a specific developer. ISO does not register second-class data formats.) ISO distributes second-class names by registering developer-specific 4-byte prefixes. Those bytes followed by a LOW LINE (5Fh) shall be used as the first characters in the names of all private data added by the developer. ISO shall not register the same prefix to two different developers, thereby ensuring that different developers’ second-class names shall not conflict. It is the responsibility of the developer not to use the same name in conflicting ways. To register a developer-specific prefix, use the following Web page:

  <http://adobe.com/go/ISO32000Registry>

- *Third class*. Names that may be used only in PDF files that other third parties will never see because they can conflict with third-class names defined by others. Third-class names shall all begin with a specific prefix reserved for private extensions. This prefix, which is XX, shall be used as the first characters in the names of all private data added by the developer. It is not necessary to contact ISO to register third-class names.

New keys for the document information dictionary (see 14.3.3, "Document Information Dictionary") or a thread information dictionary (in the ***I*** entry of a thread dictionary; see Section 12.4.3, “Articles”) shall not be registered.

More information about developer prefixes, handlers and extensions to ISO 32000-1 can be obtained at http://www.aiim.org/ISO32000Registry.

Annex F
(normative)

Linearized PDF

F.1 General

Linearization of PDF is an optional feature available beginning in PDF 1.2 that enables efficient incremental access of the file in a network environment. A conforming reader that does not support this optional feature can still successfully process linearized files although not as efficiently. Enhanced conforming readers can recognize that a PDF file has been linearized and may take advantage of that organization (as well as added hint information) to enhance viewing performance.

The primary goal for a linearized PDF file is to achieve the following behaviour for documents of arbitrary size and so that the total number of pages in the document should have little or no effect on the user-perceived performance of viewing any particular page:

- When a document is opened, display the first page as quickly as possible. The first page to be viewed may be an arbitrary page of the document, not necessarily page 0 (though opening at page 0 is most common).
- When the user requests another page of an open document (for example, by going to the next page or by following a link to an arbitrary page), display that page as quickly as possible.
- When data for a page is delivered over a slow channel, display the page incrementally as it arrives. To the extent possible, display the most useful data first.
- Permit user interaction, such as following a link, to be performed even before the entire page has been received and displayed.

NOTE A linearized PDF is optimized for viewing of read-only PDF documents. A linearized PDF should be generated once and read many times.

Incremental update shall still be permitted, but the resulting PDF is no longer linearized and subsequently shall be treated as ordinary PDF. Linearizing it again may require reprocessing the entire file; see G.7, "Accessing an Updated File" for details.

Linearized PDF requires two additions to the PDF specification:

- Rules for the ordering of objects in the PDF file
- Additional optional data structures, called *hint tables*, that enable efficient navigation within the document

Both of these additions are relatively simple to describe; however, using them effectively requires a deeper understanding of their purpose. Consequently, this annex goes considerably beyond a simple specification of these PDF extensions to include background, motivation, and strategies.

- F.2, "Background and Assumptions," provides background information about the properties of the Web that are relevant to the design of Linearized PDF.
- F.3, "Linearized PDF Document Structure," specifies the file format and object-ordering requirements of Linearized PDF.
- F.4, "Hint Tables," specifies the detailed representation of the hint tables.

- Annex G, outlines strategies for accessing Linearized PDF over a network, which in turn determine the optimal way to organize the PDF file.

The reader is assumed to be familiar with the basic architecture of the Web, including terms such as URL, HTTP, and MIME.

F.2 Background and Assumptions

NOTE 1 The principal problem addressed by the Linearized PDF design is the access of PDF documents through the Web. This environment has the following important properties:

- The access protocol (HTTP) is a transaction consisting of a request and a response. The conforming reader presents a request in the form of a URL, and the server sends a response consisting of one or more MIME-tagged data blocks.
- After a transaction has completed, obtaining more data requires a new request-response transaction. The connection between conforming reader and server does not ordinarily persist beyond the end of a transaction, although some implementations may attempt to cache the open connection to expedite subsequent transactions with the same server.
- Round-trip delay can be significant. A request-response transaction can take up to several seconds, independent of the amount of data requested.
- The data rate may be limited. A typical bottleneck is a slow link between the conforming reader and the Internet service provider.

These properties are generally shared by other wide-area network architectures besides the Web. Also, CD-ROMs share some of these properties, since they have relatively slow seek times and limited data rates compared to magnetic media. The remainder of this annex focuses on the Web.

Some additional properties of the HTTP protocol are relevant to the problem of accessing PDF files efficiently. These properties may not all be shared by other protocols or network environments.

- When a PDF file is initially accessed (such as by following a URL hyperlink from some other document), the file type is not known to the conforming reader. Therefore, the conforming reader initiates a transaction to retrieve the entire document and then inspects the MIME tag of the response as it arrives. Only at that point is the document known to be PDF. Additionally, with a properly configured server environment, the length of the document becomes known at that time.
- The conforming reader may abort a response while the transaction is still in progress if it decides that the remainder of the data is not of immediate interest. In HTTP, aborting the transaction requires closing the connection, which interferes with the strategy of caching the open connection between transactions.
- The conforming reader may request retrieval of portions of a document by specifying one or more byte ranges (by offset and count) in the HTTP request headers. Each range can be relative to either the beginning or the end of the file. The conforming reader may specify as many ranges as it wants in the request, and the response consists of multiple blocks, each properly tagged.
- The conforming reader may initiate multiple concurrent transactions in an attempt to obtain multiple responses in parallel. This is commonly done, for instance, to retrieve inline images referenced from an HTML document. This strategy is not always reliable and may backfire if the transactions interfere with each other by competing for scarce resources in the server or the communication channel.

NOTE 2 Extensive experimentation has determined that having multiple concurrent transactions does not work very well for PDF in some important environments. Therefore, Linearized PDF is designed to enable good performance to be achieved using only one transaction at a time. In particular, this means that the conforming reader needs to have sufficient information to determine the byte ranges for all the objects required to display a given page of the PDF file so that it can specify all those byte ranges in a single request.

NOTE 3 The following additional assumptions are made about the conforming reader and its local environment:

- The conforming reader has plenty of local temporary storage available. It should rarely need to retrieve a given portion of a PDF document more than once from the server.

- The conforming reader is able to display PDF data quickly once it has been received. The performance bottleneck is assumed to be in the transport system (throughput or round-trip delay), not in the processing of data after it arrives.

The consequence of these assumptions is that it may be advantageous for the conforming reader to do considerable extra work to minimize delays due to communications.

Such work includes maintaining local caches and reordering actions according to when the needed data becomes available.

F.3 Linearized PDF Document Structure

F.3.1 General

Except as noted below, all elements of a Linearized PDF file shall be as specified in 7.5, "File Structure", and all indirect objects in the file shall be shall be divided into two groups.

- The first group shall consist of the document catalogue, other document-level objects, and all objects belonging to the first page of the document. These objects shall be numbered sequentially, starting at the first object number after the last number of the second group. (The stream containing the hint tables, called a *hint stream*, may be numbered out of sequence; see F.3.6, "Hint Streams (Parts 5 and 10)".

- The second group shall consist of all remaining objects in the document, including all pages after the first, all shared objects (objects referenced from more than one page, not counting objects referenced from the first page), and so forth. These objects shall be numbered sequentially starting at 1.

These groups of objects shall be indexed by exactly two cross-reference table sections. For pedagogical reasons the linearized PDF is considered to be composed from 11 parts, in order, and the composition of these groups is discussed in more detail in the sections that follow. All objects shall have a generation number of 0.

Beginning with PDF 1.5, PDF files may contain object streams (see 7.5.7, "Object Streams"). In linearized files containing object streams, the following conditions shall apply:

- These additional objects may not be contained in an object stream: the linearization dictionary, the document catalogue, and page objects.

- Objects stored within object streams shall be given the highest range of object numbers within the main and first-page cross-reference sections.

- For files containing object streams, hint data may specify the location and size of the object streams only (or uncompressed objects), not the individual compressed objects. Similarly, shared object references shall be made to the object stream containing a compressed object, not to the compressed object itself.

- Cross-reference streams (7.5.8, "Cross-Reference Streams") may be used in place of traditional cross-reference tables. The logic described in this sub-clause shall still apply, with the appropriate syntactic changes.

EXAMPLE 1 **Part 1: Header**

```
%PDF-1.1        % …Binary characters…
```

EXAMPLE 2 **Part 2: Linearization parameter dictionary**

```
43 0 obj
   << /Linearized 1.0          % Version
      /L 54567                 % File length
      /H [475 598]             % Primary hint stream offset and length (part 5)
      /O 45                    % Object number of first page's page object (part 6)
      /E 5437                  % Offset of end of first page
      /N 11                    % Number of pages in document
```

```
        /T  52786               % Offset of first entry in main cross-reference table (part 11)
    >>
endobj
```

EXAMPLE 3 **Part 3: First-page cross-reference table and trailer**

```
xref
43  14
0000000052  00000  n
0000000392  00000  n
0000001073  00000  n
…Cross-reference entries for remaining objects in the first page…
0000000475  00000  n
trailer
    <<  /Size  57               % Total number of cross-reference table entries in document
        /Prev  52776            % Offset of main cross-reference table (part 11)
        /Root  44 0 R           % Indirect reference to catalogue (part 4)
        …Any other entries, such as Info and Encrypt…        % (part 9)
    >>
% Dummy cross-reference table offset
startxref
0
%%EOF
```

EXAMPLE 4 **Part 4: Document catalogue and other required document-level objects**

```
44  0  obj
    <<  /Type  /Catalog
        /Pages  42 0 R
    >>
endobj
…Other objects…
```

EXAMPLE 5 **Part 5: Primary hint stream (may precede or follow part 6)**

```
56  0  obj
    <<  /Length  457
        …Possibly other stream attributes, such as Filter…
        /S  221                 % Position of shared object hint table
        …Possibly entries for other hint tables…
    >>
stream
    …Page offset hint table…
    …Shared object hint table…
    …Possibly other hint tables…
endstream
endobj
```

EXAMPLE 6 **Part 6: First-page section (may precede or follow part 5)**

```
45  0  obj
    <<  /Type  /Page
        …
    >>
endobj

…Outline hierarchy (if the PageMode value in the document catalog is UseOutlines)…

…Objects for first page, including both shared and nonshared objects…
```

EXAMPLE 7 **Part 7: Remaining pages**

```
1  0  obj
    <<  /Type  /Page
```

```
        …Other page attributes, such as MediaBox, Parent, and Contents…
    >>
endobj

…Nonshared objects for this page…

…Each successive page followed by its nonshared objects…

…Last page followed by its nonshared objects…
```

EXAMPLE 8 **Part 8: Shared objects for all pages except the first**

```
…Shared objects…
```

EXAMPLE 9 **Part 9: Objects not associated with pages, if any**

```
…Other objects…
```

EXAMPLE 10 **Part 10: Overflow hint stream (optional)**

```
…Overflow hint stream…
```

EXAMPLE 11 **Part 11: Main cross-reference table and trailer**

```
xref
0  43
0000000000  65535  f
…Cross-reference entries for all except first page's objects…
trailer
    <<  /Size  43  >>               % Trailer need not contain other entries; in particular,
% it should not have a Prev entry

% Offset of first-page cross-reference table (part 3)
startxref
257
%%EOF
```

F.3.2 Header (Part 1)

The Linearized PDF file shall begin with the standard header line (see 7.5.2, "File Header"). Linearization is independent of PDF version number and may be applied to any PDF file of version 1.1 or greater.

The binary characters following the PERCENT SIGN (25h) on the second line are characters with codes 128 or greater, as recommended in 7.5.2, "File Header".

F.3.3 Linearization Parameter Dictionary (Part 2)

Following the header, the first object in the body of the file (part 2) shall be an indirect dictionary object, the *linearization parameter dictionary*, which shall contain the parameters listed in Table F.1. All values in this dictionary shall be direct objects. There shall be no references to this dictionary anywhere in the document; however, the first-page cross-reference table (part 3) shall contain a normal entry for it.

The linearization parameter dictionary shall be entirely contained within the first 1024 bytes of the PDF file. This limits the amount of data a conforming reader must read before deciding whether the file is linearized.

Table F.1 – Entries in the linearization parameter dictionary

Parameter	Type	Value
Linearized	number	*(Required)* A version identification for the linearized format.
L	integer	*(Required)* The length of the entire file in bytes. It shall be exactly equal to the actual length of the PDF file. A mismatch indicates that the file is not linearized and shall be treated as ordinary PDF, ignoring linearization information. (If the mismatch resulted from appending an update, the linearization information may still be correct but requires validation; see G.7, "Accessing an Updated File" for details.)
H	array	*(Required)* An array of two or four integers, [$offset_1$ $length_1$] or [$offset_1$ $length_1$ $offset_2$ $length_2$]. $offset_1$ shall be the offset of the primary hint stream from the beginning of the file. (This is the beginning of the stream object, not the beginning of the stream data.) $length_1$ shall be the length of this stream, including stream object overhead. If the value of the primary hint stream dictionary's **Length** entry is an indirect reference, the object it refers to shall immediately follow the stream object, and $length_1$ also shall include the length of the indirect length object, including object overhead. If there is an overflow hint stream, $offset_2$ and $length_2$ shall specify its offset and length.
O	integer	*(Required)* The object number of the first page's page object.
E	integer	*(Required)* The offset of the end of the first page (the end of EXAMPLE 6 in F.3.1, "General"), relative to the beginning of the file.
N	integer	*(Required)* The number of pages in the document.
T	integer	*(Required)* In documents that use standard main cross-reference tables (including hybrid-reference files; see 7.5.8.4, "Compatibility with Applications That Do Not Support Compressed Reference Streams"), this entry shall represent the offset of the white-space character preceding the first entry of the main cross-reference table (the entry for object number 0), relative to the beginning of the file. Note that this differs from the **Prev** entry in the first-page trailer, which gives the location of the **xref** line that precedes the table. *(PDF 1.5)* Documents that use cross-reference streams exclusively (see 7.5.8, "Cross-Reference Streams"), this entry shall represent the offset of the main cross-reference stream object.
P	integer	*(Optional)* The page number of the first page; see F.3.4, "First-Page Cross-Reference Table and Trailer (Part 3)". Default value: 0.

F.3.4 First-Page Cross-Reference Table and Trailer (Part 3)

Part 3 shall contain the cross-reference table for objects belonging to the first page (discussed in F.3.4, "First-Page Cross-Reference Table and Trailer (Part 3)") as well as for the document catalogue and document-level objects appearing before the first page (discussed in F.3.5, "Document Catalogue and Document-Level Objects (Part 4)"). Additionally, this cross-reference table shall contain entries for the linearization parameter dictionary (at the beginning) and the primary hint stream (at the end). This table shall be a valid cross-reference table as defined in 7.5.4, "Cross-Reference Table", although its position in the file shall not be at the end of the file. It shall consist of a single cross-reference subsection that has no free entries.

In PDF 1.5 and later, cross-reference streams (see 7.5.8, "Cross-Reference Streams") may be used in linearized files in place of traditional cross-reference tables. The logic described in this section, along with the appropriate syntactic changes for cross-reference streams shall still apply.

Below the table shall be the first-page trailer. The trailer’s **Prev** entry shall give the offset of the main cross-reference table near the end of the file. A conforming reader that does not support the linearized feature shall process this correctly even though the trailers are linked in an unusual order. It interprets the first-page cross-reference table as an update to an original document that is indexed by the main cross-reference table.

The first-page trailer shall contain valid **Size** and **Root** entries, as well as any other entries needed to display the document. The **Size** value shall be the combined number of entries in both the first-page cross-reference table and the main cross-reference table.

The first-page trailer may optionally end with **startxref**, an integer, and %%EOF, just as in an ordinary trailer. This information shall be ignored.

F.3.5 Document Catalogue and Document-Level Objects (Part 4)

Following the first-page cross-reference table and trailer are the catalogue dictionary and other objects that are required present when the document is opened. These additional objects (constituting part 4) shall include the values of the following entries if they are present and are indirect objects:

- The conforming reader **Preferences** entry in the catalogue.
- The **PageMode** entry in the catalogue. Note that if the value of **PageMode** is UseOutlines, the outline hierarchy shall be located in part 6; otherwise, the outline hierarchy, if any, shall be located in part 9. See F.3.10, "Other Objects (Part 9)" for details.
- The **Threads** entry in the catalogue, along with all thread dictionaries it refers to. This does not include the threads’ information dictionaries or the individual bead dictionaries belonging to the threads.
- The **OpenAction** entry in the catalogue.
- The **AcroForm** entry in the catalogue. Only the top-level interactive form dictionary shall be present, not the objects that it refers to.
- The **Encrypt** entry in the first-page trailer dictionary. All values in the encryption dictionary shall also be located here.

All other objects shall not be located here but instead shall be at the end of the file; see F.3.10, "Other Objects (Part 9)". This includes objects such as page tree nodes, the document information dictionary, and the definitions for named destinations.

NOTE The objects located here are indexed by the first-page cross-reference table, even though they are not logically part of the first page.

F.3.6 Hint Streams (Parts 5 and 10)

The core of the linearization information shall be stored in data structures known as *hint tables*, whose format is described in F.4, "Hint Tables." They shall provide indexing information that enables the conforming reader to construct a single request for all the objects that are needed to display any page of the document or to retrieve other information efficiently. The hint tables may contain additional information to optimize access by conforming writer extensions to application-specific data.

The hint tables shall not be logically part of the information content of the document; they shall be derived from the document. Any action that changes the document—for instance, appending an incremental update—invalidates the hint tables. The document remains a valid PDF file but is no longer linearized; see G.7, "Accessing an Updated File" for details.

The hint tables are binary data structures that shall be enclosed in a stream object. Syntactically, this stream shall be a PDF indirect object. However, there shall be no references to the stream anywhere in the document.

Therefore, it is not logically part of the document, and an operation that regenerates the document may remove the stream.

Usually, all the hint tables shall be contained in a single stream, known as the *primary hint stream*. Optionally, there may be an additional stream containing more hints, known as the *overflow hint stream*. The contents of the two hint streams shall be concatenated and treated as if they were a single unbroken stream.

The primary hint stream, which shall be required, is shown as part 5 in Example 5. The order of this part and the first-page section, shown as part 6, may be reversed; see Annex G for considerations on the choice of placement. The overflow hint stream, part 10, is optional.

The location and length of the primary hint stream, and of the overflow hint stream if present, shall be given in the linearization parameter dictionary at the beginning of the file.

The hint streams shall be assigned the last object numbers in the file—that is, after the object number for the last object in the first page. Their cross-reference table entries shall be at the end of the first-page cross-reference table. This object number assignment shall be independent of the physical locations of the hint streams in the file.

NOTE This convention keeps their object numbers from conflicting with the numbering of the linearized objects.

With one exception, the values of all entries in the hint streams' dictionaries shall be direct objects and may contain no indirect object references. The exception is the stream dictionary's **Length** entry (see the discussion of the **H** entry in Table F.1).

In addition to the standard stream attributes, the dictionary of the primary hint stream shall contain entries giving the position of the beginning of each hint table in the stream. These positions shall be counted in bytes relative to the beginning of the stream data (after decoding filters, if any, are applied) and with the overflow hint stream concatenated if present. The dictionary of the overflow hint stream shall not contain these entries. The keys designating the standard hint tables in the primary hint stream's dictionary are listed in Table F.2; F.4, "Hint Tables," documents the format of these hint tables. Additionally, there is a required page offset hint table, which shall be the first table in the stream and shall start at offset 0.

Table F.2 – Standard hint tables

Key	**Hint table**
S	*(Required)* Shared object hint table (see F.4.2, “Shared Object Hint Table”)
T	*(Present only if thumbnail images exist)* Thumbnail hint table (see F.4.3, "Thumbnail Hint Table")
O	*(Present only if a document outline exists)* Outline hint table (see F.4.4, “Generic Hint Tables”)
A	*(Present only if article threads exist)* Thread information hint table (see F.4.4, “Generic Hint Tables”)
E	*(Present only if named destinations exist)* Named destination hint table (see F.4.4, “Generic Hint Tables”)
V	*(Present only if an interactive form dictionary exists)* Interactive form hint table (see F.4.5, “Extended Generic Hint Tables”)
I	*(Present only if a document information dictionary exists)* Information dictionary hint table (see F.4.4, “Generic Hint Tables”)
C	*(Present only if a logical structure hierarchy exists; PDF 1.3)* Logical structure hint table (see F.4.5, “Extended Generic Hint Tables”)
L	*(PDF 1.3)* Page label hint table (see F.4.4, “Generic Hint Tables”)

Table F.2 – Standard hint tables (continued)

Key	Hint table
R	*(Present only if a renditions name tree exists; PDF 1.5)* Renditions name tree hint table (see F.4.5, “Extended Generic Hint Tables”)
B	*(Present only if embedded file streams exist; PDF 1.5)* Embedded file stream hint table (see F.4.6, “Embedded File Stream Hint Tables”)

New keys may be registered for additional hint tables required application-specific data accessed by conforming writer extensions. See Annex E for further information.

F.3.7 First-Page Section (Part 6)

This part of the file contains all the objects needed to display the first page of the document. Ordinarily, the first page is page 0—that is, the leftmost leaf page node in the page tree. However, if the document catalogue contains an **OpenAction** entry that specifies opening at some page other than page 0, that page shall be considered the first page and shall be located here. The page number of the first page is given in the **P** entry of the linearization parameter dictionary.

NOTE As mentioned earlier, the section containing objects belonging to the first page of the document may either precede or follow the primary hint stream. The starting file offset and length of this section can be determined from the hint tables. In addition, the **E** entry in the linearization parameter dictionary specifies the end of the first page (as an offset relative to the beginning of the file), and the **O** entry gives the object number of the first page’s page object.

The following objects shall be contained in the first-page section:

- The page object for the first page. This object shall be the first one in this part of the file. Its object number is given in the linearization parameter dictionary. This page object shall explicitly specify all required attributes, such as **Resources** and **MediaBox**; the attributes may not be inherited from ancestor page tree nodes.

- The entire outline hierarchy, if the value of the **PageMode** entry in the catalogue is UseOutlines. If the **PageMode** entry is omitted or has some other value and the document has an outline hierarchy, the outline hierarchy shall appear in part 9; see F.3.10, "Other Objects (Part 9)" for details.

- All objects that the page object refers to, to an arbitrary depth, except page tree nodes or other page objects. This shall include objects referred to by its **Contents**, **Resources**, **Annots**, and **B** entries, but not the **Thumb** entry.

The order of objects referenced from the page object should facilitate early user interaction and incremental display of the page data as it arrives. The following order should be used:

a) The **Annots** array and all annotation dictionaries, to a depth sufficient for those annotations to be activated. Information required to draw the annotation may be deferred until later since annotations are always drawn on top of (hence after) the contents.

b) The **B** (beads) array and all bead dictionaries, if any, for this page. If any beads exist for this page, the **B** array shall be present in the page dictionary. Additionally, each bead in the thread (not just the first bead) shall contain a **T** entry referring to the associated thread dictionary.

c) The resource dictionary, but not the resource objects contained in the dictionary.

d) Resource objects, other than the types listed below, in the order that they are first referenced (directly or indirectly) from the content stream. If the contents are represented as an array of streams, each resource object shall precede the stream in which it is first referenced. Note that **Font**, **FontDescriptor**, and **Encoding** resources shall be included here, but not substitutable font files referenced from font descriptors (see item (g) below).

e) The page contents (**Contents**). If large, this should be represented as an array of indirect references to content streams, which in turn shall be interleaved with the resources they require. If small, the entire contents should be a single content stream preceding the resources.

f) Image XObjects, in the order that they are first referenced. Images can be assumed to be large and slow to transfer; therefore, the conforming reader should defer rendering images until all the other contents have been displayed.

g) **FontFile** streams, which contain the actual definitions of embedded fonts. These can be assumed to be large and slow to transfer; therefore, the conforming reader should use substitute fonts until the real ones have arrived. Only those fonts for which substitution is possible may be deferred in this way. (Currently, this includes any Type 1 or TrueType font that has a font descriptor with the Nonsymbolic flag set, indicating the Adobe standard Latin character set).

See Annex G for additional discussion about object order and incremental drawing strategies.

F.3.8 Remaining Pages (Part 7)

Part 7 of the Linearized PDF file shall contain the page objects and nonshared objects for all remaining pages of the file, with the objects for each page grouped together. The pages shall be contiguous and shall be ordered by page number. If the first page of the file is not page 0, this section shall start with page 0 and shall skip over the first page when its position in the sequence is reached.

For each page, the objects required to display that page shall be grouped together, except for resources and other objects that are shared with other pages. Shared objects shall be located in the shared objects section (part 8). The starting file offset and length of any page can be determined from the hint tables.

The recommended order of objects within a page is essentially the same as in the first page. In particular, the page object shall be the first object in each section.

In most cases, unlike for the first page, little benefit is gained from interleaving contents with resources because most resources other than images—fonts in particular—are shared among multiple pages and therefore reside in the shared objects section. Image XObjects usually are not shared, but they should appear at the end of the page’s section of the file, since rendering of images is deferred.

F.3.9 Shared Objects (Part 8)

Part 8 of the file contains objects, primarily named resources, that are referenced from more than one page but that are not referenced (directly or indirectly) from the first page. The hint tables contain an index of these objects. For more information on named resources, see 7.8.3, "Resource Dictionaries".

The order of these objects shall be arbitrary. However, wherever a resource consists of a multiple-level structure, all components of the structure shall be grouped together. If only the top-level object is referenced from outside the group, the entire group may be described by a single entry in the shared object hint table. This helps to minimize the size of the shared object hint table and the number of individual references from entries in the page offset hint table.

F.3.10 Other Objects (Part 9)

Following the shared objects are any other objects that are part of the document but are not required for displaying pages. These objects shall be divided into functional categories. Objects within each of these categories should be grouped together; the relative order of the categories is unimportant.

• *The page tree*. This object can be located in this section because the conforming reader never needs to consult it. Note that all **Resources** attributes and other inheritable attributes of the page objects shall be pushed down and replicated in each of the leaf page objects (but they may contain indirect references to shared objects).

- *Thumbnail images*. These objects shall simply be ordered by page number. (The thumbnail image for page 0 shall be first, even if the first page of the document is some page other than 0.) Each thumbnail image consists of one or more objects, which may refer to objects in the thumbnail shared objects section (see the next item).

- *Thumbnail shared objects*. These are objects that shall be shared among some or all thumbnail images and shall not be referenced from any other objects.

- *The outline hierarchy*, if not located in part 6. The order of objects shall be the same as the order in which they shall be displayed by the conforming reader. This is a preorder traversal of the outline tree, skipping over any subtree that is closed (that is, whose parent's **Count** value is negative). Following that shall be the subtrees that were skipped over, in the order in which they would have appeared if they were all open.

- *Thread information dictionaries*, referenced from the **I** entries of thread dictionaries. Note that the thread dictionaries themselves shall be located with the document catalogue and the bead dictionaries with the individual pages.

- *Named destinations*. These objects include the value of the **Dests** or **Names** entry in the document catalogue and all the destination objects that it refers to; see G.3, "Opening at an Arbitrary Page".

- *The document information dictionary* and the objects contained within it.

- *The interactive form field hierarchy*. This group of objects shall not include the top-level interactive form dictionary, which is located with the document catalogue.

- *Other entries* in the document catalogue that are not referenced from any page.

- *(PDF 1.3) The logical structure hierarchy*.

- *(PDF 1.5) The renditions name tree hierarchy.*

- *(PDF 1.5) Embedded file streams.*

F.3.11 Main Cross-Reference and Trailer (Part 11)

Part 11 is the cross-reference table for all objects in the PDF file except those listed in the first-page cross-reference table (part 3). As indicated earlier, this cross-reference table shall play the role of the original cross-reference table for the file (before any updates are appended) and shall conform to the following rules:

- It consists of a single cross-reference subsection, beginning at object number 0.

- The first entry (for object number 0) shall be a free entry.

- The remaining entries are for in-use objects, which shall be numbered consecutively, starting at 1.

The **startxref** line shall give the offset of the first-page cross-reference table. The **Prev** entry of the first-page trailer shall give the offset of the main cross-reference table. The main trailer has no **Prev** entry and shall not contain any entries other than **Size**.

In PDF 1.5 and later, cross-reference streams (see 7.5.8, "Cross-Reference Streams") may be used in linearized files in place of traditional cross-reference tables. The logic described in this sub-clause, along with the appropriate syntactic changes for cross-reference streams, still applies.

F.4 Hint Tables

The core of the linearization information shall be stored in two or more hint tables, as indicated by the attributes of the primary hint stream; see F.3.6, "Hint Streams (Parts 5 and 10)". The format of the standard hint tables is described in this section.

A conforming writer may add additional hint tables for conforming reader-specific data. A generic format for such hint tables is defined; see F.4.4, "Generic Hint Tables." Alternatively, the format of a hint table can be private to the conforming reader; see Annex E for further information.

Each hint table shall consist of a portion of the stream, beginning at the position in the stream indicated by the corresponding stream attribute. Additionally, a conforming writer shall include a page offset hint table, which shall be the first table in the stream and shall start at offset 0. If there is an overflow hint stream, its contents shall be appended seamlessly to the primary hint stream.

NOTE 1 Hint table positions are relative to the beginning of this combined stream.

In general, this byte stream shall be treated as a bit stream, high-order bit first, which shall then subdivided into fields of arbitrary width without regard to byte boundaries. However, each hint table shall begin at a byte boundary.

NOTE 2 The hint tables are designed to encode the required information as compactly as possible. Interpreting the hint tables requires reading them sequentially; they are not designed for random access.

The conforming reader shall be expected to read and decode the tables once and retain the information for as long as the document remains open.

NOTE 3 A hint table encodes the positions of various objects in the file. The representation is either explicit (an offset from the beginning of the file) or implicit (accumulated lengths of preceding objects).

Regardless of the representation, the resulting positions shall be interpreted as if the primary hint stream itself were not present. That is, a position greater than the *hint stream offset* shall have the *hint stream length* added to it to determine the actual offset relative to the beginning of the file.

NOTE 4 The hint stream offset and hint stream length are the values offset1 and length1 in the **H** array in the linearization parameter dictionary at the beginning of the file.

The reason for this rule is that the length of the primary hint stream depends on the information contained within the hint tables, which is not known until after they have been generated. Any information contained in the hint tables shall not depend on knowing the primary hint stream's length in advance.

Note that this rule applies only to offsets given in the hint tables and not to offsets given in the cross-reference tables or linearization parameter dictionary. Also, the offset and length of the overflow hint stream, if present, does not be taken into account, since this object follows all other objects in the file.

In linearized files that use object streams (7.5.7, "Object Streams"), the position specified in a hint table for a compressed object is to be interpreted as a byte range in which the object can be found, not as a precise offset. conforming readers should locate the object via a cross-reference stream, as it would if the hint table were not present.

F.4.1 Page Offset Hint Table

The page offset hint table provides information required for locating each page. Additionally, for each page except the first, it also enumerates all shared objects that the page references, directly or indirectly.

This table shall begin with a header section, described in Table F.3, followed by one or more per-page entries, described in Table F.4.

NOTE The items making up each per-page entry are not contiguous; they are broken up with items from entries for other pages.

The order of items making up the per-page entries shall be as follows:

a) Item 1 for all pages, in page order starting with the first page

b) Item 2 for all pages, in page order starting with the first page

c) Item 3 for all pages, in page order starting with the first page

d) Item 4 for all shared objects in the second page, followed by item 4 for all shared objects in the third page, and so on

e) Item 5 for all shared objects in the second page, followed by item 5 for all shared objects in the third page, and so on

f) Item 6 for all pages, in page order starting with the first page

g) Item 7 for all pages, in page order starting with the first page

All the items in Table F.3 that specify a number of bits needed, such as item 3, have values in the range 0 through 32. Although that range requires only 6 bits, 16-bit numbers shall be used.

Table F.3 – Page offset hint table, header section

Item	Size (bits)	Description
1	32	The least number of objects in a page (including the page object itself).
2	32	The location of the first page's page object.
3	16	The number of bits needed to represent the difference between the greatest and least number of objects in a page.
4	32	The least length of a page in bytes. This shall be the least length from the beginning of a page object to the last byte of the last object used by that page.
5	16	The number of bits needed to represent the difference between the greatest and least length of a page, in bytes.
6	32	The least offset of the start of any content stream, relative to the beginning of its page.
7	16	The number of bits needed to represent the difference between the greatest and least offset to the start of the content stream.
8	32	The least content stream length.
9	16	The number of bits needed to represent the difference between the greatest and least content stream length.
10	16	The number of bits needed to represent the greatest number of shared object references.
11	16	The number of bits needed to represent the numerically greatest shared object identifier used by the pages (discussed further in Table F.4, item 4).

Table F.3 – Page offset hint table, header section (continued)

Item	Size (bits)	Description
12	16	The number of bits needed to represent the numerator of the fractional position for each shared object reference. For each shared object referenced from a page, there shall be an indication of where in the page's content stream the object is first referenced. That position shall be given as the numerator of a fraction, whose denominator is specified once for the entire document (in the next item in this table). The fraction is explained in more detail in Table F.4, item 5.
13	16	The denominator of the fractional position for each shared object reference.

Table F.4 – Page offset hint table, per-page entry

Item	Size (bits)	Description
1	See Table F.3, item 3	A number that, when added to the least number of objects in a page (Table F.3, item 1), shall give the number of objects in the page. The first object of the first page shall have an object number that is the value of the **O** entry in the linearization parameter dictionary at the beginning of the file. The first object of the second page shall have an object number of 1. Object numbers for subsequent pages shall be determined by accumulating the number of objects in all previous pages.
2	See Table F.3, item 5	A number that, when added to the least page length (Table F.3, item 4), shall give the length of the page in bytes. The location of the first object of the first page may be determined from its object number (the **O** entry in the linearization parameter dictionary) and the cross-reference table entry for that object; see F.3.4, "First-Page Cross-Reference Table and Trailer (Part 3)". The locations of subsequent pages shall be determined by accumulating the lengths of all previous pages. A conforming product shall skip over the primary hint stream, wherever it is located.
3	See Table F.3, item 10	The number of shared objects referenced from the page. For the first page, this number shall be 0; the next two items start with the second page.
4	See Table F.3, item 11	*(One item for each shared object referenced from the page)* A *shared object identifier*—that is, an index into the shared object hint table (described in F.4.2, "Shared Object Hint Table"). A single entry in the shared object hint table may designate a group of shared objects, but only one of which shall be referenced from outside the group. That is, shared object identifiers shall not be directly related to object numbers. This identifier combines with the numerators provided in item 5 to form a *shared object reference*.

Table F.4 – Page offset hint table, per-page entry (continued)

Item	Size (bits)	Description
5	See Table F.3, item 12	*(One item for each shared object referenced from the page)* The numerator of the fractional position for each shared object reference, which shall be in the same order as the preceding item. The fraction shall indicate where in the page's content stream the shared object is first referenced. This item shall be interpreted as the numerator of a fraction whose denominator is specified once for the entire document (Table F.3, item 13). EXAMPLE If the denominator is *d*, a numerator ranging from 0 to $d - 1$ indicates the corresponding portion of the page's content stream. For example, if the denominator is 4, a numerator of 0, 1, 2, or 3 indicates that the first reference lies in the first, second, third, or fourth quarter of the content stream, respectively. There are two (or more) other possible values for the numerator, which shall indicate that the shared object is not referenced from the content stream but instead is needed by annotations or other objects that are drawn after the contents. The value *d* shall indicate that the shared object is needed before image XObjects and other nonshared objects that are at the end of the page. A value of $d + 1$ or greater shall indicate that the shared object is needed after those objects. NOTE This method of dividing the page into fractions is only approximate. Determining the first reference to a shared object entails inspecting the unencoded content stream. The relationship between positions in the unencoded and encoded streams is not necessarily linear.
6	See Table F.3, item 7	A number that, when added to the least offset to the start of the content stream (Table F.3, item 6), shall give the offset in bytes of the start of the page's content stream (the stream object, not the stream data), relative to the beginning of the page.
7	See Table F.3, item 9	A number that, when added to the least content stream length (Table F.3, item 8), shall give the length of the page's content stream in bytes. This length shall include object overhead preceding and following the stream data.

F.4.2 Shared Object Hint Table

The shared object hint table gives information required to locate shared objects; see F.3.9, "Shared Objects (Part 8)". Shared objects may be physically located in either of two places: objects that are referenced from the first page shall be located with the first-page objects (part 6); all other shared objects shall be located in the shared objects section (part 8).

A single entry in the shared object hint table may describe a group of adjacent objects under the following condition: Only the first object in the group is referenced from outside the group; the remaining objects in the group are referenced only from other objects in the same group. The objects in a group shall have adjacent object numbers.

The page offset hint table, interactive form hint table, and logical structure hint table shall refer to an entry in the shared object hint table by a simple index that is its sequential position in the table, counting from 0.

The shared object hint table shall consist of a header section (Table F.5) followed by one or more shared object group entries (Table F.6). There shall be two sequences of shared object group entries: the ones for objects located in the first page, followed by the ones for objects located in the shared objects section. The entries shall have the same format in both cases. Note that the items making up each shared object group entry need not be contiguous; they may be broken up with items from entries for other shared object groups. The order of items in each sequence shall be as follows:

a) Item 1 for the first group, item 1 for the second group, and so on

b) Item 2 for the first group, item 2 for the second group, and so on

c) Item 3 for the first group, item 3 for the second group, and so on

d) Item 4 for the first group, item 4 for the second group, and so on

All objects associated with the first page (part 6) shall have entries in the shared object hint table, regardless of whether they are actually shared. The first entry shall refer to the beginning of the first page and shall be have an object count and length that shall span all the initial nonshared objects. The next entry shall refer to a group of shared objects. Subsequent entries shall span additional groups of either shared or nonshared objects consecutively until all shared objects in the first page have been enumerated. (There shall not be any entries that refer to nonshared objects.)

Table F.5 – Shared object hint table, header section

item	Size (bits)	Description
1	32	The object number of the first object in the shared objects section (part 8).
2	32	The location of the first object in the shared objects section.
3	32	The number of shared object entries for the first page (including nonshared objects, as noted above).
4	32	The number of shared object entries for the shared objects section, including the number of shared object entries for the first page (that is, the value of item 3).
5	16	The number of bits needed to represent the greatest number of objects in a shared object group.
6	32	The least length of a shared object group in bytes.
7	16	The number of bits needed to represent the difference between the greatest and least length of a shared object group, in bytes.

Table F.6 – Shared object hint table, shared object group entry

Item	Size (bits)	Description
1	See Table F.5, item 7	A number that, when added to the least shared object group length (Table F.5, item 6), gives the length of the object group in bytes. The location of the first object of the first page shall be given in the page offset hint table, header section (Table F.3, item 4). The locations of subsequent object groups can be determined by accumulating the lengths of all previous object groups until all shared objects in the first page have been enumerated. Following that, the location of the first object in the shared objects section can be obtained from the header section of the shared object hint table (Table F.5, item 2).
2	1	A flag indicating whether the shared object signature (item 3) is present; its value shall be 1 if the signature is present and 0 if it is absent.

Table F.6 – Shared object hint table, shared object group entry (continued)

Item	Size (bits)	Description
3	128	*(Only if item 2 is 1)* The *shared object signature*, a 16-byte MD5 hash that uniquely identifies the resource that the group of objects represents. NOTE It enables the conforming reader to substitute a locally cached copy of the resource instead of reading it from the PDF file. Note that this signature is unrelated to signature fields in interactive forms, as defined in 12.7.4.5, "Signature Fields".
4	See Table F.5, item 5	A number equal to 1 less than the number of objects in the group. The first object of the first page shall be the one whose object number is given by the **O** entry in the linearization parameter dictionary at the beginning of the file. Object numbers for subsequent entries can be determined by accumulating the number of objects in all previous entries until all shared objects in the first page have been enumerated. Following that, the first object in the shared objects section has a number that can be obtained from the header section of the shared object hint table (Table F.5, item 1).

NOTE In a document consisting of only one page, all of that page’s objects shall be treated as if they were shared; the shared object hint table reflects this.

F.4.3 Thumbnail Hint Table

The thumbnail hint table shall consist of a header section (Table F.7) followed by the thumbnails section, which shall include one or more per-page entries (Table F.8), each of which describes the thumbnail image for a single page. The entries shall be in page number order starting with page 0, even if the document catalogue contains an **OpenAction** entry that specifies opening at some page other than page 0. Thumbnail images may exist for some pages and not for others.

Table F.7 – Thumbnail hint table, header section

Item	Size (bits)	Description
1	32	The object number of the first thumbnail image (that is, the thumbnail image that is described by the first entry in the thumbnails section).
2	32	The location of the first thumbnail image.
3	32	The number of pages that have thumbnail images.
4	16	The number of bits needed to represent the greatest number of consecutive pages that do not have a thumbnail image.
5	32	The least length of a thumbnail image in bytes.
6	16	The number of bits needed to represent the difference between the greatest and least length of a thumbnail image.
7	32	The least number of objects in a thumbnail image.
8	16	The number of bits needed to represent the difference between the greatest and least number of objects in a thumbnail image.
9	32	The object number of the first object in the thumbnail shared objects section (a subsection of part 9). This section includes objects (colour spaces, for example) that shall be referenced from some or all thumbnail objects and are not referenced from any other objects. The thumbnail shared objects shall be undifferentiated; there is no indication of which shared objects shall be referenced from any given page’s thumbnail image.

Table F.7 – Thumbnail hint table, header section (continued)

Item	Size (bits)	Description
10	32	The location of the first object in the thumbnail shared objects section.
11	32	The number of thumbnail shared objects.
12	32	The length of the thumbnail shared objects section in bytes.

Table F.8 – Thumbnail hint table, per-page entry

Item	Size (bits)	Description
1	See Table F.7, item 4	*(Optional)* The number of preceding pages lacking a thumbnail image. This number indicates how many pages without a thumbnail image lie between the previous entry's page and this page.
2	See Table F.7, item 8	A number that, when added to the least number of objects in a thumbnail image (Table F.7, item 7), gives the number of objects in this page's thumbnail image.
3	See Table F.7, item 6	A number that, when added to the least length of a thumbnail image (Table F.7, item 5), gives the length of this page's thumbnail image in bytes.

The order of items in Table F.8 is as follows:

a) Item 1 for all pages, in page order starting with the first page

b) Item 2 for all pages, in page order starting with the first page

c) Item 3 for all pages, in page order starting with the first page

F.4.4 Generic Hint Tables

Categories of objects are associated with the document as a whole rather than with individual pages (see F.3.10, "Other Objects (Part 9)"), and hints should be provided for accessing those objects efficiently. For each category of hints, there shall be a separate entry in the primary hint stream giving the starting position of the table within the stream; see F.3.6, "Hint Streams (Parts 5 and 10)".

Such hints shall be represented by a generic hint table, which describes a single group of objects that are located together in the PDF file. The entries in this table are listed in Table F.9. This representation shall be used for the following hint tables, if needed:

- Outline hint table
- Thread information hint table
- Named destination hint table
- Information dictionary hint table
- Page label hint table

Generic hint tables may be used for product-specific objects accessed by conforming readers.

NOTE It is considerably more convenient for a conforming reader to use the generic hint representation than to specify custom hints.

Table F.9 – Generic hint table

item	Size (bits)	Description
1	32	The object number of the first object in the group.
2	32	The location of the first object in the group.
3	32	The number of objects in the group.
4	32	The length of the object group in bytes.

F.4.5 Extended Generic Hint Tables

An extended generic hint table shall begin with the same entries as in a generic hint table, and shall be followed by three additional entries, as shown in Table F.10. This table provides hints for accessing objects that reference shared objects. As of PDF 1.5, the following hint tables, if needed, shall use the extended generic format:

- Interactive form hint table
- Logical structure hint table
- Renditions name tree hint table

Embedded file streams shall not be referred to by this hint table, even if they are reachable from nodes in the renditions name tree; instead they shall use the hint table described in F.4.6, "Embedded File Stream Hint Tables."

Table F.10 – Extended generic hint table

Item	Size (bits)	Description
1	32	The object number of the first object in the group.
2	32	The location of the first object in the group.
3	32	The number of objects in the group.
4	32	The length of the object group in bytes.
5	32	The number of shared object references.
6	16	The number of bits needed to represent the numerically greatest shared object identifier used by the objects in the group.
7…	See Table F.3, item 11	Starting with item 7, each of the remaining items in this table shall be a shared object identifier—that is, an index into the shared object hint table (described in F.4.2, “Shared Object Hint Table”).

F.4.6 Embedded File Stream Hint Tables

The embedded file streams hint table allows a conforming reader to locate all byte ranges of a PDF file needed to access its embedded file streams. An embedded file stream may be grouped with other objects that it references; all objects in such a group shall have adjacent object numbers. (A group shall contain no objects at all if it contains shared object references.)

This hint table shall have a header section (see Table F.11), which shall have general information about the embedded file stream groups. The header section shall be followed by the entries in Table F.12. Each of the items in Table F.12 shall be repeated for each embedded file stream group (the number of groups being represented by item 3 in Table F.11). That is, the order of items in Table F.12 shall be item 1 for the first group, item 1 for the second group, and so on; item 2 for the first group, item 2 for the second group, and so on; repeated for the 5 items.

Table F.11 – Embedded file stream hint table, header section

Item	Size (bits)	Description
1	32	The object number of the first object in the first embedded file stream group.
2	32	The location of the first object in the first embedded file stream group.
3	32	The number of embedded file stream groups referenced by this hint table.
4	16	The number of bits needed to represent the highest object number corresponding to an embedded file stream object.
5	16	The number of bits needed to represent the greatest number of objects in an embedded file stream group.
6	16	The number of bits needed to represent the greatest length of an embedded file stream group, in bytes.
7	16	The number of bits needed to represent the greatest number of shared object references in any embedded file stream group.

Table F.12 – Embedded file stream hint table, per-embedded file stream group entries

Item	Size (bits)	Description
1	See Table F.11, item 4	The object number of the embedded file stream that this entry is associated with.
2	See Table F.11, item 5	The number of objects in this embedded file streams group. This item may be 0, meaning that there are only shared object references. In this case, item 4 for this group shall be greater than zero and item 3 shall be zero.
3	See Table F.11, item 6	The length of this embedded file stream group, in bytes. This item may be 0, which shall mean that there are only shared object references. In this case, item 4 for this group shall be greater than zero and item 2 shall be zero.
4	See Table F.11, item 7	The number of shared objects referenced by this embedded file stream group.
5	See Table F.3, item 11	A bit-packed list of shared object identifiers; that is, indices into the shared object hint table (see F.4.2, “Shared Object Hint Table”). Item 4 for this group shall specify how many shared object identifiers shall be associated with the group.

Annex G
(informative)

Linearized PDF Access Strategies

G.1 General

This section outlines how the conforming reader can take advantage of the structure of a Linearized PDF file to retrieve and display it efficiently. This material may help explain the rationale for the organization.

G.2 Opening at the First Page

As described earlier, when a document is initially accessed, a request is issued to retrieve the entire file, starting at the beginning. Consequently, Linearized PDF is organized so that all the data required to display the first page is at the beginning of the file. This includes all resources that are referenced from the first page, regardless of whether they are also referenced from other pages.

The first page is usually but not necessarily page 0. If the document catalogue contains an **OpenAction** entry that specifies opening at some page other than page 0, that page is the one physically located at the beginning of the document. Thus, opening a document at the default place (rather than a specific destination) requires simply waiting for the first-page data to arrive; no additional transactions are required.

In an ordinary conforming reader, opening a document requires first positioning to the end to obtain the **startxref** line. Since a Linearized PDF file has the first page’s cross-reference table at the beginning, reading the **startxref** line is not necessary. All that is required is to verify that the file length given in the linearization parameter dictionary at the beginning of the file matches the actual length of the file, indicating that no updates have been appended to the PDF file.

The primary hint stream is located either before or after the first-page section, which means that it is also retrieved as part of the initial sequential read of the file. The conforming reader is expected to interpret and retain all the information in the hint tables. The tables are reasonably compact and are not designed to be obtained from the file in random pieces.

The conforming reader must now decide whether to continue reading the remainder of the document sequentially or to abort the initial transaction and access subsequent pages by using separate transactions requesting byte ranges. This decision is a function of the size of the file, the data rate of the channel, and the overhead cost of a transaction.

G.3 Opening at an Arbitrary Page

The conforming reader may be requested to open a PDF file at an arbitrary page. The page can be specified in one of three ways:

- By page number (remote go-to action, integer page specifier)
- By named destination (remote go-to action, name or string page specifier)
- By article thread (thread action)

Additionally, an indexed search results in opening a document by page number. Handling this case efficiently is especially important.

As indicated above, when the document is initially opened, it is retrieved sequentially starting at the beginning. As soon as the hint tables have been received, the conforming reader has sufficient information to request retrieval of any page of the document given its page number. Therefore, the conforming reader can abort the initial transaction and issue a new transaction for the target page, as described in G.4, "Going to Another Page of an Open Document".

The position of the primary hint stream (part 5 in F.3.1, "General") with respect to the first-page section (part 6 in F.3.1, "General") determines how quickly this can be done. If the primary hint stream precedes the first-page section, the initial transaction can be aborted very quickly; however, this is at the cost of increased delay when opening the document at the first page. On the other hand, if the primary hint stream follows the first-page section, displaying the first page is quicker (since the hint tables are not needed for that), but opening at an arbitrary page is delayed by the time required to receive the first page. The decision whether to favour opening at the first page or opening at an arbitrary page must be made at the time a PDF file is linearized.

If an overflow hint stream exists, obtaining it requires issuing an additional transaction. For this reason, inclusion of an overflow hint stream in Linearized PDF, although permitted, is not recommended. The feature exists to allow the linearizer to write the PDF file with space reserved for a primary hint stream of an estimated size and then go back and fill in the hint tables. If the estimate is too small, the linearizer can append an overflow stream containing the remaining hint table data. Thus, the PDF file can be written in one pass, which may be an advantage if the performance of writing PDF is considered important.

Opening at a named destination requires the conforming reader first to read the entire **Dests** or **Names** dictionary, for which a hint is present. Using this information, it is possible to determine the page containing the specific destination identified by the name.

Opening to an article requires the conforming reader first to read the entire **Threads** array, which is located with the document catalogue at the beginning of the document. Using this information, it is possible to determine the page containing the first bead of any thread. Opening at other than the first bead of a thread requires chaining through all the beads until the desired one is reached; there are no hints to accelerate this.

G.4 Going to Another Page of an Open Document

Given a page number and the information in the hint tables, it is now straightforward for the conforming reader to construct a single request to retrieve any arbitrary page of the document. The request should include the following items:

- The objects of the page itself, whose byte range can be determined from the entry in the page offset hint table.
- The portion of the main cross-reference table referring to those objects. This can be computed from main cross-reference table location (the **T** entry in the linearization parameter dictionary) and the cumulative object number in the page offset hint table.
- The shared objects referenced from the page, whose byte ranges can be determined from information in the shared object hint table.
- The portion or portions of the main cross-reference table referring to those objects, as described above.

The purpose of the fractions in the page offset hint table is to enable the conforming reader to schedule retrieval of the page in a way that allows incremental display of the data as it arrives. It accomplishes this by constructing a request that interleaves pieces of the page contents with the shared resources that the contents refer to. This serves much the same purpose as the physical interleaving that is done for the first page.

G.5 Drawing a Page Incrementally

The ordering of objects in pages and the organization of the hint tables are intended to allow progressive update of the display and early opportunities for user interaction when the data is arriving slowly. The conforming reader must recognize instances in which the targets of indirect object references have not yet arrived and, where possible, rearrange the order in which it acts on the objects in the page.

The following sequence of actions is recommended:

a) Activate the annotations, but do not draw them yet. Also activate the cursor feedback for any article threads in the page.

b) Begin drawing the contents. Whenever there is a reference to an image XObject that has not yet arrived, skip over it. Whenever there is a reference to a font whose definition is an embedded font file that has not yet arrived, draw the text using a substitute font (if that is possible).

c) Draw the annotations.

d) Draw the images as they arrive, together with anything that overlaps them.

e) Once the embedded font definitions have arrived, redraw the text using the correct fonts, together with anything that overlaps the text.

The last two steps should be done using an off-screen buffer, if possible, to avoid objectionable flashing during the redraw process.

On encountering a reference XObject (see 8.10.4, "Reference XObjects"), the conforming reader may choose to initially display the object as a proxy and defer the retrieval and rendering of the imported content. Note that, since all XObjects in a Linearized PDF file follow the content stream of the page on which they appear, their retrieval is already deferred; the use of a reference XObject results in an additional level of deferral.

G.6 Following an Article Thread

As indicated earlier, the bead dictionaries for any article thread that visits a given page are located with that page. This enables the bead rectangles to be activated and proper cursor feedback to be shown.

If the user follows a thread, the conforming reader can obtain the object number from the **N** or **P** entry of the bead dictionary. This identifies a target bead, which is located with the page to which it belongs. Given this object number, the conforming reader can go to that page, as discussed in G.4, "Going to Another Page of an Open Document."

G.7 Accessing an Updated File

As stated earlier, if a Linearized PDF file subsequently has an incremental update appended to it, the linearization and hints are no longer valid. Actually, this is not necessarily true, but the conforming reader must do some additional work to validate the information.

When the conforming reader sees that the file is longer than the length given in the linearization parameter dictionary, it must issue an additional transaction to read everything that was appended. It must then analyse the objects in that update to see whether any of them modify objects that are in the first page or that are the targets of hints. If so, it must augment its internal data structures as necessary to take the updates into account.

For a PDF file that has received only a small update, this approach may be worthwhile. Accessing the file this way is quicker than accessing it without hints or retrieving the entire file before displaying any of it.

THIS PAGE BLANK

Annex H
(informative)

Example PDF Files

H.1 General

This annex presents several examples showing the structure of actual PDF files:

- A minimal file that can serve as a starting point for creating other PDF files (and that is the basis of later examples)
- A simple example that shows a text string—the classic “Hello World”—and a simple graphics example that draws lines and shapes
- A fragment of a PDF file that illustrates the structure of the page tree for a large document and, similarly, two fragments that illustrate the structure of an outline hierarchy
- An example showing the structure of a PDF file as it is updated several times, illustrating multiple body sections, cross-reference sections, and trailers

NOTE The ***Length*** values of stream objects in the examples and the byte addresses in cross-reference tables are not necessarily accurate.

H.2 Minimal PDF File

The example in H.2, "Minimal PDF File" is a PDF file that does not draw anything; it is almost the minimum acceptable PDF file. It is not strictly the minimum acceptable because it contains an outline dictionary (**Outlines** in the document catalog) with a zero count (in which case this object would normally be omitted); a page content stream (**Contents** in the page object); and a resource dictionary (**Resources** in the page object) containing a **ProcSet** array. These objects were included to make this file useful as a starting point for creating other, more realistic PDF files.

Table H.1 lists the objects in this example.

Table H.1 – Objects in minimal example

Object number	Object type
1	**Catalog** (document catalog)
2	**Outlines** (outline dictionary)
3	**Pages** (page tree node)
4	**Page** (page object)
5	Content stream
6	Procedure set array

NOTE When using the example in H.2, "Minimal PDF File" as a starting point for creating other files, remember to update the ***ProcSet*** array as needed (see 14.2, "Procedure Sets"). Also, remember that the cross-reference table entries may need to have a trailing SPACE (see 7.5.4, "Cross-Reference Table").

EXAMPLE
```
%PDF-1.4
1  0  obj
    <<  /Type  /Catalog
         /Outlines  2 0 R
         /Pages  3 0 R
    >>
endobj

2  0  obj
    <<  /Type  Outlines
         /Count  0
    >>
endobj

3  0  obj
    <<  /Type  /Pages
         /Kids  [4 0 R]
         /Count  1
    >>
endobj

4  0  obj
    <<  /Type  /Page
         /Parent  3 0 R
         /MediaBox  [0  0  612  792]
         /Contents  5 0 R
         /Resources  <<  /ProcSet  6 0 R  >>
    >>
endobj

5  0  obj
    <<  /Length  35  >>
stream
…Page-marking operators…
endstream
endobj

6  0  obj
    [/PDF]
endobj

xref
0  7
0000000000  65535  f
0000000009  00000  n
0000000074  00000  n
0000000120  00000  n
0000000179  00000  n
0000000300  00000  n
0000000384  00000  n

trailer
    <<  /Size  7
         /Root  1 0 R
    >>
startxref
408
%%EOF
```

H.3 Simple Text String Example

The example in H.3, "Simple Text String Example" is the classic “Hello World” example built from the preceding example. It shows a single line of text consisting of the string *Hello World*, illustrating the use of fonts and several text-related PDF operators. The string is displayed in 24-point Helvetica. Because Helvetica is one of the standard 14 fonts, no font descriptor is needed.

Table H.2 lists the objects in this example.

Table H.2 – Objects in simple text string example

Object number	Object type
1	**Catalog** (document catalog)
2	**Outlines** (outline dictionary)
3	**Pages** (page tree node)
4	**Page** (page object)
5	Content stream
6	Procedure set array
7	**Font** (Type 1 font)

EXAMPLE

```
%PDF-1.4
1 0 obj
   << /Type /Catalog
       /Outlines 2 0 R
       /Pages 3 0 R
   >>
endobj

2 0 obj
   << /Type /Outlines
       /Count 0
   >>
endobj

3 0 obj
   << /Type /Pages
       /Kids [4 0 R]
       /Count 1
   >>
endobj

4 0 obj
   << /Type /Page
       /Parent 3 0 R
       /MediaBox [0 0 612 792]
       /Contents 5 0 R
       /Resources << /ProcSet 6 0 R
                     /Font << /F1 7 0 R >>
                  >>
   >>
endobj

5 0 obj
   << /Length 73 >>
stream
   BT
      /F1 24 Tf
      100 100 Td
      (Hello World) Tj
   ET
endstream
endobj
```

```
6  0  obj
    [/PDF  /Text]
endobj

7  0  obj
    <<  /Type  /Font
          /Subtype  /Type1
          /Name  /F1
          /BaseFont  /Helvetica
          /Encoding  /MacRomanEncoding
    >>
endobj

xref
0  8
0000000000  65535  f
0000000009  00000  n
0000000074  00000  n
0000000120  00000  n
0000000179  00000  n
0000000364  00000  n
0000000466  00000  n
0000000496  00000  n

trailer
    <<  /Size  8
          /Root  1 0 R
    >>
startxref
625
%%EOF
```

H.4 Simple Graphics Example

The example in H.4, "Simple Graphics Example" draws a thin black line segment, a thick black dashed line segment, a filled and stroked rectangle, and a filled and stroked cubic Bézier curve. Table H.3 lists the objects in this example, and Figure H.1 shows the resulting output. (Each shape has a red border, and the rectangle is filled with light blue.)

Table H.3 – Objects in simple graphics example

Object number	Object type
1	**Catalog** (document catalog)
2	**Outlines** (outline dictionary)
3	**Pages** (page tree node)
4	**Page** (page object)
5	Content stream
6	Procedure set array

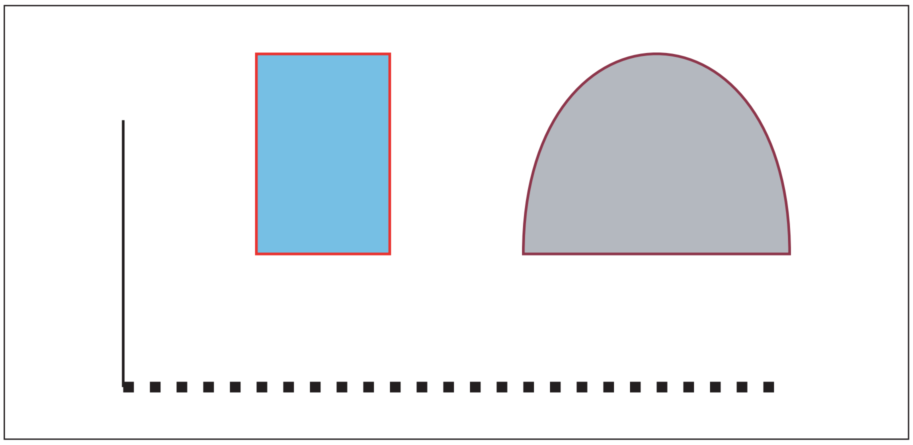


Figure H.1 – Output of the following example

EXAMPLE

```
%PDF-1.4
1 0 obj
    << /Type /Catalog
         /Outlines 2 0 R
         /Pages 3 0 R
    >>
endobj

2 0 obj
    << /Type /Outlines
         /Count 0
    >>
endobj

3 0 obj
    << /Type /Pages
         /Kids [4 0 R]
         /Count 1
    >>
endobj

4 0 obj
    << /Type /Page
         /Parent 3 0 R
         /MediaBox [0 0 612 792]
         /Contents 5 0 R
         /Resources << /ProcSet 6 0 R >>
    >>
endobj

5 0 obj
    << /Length 883 >>
stream
    % Draw a black line segment, using the default line width.
    150 250 m
    150 350 l
    S

    % Draw a thicker, dashed line segment.
    4 w                                                  % Set line width to 4 points
    [4 6] 0 d                                            % Set dash pattern to 4 units on, 6 units off
    150 250 m
    400 250 l
    S
    [] 0 d                                               % Reset dash pattern to a solid line
    1 w                                                  % Reset line width to 1 unit
```

```
        % Draw a rectangle with a 1-unit red border, filled with light blue.
        1.0  0.0  0.0  RG                                   % Red for stroke color
        0.5  0.75  1.0  rg                                  % Light blue for fill color
        200  300  50  75  re
        B

        % Draw a curve filled with gray and with a colored border.
        0.5  0.1  0.2  RG
        0.7  g
        300  300  m
        300  400  400  400  400  300  c
        b
    endstream
    endobj

    6  0  obj
        [/PDF]
    endobj

    xref
    0  7
    0000000000  65535  f
    0000000009  00000  n
    0000000074  00000  n
    0000000120  00000  n
    0000000179  00000  n
    0000000300  00000  n
    0000001532  00000  n

    trailer
        <<  /Size  7
            /Root  1 0 R
        >>
    startxref
    1556
    %%EOF
```

H.5 Page Tree Example

The example in H.5, "Page Tree Example" is a fragment of a PDF file illustrating the structure of the page tree for a large document. It contains the page tree nodes for a 62-page document. Figure H.2 shows the structure of this page tree. Numbers in the figure are object numbers corresponding to the objects in the example.

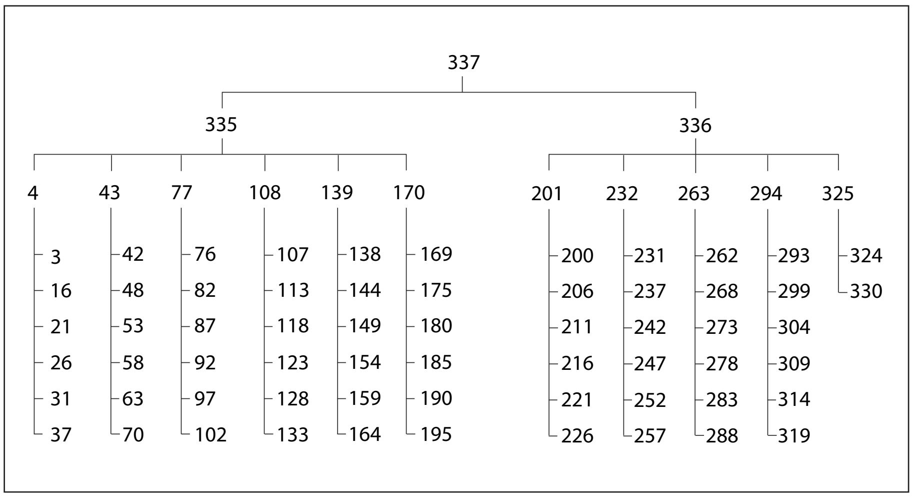


Figure H.2 – Page tree for the following example

EXAMPLE

```
337 0 obj
    << /Type /Pages
         /Kids [   335 0 R
                   336 0 R
               ]
         /Count 62
    >>
endobj

335 0 obj
    << /Type /Pages
         /Parent 337 0 R
         /Kids [   4 0 R
                   43 0 R
                   77 0 R
                   108 0 R
                   139 0 R
                   170 0 R
               ]
         /Count 36
    >>
endobj

336 0 obj
    << /Type /Pages
         /Parent 337 0 R
         /Kids [   201 0 R
                   232 0 R
                   263 0 R
                   294 0 R
                   325 0 R
               ]
         /Count 26
    >>
endobj

4 0 obj
    << /Type /Pages
         /Parent 335 0 R
         /Kids [   3 0 R
                   16 0 R
                   21 0 R
                   26 0 R
```

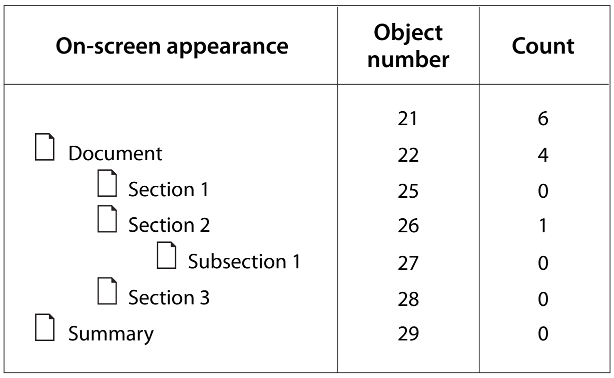
```
                    31 0 R
                    37 0 R
                  ]
            /Count  6
        >>
    endobj

    43  0  obj
        <<  /Type  /Pages
            /Parent  335 0 R
            /Kids  [    42 0 R
                        48 0 R
                        53 0 R
                        58 0 R
                        63 0 R
                        70 0 R
                  ]
            /Count  6
        >>
    endobj

    77  0  obj
        <<  /Type  /Pages
            /Parent  335 0 R
            /Kids  [    76 0 R
                        82 0 R
                        87 0 R
                        92 0 R
                        97 0 R
                        102 0 R
                  ]
            /Count  6
        >>
    endobj

    108  0  obj
        <<  /Type  /Pages
            /Parent  335 0 R
            /Kids  [    107 0 R
                        113 0 R
                        118 0 R
                        123 0 R
                        128 0 R
                        133 0 R
                  ]
            /Count  6
        >>
    endobj

    139  0  obj
        <<  /Type  /Pages
            /Parent  335 0 R
            /Kids  [    138 0 R
                        144 0 R
                        149 0 R
                        154 0 R
                        159 0 R
                        164 0 R
                  ]
            /Count  6
        >>
    endobj

    170  0  obj
        <<  /Type  /Pages
```

```
            /Parent  335 0 R
            /Kids  [    169 0 R
                        175 0 R
                        180 0 R
                        185 0 R
                        190 0 R
                        195 0 R
                     ]
            /Count  6
      >>
endobj

201  0  obj
      <<  /Type  /Pages
            /Parent  336 0 R
            /Kids  [    200 0 R
                        206 0 R
                        211 0 R
                        216 0 R
                        221 0 R
                        226 0 R
                     ]
            /Count  6
      >>
endobj

232  0  obj
      <<  /Type  /Pages
            /Parent  336 0 R
            /Kids  [    231 0 R
                        237 0 R
                        242 0 R
                        247 0 R
                        252 0 R
                        257 0 R
                     ]
            /Count  6
      >>
endobj

263  0  obj
      <<  /Type  /Pages
            /Parent  336 0 R
            /Kids  [    262 0 R
                        268 0 R
                        273 0 R
                        278 0 R
                        283 0 R
                        288 0 R
                     ]
            /Count  6
      >>
endobj

294  0  obj
      <<  /Type  /Pages
            /Parent  336 0 R
            /Kids  [    293 0 R
                        299 0 R
                        304 0 R
                        309 0 R
                        314 0 R
                        319 0 R
                     ]
            /Count  6
```

```
    >>
endobj

325  0  obj
    <<  /Type  /Pages
         /Parent  336 0 R
         /Kids  [    324 0 R
                     330 0 R
                 ]
         /Count  2
    >>
endobj
```

H.6 Outline Hierarchy Example

This section from a PDF file illustrates the structure of an outline hierarchy with six items. Example 1 in H.6, "Outline Hierarchy Example" shows the outline with all items open, as illustrated in Figure H.3.



On-screen appearance	Object number	Count
	21	6
🗋 Document	22	4
🗋 Section 1	25	0
🗋 Section 2	26	1
🗋 Subsection 1	27	0
🗋 Section 3	28	0
🗋 Summary	29	0

Figure H.3 – Document outline as displayed in Example 1

EXAMPLE 1

```
21  0  obj
    <<  /Type  /Outlines
         /First  22 0 R
         /Last  29 0 R
         /Count  6
    >>
endobj

22  0  obj
    <<  /Title  (Document)
         /Parent  21 0 R
         /Next  29 0 R
         /First  25 0 R
         /Last  28 0 R
         /Count  4
         /Dest  [3 0 R  /XYZ  0  792  0]
    >>
endobj

25  0  obj
    <<  /Title  (Section 1)
         /Parent  22 0 R
         /Next  26 0 R
         /Dest  [3 0 R  /XYZ  null  701  null]
    >>
endobj

26  0  obj
```

```
    <<  /Title  (Section 2)
          /Parent  22 0 R
          /Prev  25 0 R
          /Next  28 0 R
          /First  27 0 R
          /Last  27 0 R
          /Count  1
          /Dest  [3 0 R  /XYZ  null  680  null]
    >>
endobj

27  0  obj
    <<  /Title  (Subsection 1)
          /Parent  26 0 R
          /Dest  [3 0 R  /XYZ  null  670  null]
    >>
endobj

28  0  obj
    <<  /Title  (Section 3)
          /Parent  22 0 R
          /Prev  26 0 R
          /Dest  [7 0 R  /XYZ  null  500  null]
    >>
endobj

29  0  obj
    <<  /Title  (Summary)
          /Parent  21 0 R
          /Prev  22 0 R
          /Dest  [8 0 R  /XYZ  null  199  null]
    >>
endobj
```

Example 2 in H.6, "Outline Hierarchy Example" is the same as Example 1, except that one of the outline items has been closed in the display. The outline appears as shown in Figure H.4.

On-screen appearance	Object number	Count
	21	5
🗋 Document	22	3
🗋 Section 1	25	0
🗋 Section 2	26	−1
🗋 Section 3	28	0
🗋 Summary	29	0

Figure H.4 – Document outline as displayed in Example 2

EXAMPLE 2

```
21  0  obj
    <<  /Type  /Outlines
          /First  22 0 R
          /Last  29 0 R
          /Count  5
    >>
endobj

22  0  obj
    <<  /Title  (Document)
          /Parent  21 0 R
```

```
            /Next  29 0 R
            /First  25 0 R
            /Last  28 0 R
            /Count  3
            /Dest  [3 0 R  /XYZ  0  792  0]
    >>
endobj

25  0  obj
    <<  /Title  (Section 1)
            /Parent  22 0 R
            /Next  26 0 R
            /Dest  [3 0 R  /XYZ  null  701  null]
    >>
endobj

26  0  obj
    <<  /Title  (Section 2)
            /Parent  22 0 R
            /Prev  25 0 R
            /Next  28 0 R
            /First  27 0 R
            /Last  27 0 R
            /Count  −1
            /Dest  [3 0 R  /XYZ  null  680  null]
    >>
endobj

27  0  obj
    <<  /Title  (Subsection 1)
            /Parent  26 0 R
            /Dest  [3 0 R  /XYZ  null  670  null]
    >>
endobj

28  0  obj
    <<  /Title  (Section 3)
            /Parent  22 0 R
            /Prev  26 0 R
            /Dest  [7 0 R  /XYZ  null  500  null]
    >>
endobj

29  0  obj
    <<  /Title  (Summary)
            /Parent  21 0 R
            /Prev  22 0 R
            /Dest  [8 0 R  /XYZ  null  199  null]
    >>
endobj
```

H.7 Updating Example

This example shows the structure of a PDF file as it is updated several times; it illustrates multiple body sections, cross-reference sections, and trailers. In addition, it shows that once an object has been assigned an object identifier, it keeps that identifier until the object is deleted, even if the object is altered. Finally, the example illustrates the reuse of cross-reference entries for objects that have been deleted, along with the incrementing of the generation number after an object has been deleted.

The original file is the example in H.2, "Minimal PDF File". The updates are divided into four stages, with the file saved after each stage:

a) Four text annotations are added.

b) The text of one of the annotations is altered.

c) Two of the text annotations are deleted.

d) Three text annotations are added.

The following sections show the segments added to the file at each stage. Throughout this example, objects are referred to by their object identifiers, which are made up of the object number and the generation number, rather than simply by their object numbers as in earlier examples. This is necessary because the example reuses object numbers; therefore, the objects they denote are not unique.

NOTE The tables in these sections show only those objects that are modified during the updating process. Objects from H.2, "Minimal PDF File" that are not altered during the update are not shown.

H.7.1 Stage 1: Add Four Text Annotations

Four text annotations are added to the initial file and the file is saved. Table H.4 lists the objects involved in this update.

Table H.4 – Object usage after adding four text annotations

Object identifier	Object type
4 0	**Page** (page object)
7 0	Annotation array
8 0	**Annot** (annotation dictionary)
9 0	**Annot** (annotation dictionary)
10 0	**Annot** (annotation dictionary)
11 0	**Annot** (annotation dictionary)

The example in H.7.1, "Stage 1: Add Four Text Annotations" shows the lines added to the file by this update. The page object is updated because an **Annots** entry has been added to it. Note that the file's trailer now contains a **Prev** entry, which points to the original cross-reference section in the file, while the **startxref** value at the end of the trailer points to the cross-reference section added by the update.

EXAMPLE
```
4 0 obj
   << /Type /Page
        /Parent 3 0 R
        /MediaBox [0 0 612 792]
        /Contents 5 0 R
        /Resources << /ProcSet 6 0 R >>
        /Annots 7 0 R
   >>
endobj

7 0 obj
   [ 8 0 R
     9 0 R
     10 0 R
     11 0 R
   ]
endobj

8 0 obj
   << /Type /Annot
        /Subtype /Text
```

```
            /Rect  [44  616  162  735]
            /Contents  (Text #1)
            /Open  true
     >>
endobj

9  0  obj
     <<  /Type  /Annot
            /Subtype  /Text
            /Rect  [224  668  457  735]
            /Contents  (Text #2)
            /Open  false
     >>
endobj

10  0  obj
     <<  /Type  /Annot
            /Subtype  /Text
            /Rect  [239  393  328  622]
            /Contents  (Text #3)
            /Open  true
     >>
endobj

11  0  obj
     <<  /Type  /Annot
            /Subtype  /Text
            /Rect  [34  398  225  575]
            /Contents  (Text #4)
            /Open  false
     >>
endobj

xref
0  1
0000000000  65535  f
4  1
0000000632  00000  n
7  5
0000000810  00000  n
0000000883  00000  n
0000001024  00000  n
0000001167  00000  n
0000001309  00000  n

trailer
     <<  /Size  12
            /Root  1 0 R
            /Prev  408
     >>
startxref
1452
%%EOF
```

H.7.2 Stage 2: Modify Text of One Annotation

One text annotation is modified and the file is saved. The example in H.7.2, "Stage 2: Modify Text of One Annotation" shows the lines added to the file by this update. Note that the file now contains two copies of the object with identifier 10 0 (the text annotation that was modified) and that the added cross-reference section points to the more recent version of the object. This added cross-reference section contains one subsection, which contains only an entry for the object that was modified. In addition, the **Prev** entry in the file's trailer has been updated to point to the cross-reference section added in the previous stage, while the **startxref** value at the end of the trailer points to the newly added cross-reference section.

EXAMPLE
```
10 0 obj
   << /Type /Annot
      /Subtype /Text
      /Rect [239 393 328 622]
      /Contents (Modified Text #3)
      /Open true
   >>
endobj

xref
0 1
0000000000 65535 f
10 1
0000001703 00000 n

trailer
   << /Size 12
      /Root 1 0 R
      /Prev 1452
   >>
startxref
1855
%%EOF
```

H.7.3 Stage 3: Delete Two Annotations

Two text annotation are deleted and the file is saved. Table H.5 lists the objects updated.

Table H.5 – Object usage after deleting two text annotations

Object identifier	Object type
7 0	Annotation array
8 0	Free
9 0	Free

The **Annots** array is the only object that is written in this update. It is updated because it now contains two annotations fewer.

The example in H.7.3, "Stage 3: Delete Two Annotations" shows the lines added when the file was saved. Note that objects with identifiers 8 0 and 9 0 have been deleted, as can be seen from the fact that their entries in the cross-reference section end with the keyword **f**.

EXAMPLE
```
7 0 obj
   [ 10 0 R
     11 0 R
   ]
endobj

xref
0 1
0000000008 65535 f
7 3
0000001978 00000 n
0000000009 00001 f
0000000000 00001 f

trailer
   << /Size 12
      /Root 1 0 R
```

```
            /Prev  1855
        >>
    startxref
    2027
    %%EOF
```

The cross-reference section added at this stage contains four entries, representing object number 0, the **Annots** array, and the two deleted text annotations.

- The cross-reference entry for object number 0 is updated because it is the head of the linked list of free entries and points to the entry for the newly freed object number 8. The entry for object number 8 points to the entry for object number 9 (the next free entry), while the entry for object number 9 is the last free entry in the cross-reference table, indicated by the fact that it points back to object number 0.
- The entries for the two deleted text annotations are marked as free and as having generation numbers of 1, which are used for any objects that reuse these cross-reference entries. Keep in mind that, although the two objects have been deleted, they are still present in the file. It is the cross-reference table that records the fact that they have been deleted.

The **Prev** entry in the trailer has again been updated so that it points to the cross-reference section added at the previous stage, and the **startxref** value points to the newly added cross-reference section.

H.7.4 Stage 4: Add Three Annotations

Finally, three new text annotations are added to the file. Table H.6 lists the objects involved in this update.

Table H.6 – Object usage after adding three text annotations

Object identifier	Object type
7 0	Annotation array
8 1	**Annot** (annotation dictionary)
9 1	**Annot** (annotation dictionary)
12 0	**Annot** (annotation dictionary)

Object numbers 8 and 9, which were used for the two annotations deleted in the previous stage, have been reused; however, the new objects have been given a generation number of 1. In addition, the third text annotation added has been assigned the previously unused object identifier of 12 0.

The example in H.7.4, "Stage 4: Add Three Annotations" shows the lines added to the file by this update. The added cross-reference section contains five entries, corresponding to object number 0, the **Annots** array, and the three annotations added. The entry for object number 0 is updated because the previously free entries for object numbers 8 and 9 have been reused. The entry for object number 0 now shows that the cross-reference table has no free entries. The **Annots** array is updated to reflect the addition of the three text annotations.

EXAMPLE

```
7  0  obj
    [  10 0 R
       11 0 R
       8 1 R
       9 1 R
       12 0 R
    ]
endobj

8  1  obj
    <<  /Type  /Annot
        /Subtype  /Text
        /Rect [58 657 172 742]
        /Contents (New Text #1)
        /Open true
    >>
endobj

9 1 obj
    << /Type /Annot
        /Subtype /Text
        /Rect [389 459 570 537]
        /Contents (New Text #2)
        /Open false
    >>
endobj

12 0 obj
    << /Type /Annot
        /Subtype /Text
        /Rect [44 253 473 337]
        /Contents (New Text #3\203a longer text annotation which we will continue \
          onto a second line)
        /Open true
    >>
endobj

xref
0 1
0000000000 65535 f
7 3
0000002216 00000 n
0000002302 00001 n
0000002447 00001 n
12 1
0000002594 00000 n

trailer
    << /Size 13
        /Root 1 0 R
        /Prev 2027
    >>
startxref
2814
%%EOF
```

The annotation with object identifier 12 0 illustrates splitting a long text string across multiple lines, as well as the technique for including nonstandard characters in a string. In this case, the character is an ellipsis (…), which is character code 203 (octal) in **PDFDocEncoding**, the encoding used for text annotations.

As in previous updates, the trailer's **Prev** entry and **startxref** value have been updated.

H.8 Structured Elements That Describe Hierarchical Lists

H.8, "Structured Elements That Describe Hierarchical Lists" presents examples that illustrate how structured elements are used to describe hierarchical lists, such as a table of contents or an index.

H.8.1 Table of Contents

The structured element’s structure type entry (**S**) may have values that establish hierarchical relationships between entries in a table of content. The TOCI value specifies an individual member of a table of contents. The TOC value specifies a list made up of other table of contents items that are individual members of the table of contents and/or lists of table of contents items. (The trailing character in ***TOCI*** is an upper case “I”.)

Figure H.5 shows the table of contents described by the example in H.8.1, "Table of Contents".

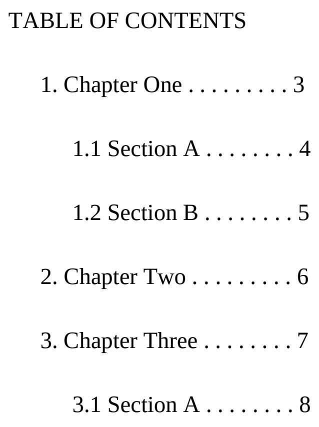


TABLE OF CONTENTS

1. Chapter One 3

    1.1 Section A 4

    1.2 Section B 5

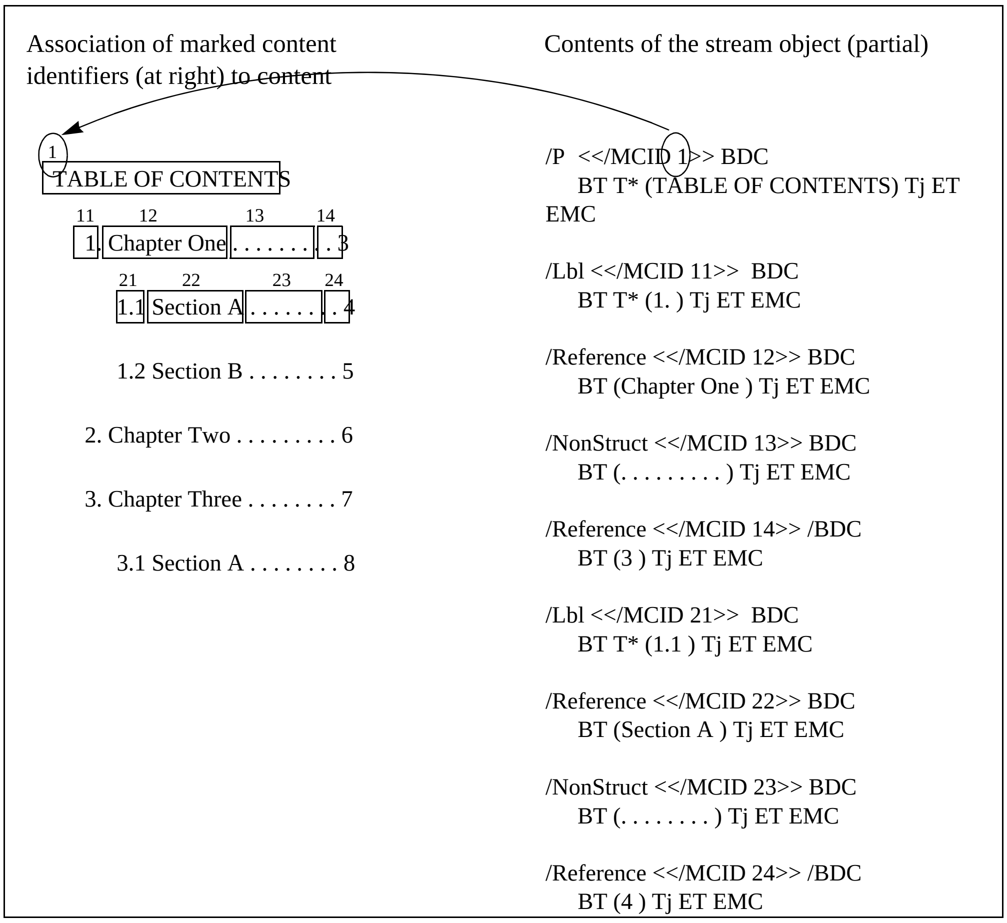
2. Chapter Two 6

3. Chapter Three 7

    3.1 Section A 8

Figure H.5 – Table of contents

Figure H.6 illustrates the association between marked content identifiers (**MCID**) and content. This illustration includes part of the stream object so you can see how the MCID entries are associated with the content in the table of contents.


Association of marked content identifiers (at right) to content
1
TABLE OF CONTENTS
11 12 13 14
1. Chapter One 3
21 22 23 24
1.1 Section A 4
1.2 Section B 5
2. Chapter Two 6
3. Chapter Three 7
3.1 Section A 8
Contents of the stream object (partial)

```
/P  <</MCID 1>> BDC
    BT T* (TABLE OF CONTENTS) Tj ET
EMC

/Lbl <</MCID 11>>  BDC
    BT T* (1. ) Tj ET EMC

/Reference <</MCID 12>> BDC
    BT (Chapter One ) Tj ET EMC

/NonStruct <</MCID 13>> BDC
    BT (. . . . . . . . . ) Tj ET EMC

/Reference <</MCID 14>> /BDC
    BT (3 ) Tj ET EMC

/Lbl <</MCID 21>>  BDC
    BT T* (1.1 ) Tj ET EMC

/Reference <</MCID 22>> BDC
    BT (Section A ) Tj ET EMC

/NonStruct <</MCID 23>> BDC
    BT (. . . . . . . . ) Tj ET EMC

/Reference <</MCID 24>> /BDC
    BT (4 ) Tj ET EMC
```

Figure H.6 – Association between content and marked content identifiers

Figure H.7 shows how the relationships of the structure elements and their use of the TOC and TOCI structure types represent the structure of a table of contents. This figure also shows the relationship between the structured content elements and the marked content in the stream. Gray text indicates marked content identifiers (MCID).

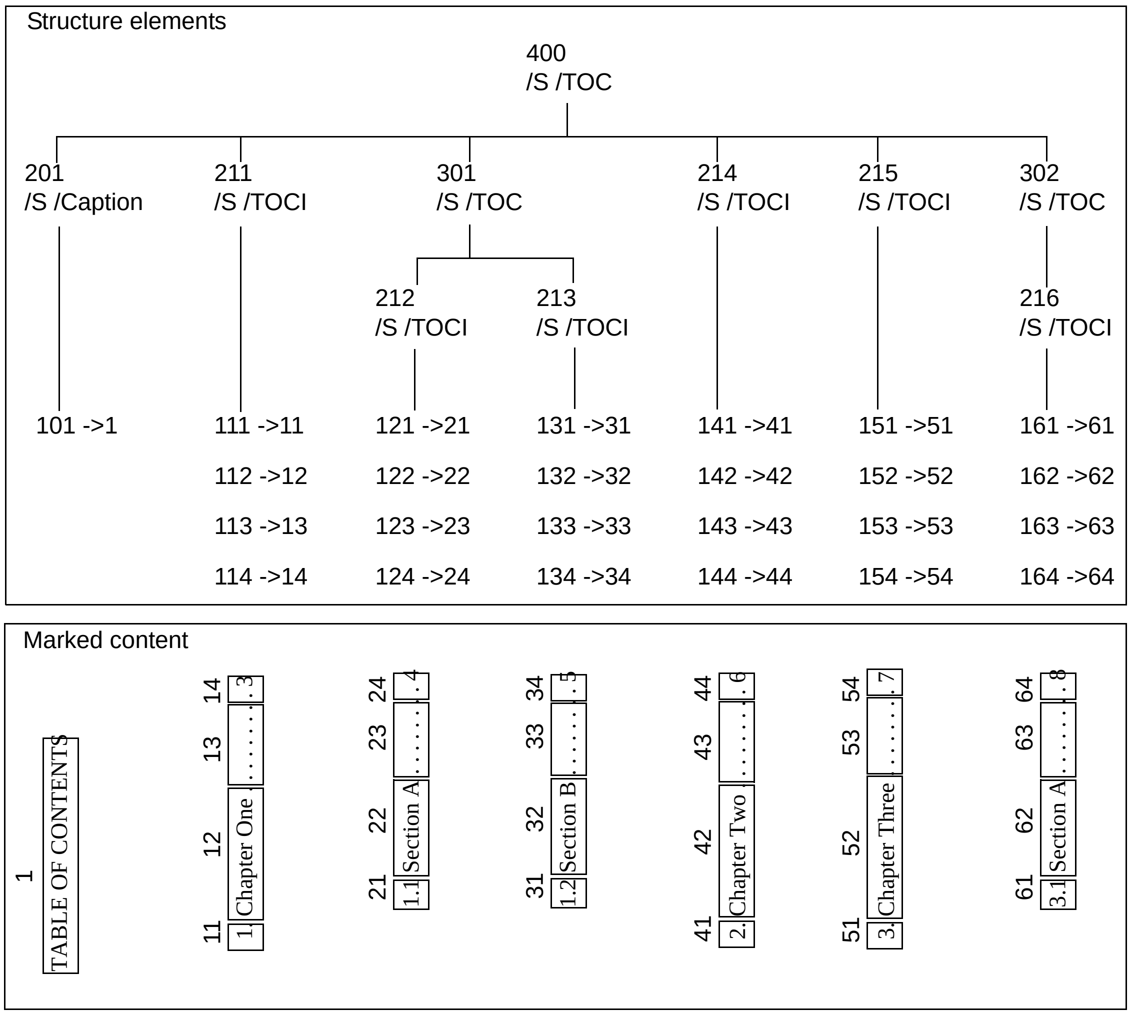


Figure H.7 – Hierarchy of structure elements and relationship with marked content

EXAMPLE

```
4 0 obj
   <</Type /Page
      /Contents 5 0 R
   >>

5 0 obj
   <</Length  6 0 R >>
   stream
      /P  <</MCID 1>> BDC
         BT T* (TABLE OF CONTENTS) Tj ET EMC

      /Lbl <</MCID 11>>  BDC
         BT T* (1. ) Tj ET EMC
      /Reference <</MCID 12>> BDC
         BT (Chapter One ) Tj ET EMC
      /NonStruct <</MCID 13>> BDC
         BT (. . . . . . . . . ) Tj ET EMC
      /Reference <</MCID 14>> /BDC
         BT (3 ) Tj ET EMC

      /Lbl <</MCID 21>>  BDC
```

```
            BT T* (1.1 ) Tj ET EMC
        /Reference <</MCID 22>> BDC
            BT (Section A ) Tj ET EMC
        /NonStruct <</MCID 23>> BDC
            BT (. . . . . . . . ) Tj ET EMC
        /Reference <</MCID 24>> /BDC
            BT (4 ) Tj ET EMC

        /Lbl <</MCID 31>>  BDC
            BT T* (1.2 ) Tj ET EMC
        /Reference <</MCID 32>> BDC
             BT (Section B ) Tj ET EMC
        /NonStruct <</MCID 33>> BDC
            BT (. . . . . . . . ) Tj ET EMC
        /Reference <</MCID 34>> /BDC
            BT (5 ) Tj ET EMC

        /Lbl <</MCID 41>>  BDC
            BT T* (2. ) Tj ET EMC
        /Reference <</MCID 42>> BDC
             BT (Chapter Two ) Tj ET EMC
        /NonStruct <</MCID 43>> BDC
            BT (. . . . . . . . . ) Tj ET EMC
        /Reference <</MCID 44>> /BDC
             BT (6 ) Tj ET EMC

        /Lbl <</MCID 51>>  BDC
            BT T* (3. ) Tj ET EMC
        /Reference <</MCID 52>> BDC
            BT (Chapter Three ) Tj ET EMC
        /NonStruct <</MCID 53>> BDC
            BT (. . . . . . . . ) Tj ET EMC
        /Reference <</MCID 54>> /BDC
            BT (7 ) Tj ET EMC

        /Lbl <</MCID 61>>  BDC
            BT T* (3.1 ) Tj ET EM
        /Reference <</MCID 62>> BDC
            BT (Section A ) Tj ET EM
        /NonStruct <</MCID 63>> BDC
            BT (. . . . . . . . ) Tj ET EM
        /Reference <</MCID 64>> /BDC
            BT (8 ) Tj ET EMC
    endstream
endobj

101 0 obj
    << /Type /StructElem
        /S /P
        /P 201 0 R
        /Pg 4 0 R
        /K 1
    >>
endobj
111 0 obj
    << /Type /StructElem
        /S /Lbl
        /P 211 0 R
        /Pg 4 0 R
         /K 11
    >>
endobj

112 0 obj
    << /Type /StructElem
        /S /Reference
        /P 211 0 R
        /Pg 4 0 R
        /K 12
    >>
endobj

113 0 obj
    << /Type /StructElem
        /S /NonStruct
        /P 211 0 R
        /Pg 4 0 R
         /K 13
    >>
endobj

114 0 obj
    << /Type /StructElem
        /S /Reference
        /P 211 0 R
        /Pg 4 0 R
        /K 14
     >>
endobj
```

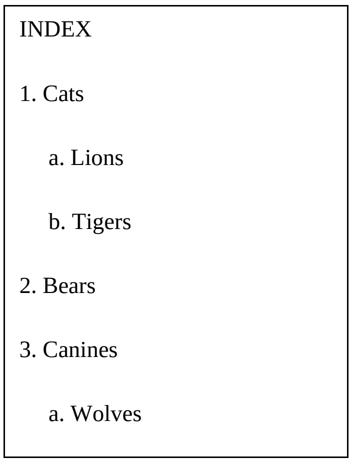
objects 121-124, 131-134, 141-144, 151-154 and 161-164 referencing MCIDs 21-24, 31-34, 41-44, 51-54, and 61-64 are omitted in the interest of space.

```
201 0 obj
    << /Type /StructElem
        /S /Caption
        /P 400 0 R
        /K [101 0 R]
    >>
endobj

211 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 400 0 R
        /K [111 0 R 112 0 R 113 0 R 114 0 R]
    >>
endobj

212 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 301 0 R
        /K [121 0 R 122 0 R 123 0 R 124 0 R]
    >>
endobj

213 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 301 0 R
        /K [131 0 R 132 0 R 133 0 R 134 0 R]
    >>
endobj

214 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 400 0 R
        /K [141 0 R 142 0 R 143 0 R 144 0 R]
    >>
endobj

215 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 400 0 R
        /K [151 0 R 152 0 R 153 0 R 154 0 R]
    >>
endobj

216 0 obj
    << /Type /StructElem
        /S /TOCI
        /P 302 0 R
        /K [161 0 R 162 0 R 163 0 R 164 0 R]
    >>
endobj

301 0 obj
    << /Type /StructElem
        /S /TOC
        /P 400 0 R
        /K [212 0 R 213 0 R]
    >>
endobj

302 0 obj
    << /Type /StructElem
        /S /TOC
        /P 400 0 R
        /K [216 0 R]
    >>
endobj

400 0 obj
    << /Type /StructElem
        /S TOC
        /K [201 0 R 211 0 R 301 0 R 214 0 R 215 0 R 302 0 R]
    >>
endobj
```

H.8.2 Nested Lists

The structured element’s structure type entry (**S**) may have values that establish hierarchical relationships between entries in an index. The LI value specifies an individual index entry. The L value specifies a list made up of individual index entries and/or lists of index entries. (The trailing character in ***LI*** is an upper case “I”.)

Figure H.8 shows the index described by the example in H.8.2, "Nested Lists".



INDEX

1. Cats

   a. Lions

   b. Tigers

2. Bears

3. Canines

   a. Wolves

Figure H.8 – Index

Figure H.9 shows how the relationships of the structure elements and their use of the L and LI structure types defines the structure of an index. This figure also shows the relationship between the structured content elements and the marked content in the stream. Gray text indicates marked content identifiers (**MCID**).

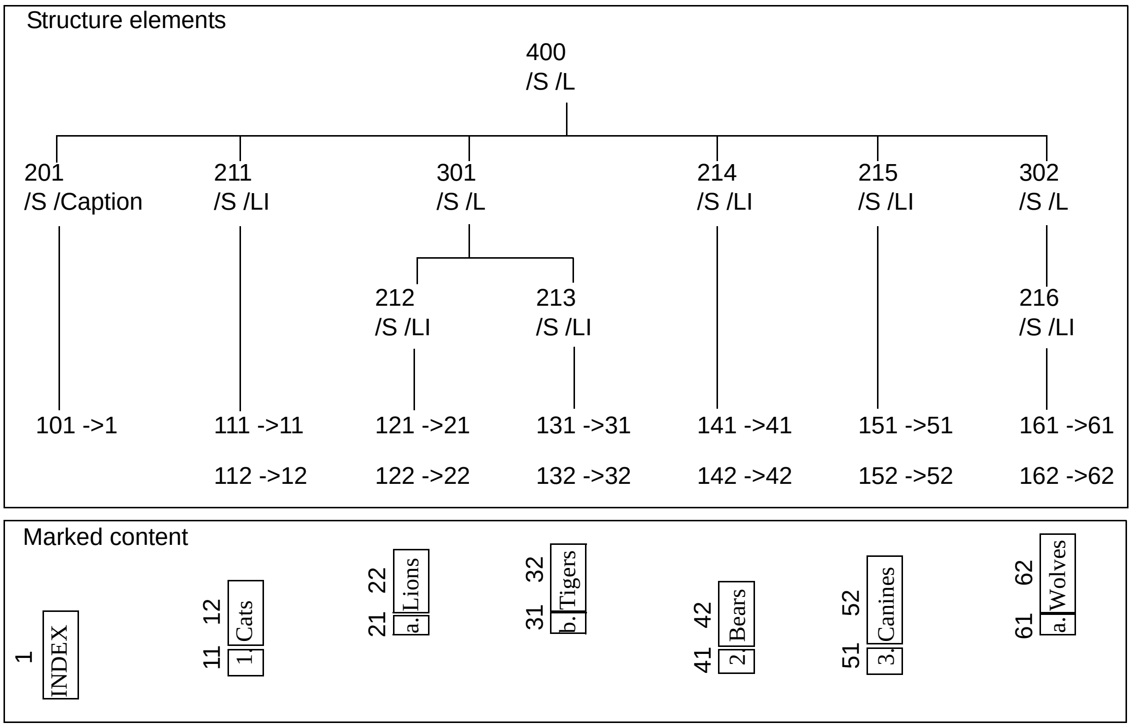


Figure H.9 – Hierarchy of structure elements and relationship with marked content

EXAMPLE

```
4 0 obj
    << /Type /Page
        /Contents 5 0 R
    >>
endobj

5 0 obj
    << /Length 6 0 R >>
    stream
        /P <</MCID 1>> BDC
            BT T* (INDEX) Tj ET EMC

        /Lbl <</MCID 11>> BDC
            BT T* (1. ) Tj ET EMC
        /LBody <</MCID 12>> /BDC
            BT (Cats ) Tj ET EMC

        /Lbl <</MCID 21>> BDC
            BT T* (a. ) Tj ET EMC
        /LBody <</MCID 22>> /BDC
            BT (Lions ) Tj ET EMC

        /Lbl <</MCID 31>> BDC
            BT T* (b. ) Tj ET EMC
        /LBody <</MCID 32>> /BDC
            BT (Tigers ) Tj ET EMC

        /Lbl <</MCID 41>> BDC
            BT T* (2. ) Tj ET EMC
        /LBody <</MCID 42>> /BDC
            BT (Bears ) Tj ET EMC

        /Lbl <</MCID 51>> BDC
            BT T* (3. ) Tj ET EM
        /LBody <</MCID 52>> /BDC
            BT (Canines ) Tj ET EMC

        /Lbl <</MCID 61>> BDC
            BT T* (a. ) Tj ET EM
        /LBody <</MCID 62>> /BDC
            BT (Wolves ) Tj ET EMC

    endstream
endobj

101 0 obj
    << /Type /StructElem
        /S /P
        /P 201 0 R
        /Pg 4 0 R
        /K 1
    >>
endobj

111 0 obj
```

```
    << /Type /StructElem
        /S /Lbl
        /P 211 0 R
        /Pg 4 0 R
        /K 11
    >>
endobj

112 0 obj
    << /Type /StructElem
        /S /LBody
        /P 211 0 R
        /Pg 4 0 R
        /K 12
    >>
endobj
```

objects 121-122, 131-132, 141-142, 151-152 and 161-162 referencing MCIDs 21-22, 31-32, 41-42, 51-52, and 61-62 are omitted in the interest of space.

```
201 0 obj
    << /Type /StructElem
        /S /Caption
        /P 400 0 R
        /K [101 0 R]
    >>
endobj

211 0 obj
    << /Type /StructElem
        /S /LI
        /P 400 0 R
        /K [111 0 R 112 0 R]
    >>
endobj

212 0 obj
    << /Type /StructElem
        /S /LI
        /P 301 0 R
        /K [121 0 R 122 0 R]
    >>
endobj

213 0 obj
    << /Type /StructElem
        /S /LI
        /P 301 0 R
        /K [131 0 R 132 0 R]
    >>
endobj

214 0 obj
    << /Type /StructElem
        /S /LI
        /P 400 0 R
        /K [141 0 R 142 0 R]
    >>
endobj

215 0 obj
    << /Type /StructElem
        /S /LI
        /P 400 0 R
        /K [151 0 R 152 0 R]
    >>
endobj

216 0 obj
    << /Type /StructElem
        /S /LI
        /P 302 0 R
        /K [161 0 R 162 0 R]
    >>
endobj

301 0 obj
    << /Type /StructElem
        /S /L
        /P 400 0 R
        /K [212 0 R 213 0 R]
    >>

302 0 obj
    << /Type /StructElem
        /S /L
        /P 400 0 R
        /K [216 0 R]
    >>
endobj

400 0 obj
    << /Type /StructElem
        /S /L
        /K [201 0 R 211 0 R 301 0 R 214 0 R 215 0 R 302 0 R]
    >>
endobj
```

THIS PAGE BLANK

Annex I
(normative)

PDF Versions and Compatibility

I.1 General

The goal of PDF is to enable people to exchange and view electronic documents easily and reliably. Ideally, this means that any conforming reader should be able to display the contents of any PDF file, even if the PDF file was created long before or long after the conforming reader was developed. In reality, new versions of PDF are occasionally introduced to provide additional capabilities not present before. Furthermore, conforming readers may support private extensions to PDF, making some conforming readers more capable than others, depending on what extensions are present.

PDF has been designed to enable users to view everything in the document that the conforming reader understands and to enable the conforming reader to ignore or inform the user about objects not understood. The decision whether to ignore or inform the user is made on a feature-by-feature basis, at the discretion of the confirming reader.

I.2 PDF Version Numbers

The PDF version number identifies a specific version of the Adobe PDF specification. A PDF file is labelled with the version number of the Adobe PDF specification that the file conforms to.

PDF version numbers take the form *M*.*m*, where *M* is the major and *m* the minor version number, each represented as a decimal integer.

The version number for a subsequent version of the PDF specification is formed either by incrementing *m* or by incrementing *M* and setting *m* to zero, as follows:

- The major version is incremented if PDF changes in a way that is not upward-compatible from previous versions. (In practice, this has never happened; the current major version is 1.)
- The minor version is incremented if PDF changes in a way that is upward-compatible from previous versions. (The current minor version is 7.)
- The PDF version number does not change at all if private data is included in a PDF file by one of the extension mechanisms defined in this specification.

The header in the first line of a PDF file specifies a PDF version (see 7.5.2, "File Header"). Starting with PDF 1.4, a PDF version can also be specified in the **Version** entry of the document catalogue, essentially updating the version associated with the file by overriding the one specified in the file header (see 7.7.2, "Document Catalog"). As described in the following paragraphs, the conforming product's behaviour upon opening or saving a document depends on comparing the PDF file's version with the PDF version that the conforming product supports.

A conforming readers shall attempt to read any PDF file, even if the file's version is more recent than that of the conforming reader.

If a conforming reader opens a PDF file with a major version number newer than the version that it supports, it should warn the user that it is unlikely to be able to read the document successfully and that the user cannot change or save the document. Upon the first error that is caused by encountering an unrecognized feature, the conforming reader should notify the user that an error has occurred but that no further errors will be reported.

(Some errors should nevertheless be always reported, including file I/O errors, out-of-memory errors, and notifications that a command has failed.) Processing should continue if possible.

If a conforming reader opens a PDF file that has a minor version number newer than the version that it supports, it should notify the user that the document may contain information the conforming reader does not understand. If the conforming reader encounters an error, it should notify the user that the PDF file’s version is newer than expected, an error has occurred, and no further errors will be reported.

Whether and how the version of a PDF file should change when the document is modified and saved depends on several factors. If the PDF file has a newer version than the conforming product supports, the conforming product should not alter the version—that is, a PDF file’s version should never be changed to an older version. If the PDF file has an older version than the conforming product supports, the conforming product may update the PDF file’s version to match the conforming product’s version. If a user modifies a document by inserting the contents of another PDF file into it, the saved document’s version should be the most recent of the conforming product’s version, the original PDF file's version, and the inserted PDF file’s version.

I.3 Feature Compatibility

When a new version of PDF is defined, many features are introduced simply by adding new entries to existing dictionaries. Earlier versions of conforming readers do not notice the existence of such entries and behave as if they were not there. Such new features are therefore both forward- and backward-compatible. Likewise, adding entries not described in the PDF specification to dictionary objects does not affect the conforming reader’s behaviour. See Annex E for information on how to choose key names that are compatible with future versions of PDF. See 7.12.2, “Developer Extensions Dictionary” for a discussion of how to designate the use of public extensions in PDF file.

In some cases, a new feature is impossible to ignore, because doing so would preclude some vital operation such as viewing or printing a page. For instance, if a page’s content stream is encoded with some new type of filter, there is no way for an earlier version of conforming reader to view or print the page, even though the content stream (if decoded) would be perfectly understood by the reader. There is little choice but to give an error in cases like these. Such new features are forward-compatible but not backward-compatible.

In a few cases, new features are defined in a way that earlier versions of conforming readers will ignore, but the output will be degraded in some way without any error indication. If a PDF file undergoes editing by an earlier version of a conforming product that does not understand some of the features that the file uses, the occurrences of those features may or may not survive.

Annex J
(informative)

FDF Rename Flag Implementation Example

J.1 General

The **Rename** flag is used to specify whether fields imported from the template shall be renamed in the event of name conflicts with existing fields;

J.2 Implementation Example

If the **Rename** flag in the FDF template dictionary is **true**, fields with such conflicting names shall be renamed to guarantee their uniqueness. If **Rename** is **false**, the fields shall not be renamed; this results in multiple fields with the same name in the target document. Each time the FDF file provides attributes for a given field name, all fields with that name shall be updated.

This can be implemented by a conforming product renaming fields by prepending a page number, a template name, and an ordinal number to the field name. The ordinal number corresponds to the order in which the template is applied to a page, with 0 being the first template specified for the page.

EXAMPLE If the first template used on the fifth page has the name Template and has the **Rename** flag set to **true**, fields defined in that template are renamed by prepending the character string P5.Template_0. to their field names.

THIS PAGE BLANK

Annex K
(informative)

PostScript Compatibility — Transparent Imaging Model

K.1 General

Because the PostScript language does not support the transparent imaging model, a conforming reader desiring to print on a PostScript output device needs to have some means for converting the appearance of a document that uses transparency to a purely opaque description.

K.2 Conversion

Converting the contents of a page from transparent to opaque form entails some combination of shape decomposition and prerendering to flatten the stack of transparent objects on the page, performing all the needed transparency computations, and describing the final appearance using opaque objects only. Whether the page contains transparent content needing to be flattened can be determined by straightforward analysis of the page's resources; it is not necessary to analyse the content stream itself. The conversion to opaque form is irreversible, since all information about how the transparency effects were produced is lost.

To perform the transparency computations properly, the conforming reader needs to know the native colour space of the output device. This is no problem when the conforming reader controls the output device directly. However, when generating PostScript output, the conforming reader has no way of knowing the native colour space of the PostScript output device. An incorrect assumption will ruin the calibration of any CIE-based colours appearing on the page. This problem can be addressed in either of two ways:

- If the entire page consists of CIE-based colours, flatten the colours to a single CIE-based colour space rather than to a device colour space. The preferred colour space for this purpose can easily be determined if the page has a group attributes dictionary (**Group** entry in the page object) specifying a CIE-based colour space (see 11.6.6, "Transparency Group XObjects").
- Otherwise, flatten the colours to some assumed device colour space with predetermined calibration. In the generated PostScript output, paint the flattened colours in a CIE-based colour space having that calibration.

Because the choice between using spot colorants and converting them to an alternate colour space affects the flattened results of process colours, a decision needs to be made during PostScript conversion about the set of available spot colorants to assume. (This differs from strictly opaque painting, where the decision can be deferred until the generated PostScript code is executed.)

THIS PAGE BLANK

Annex L
(informative)

Colour Plates

L.1 Colour Plates

This annex consists of figures that logically belong in other parts of this specification. They are collected here so that all colour figures appear together as a sequence of colour plates that may be produced separately from the remainder of the specification.

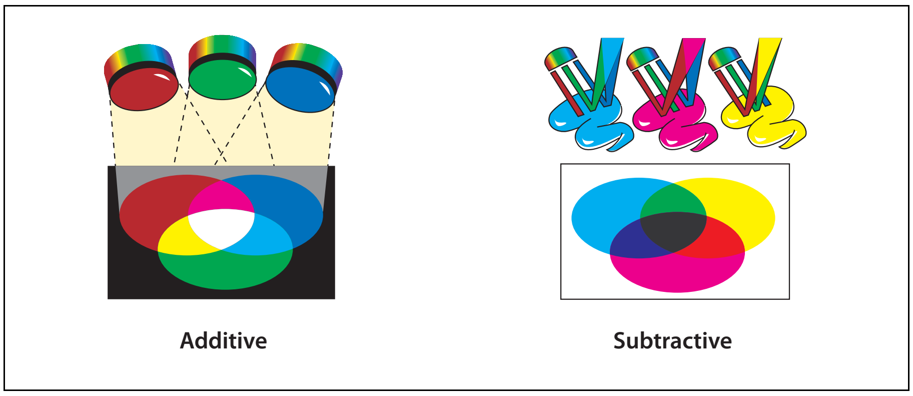


Figure L.1 – Additive and subtractive colour (8.6.4, "Device Colour Spaces")

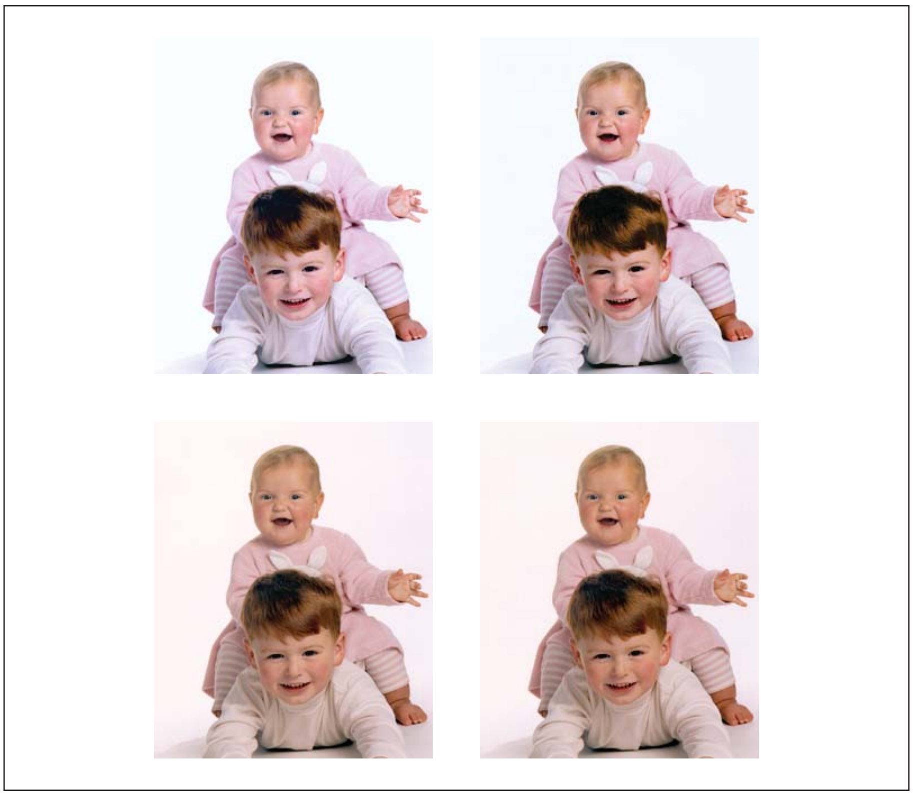


Figure L.2 – Uncalibrated colour (8.6.5, "CIE-Based Colour Spaces")

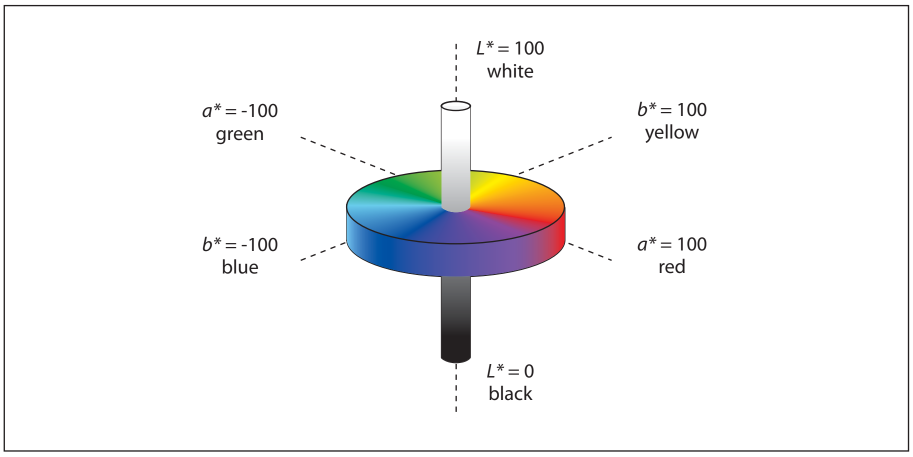


Figure L.3 – Lab colour space (8.6.5.4, "Lab Colour Spaces")

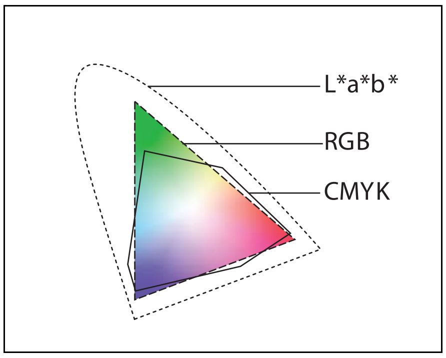


Figure L.4 – Color gamuts (8.6.5.4, "Lab Colour Spaces")

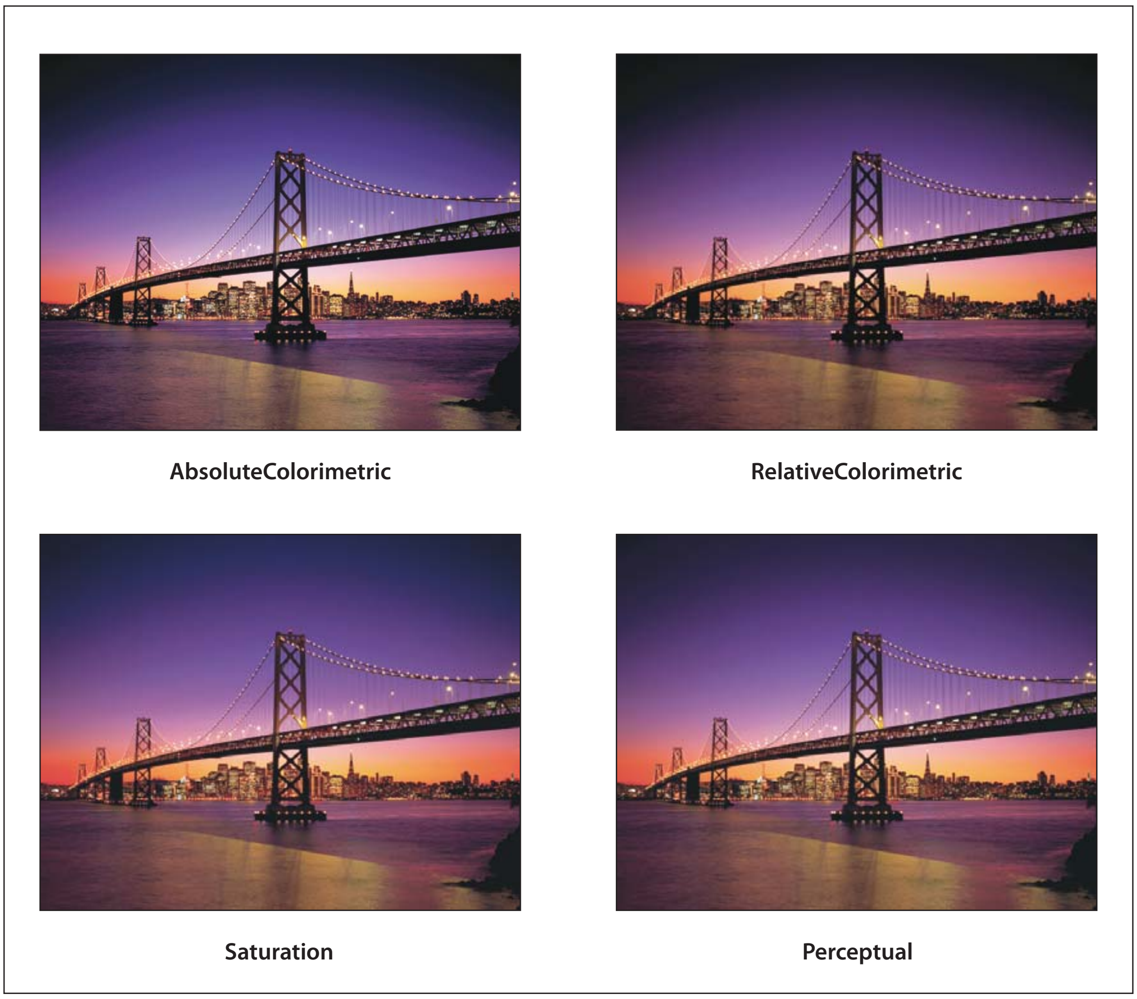


Figure L.5 – Rendering intents (8.6.5.8, "Rendering Intents")

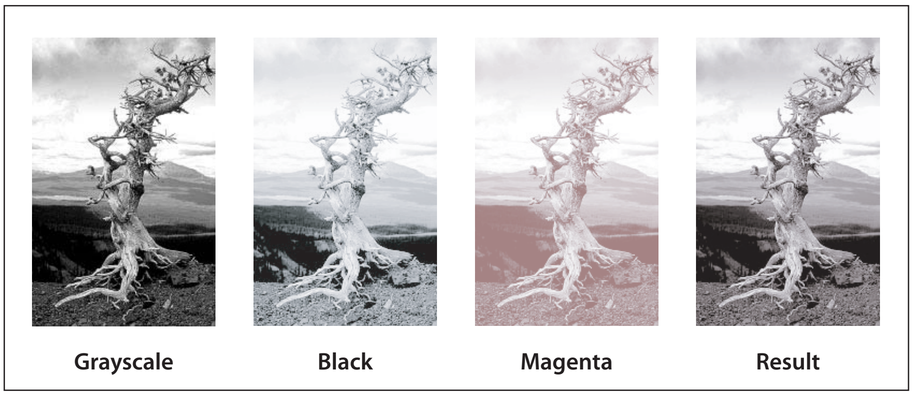


Figure L.6 – Duotone image (8.6.6.5, "DeviceN Colour Spaces")

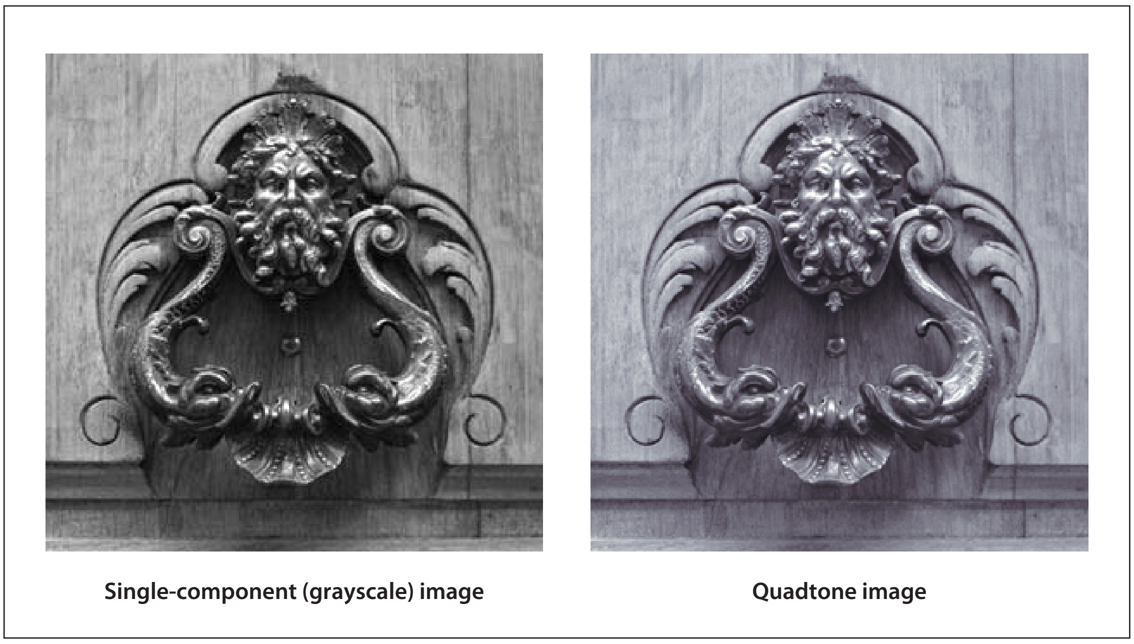


Figure L.7 – Quadtone image (8.6.6.5, "DeviceN Colour Spaces"

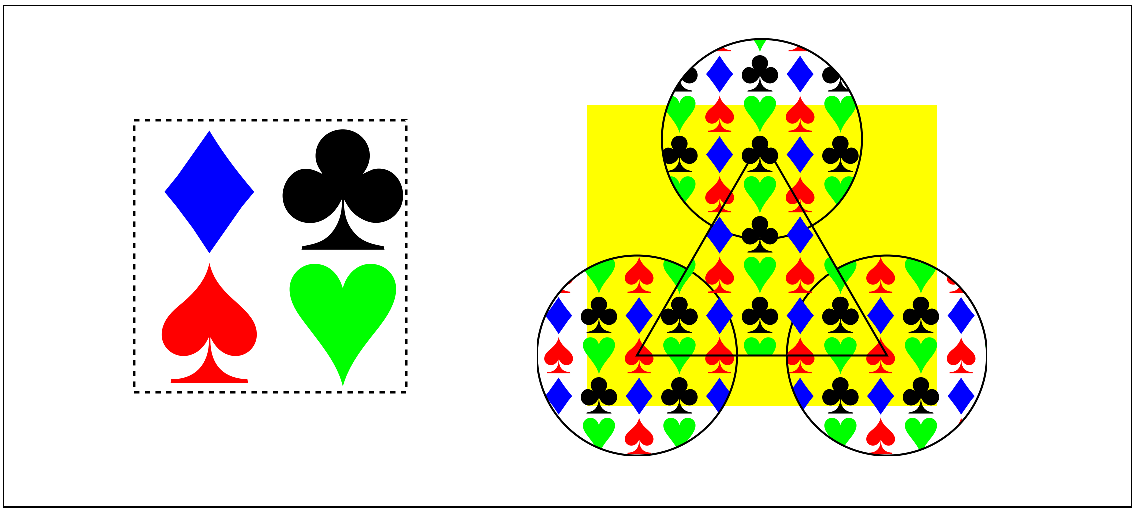


Figure L.8 – Colored tiling pattern (8.7.3.2, "Coloured Tiling Patterns")

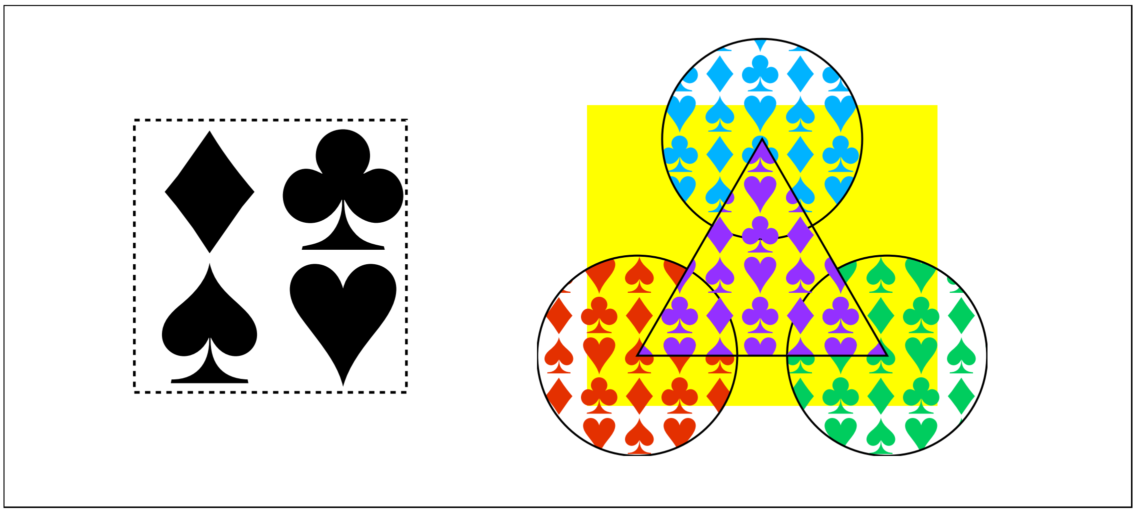
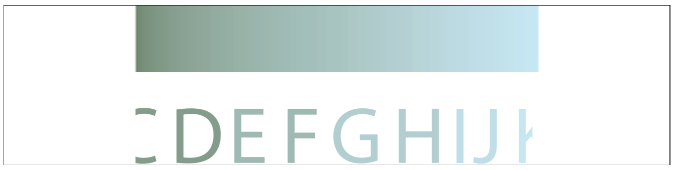
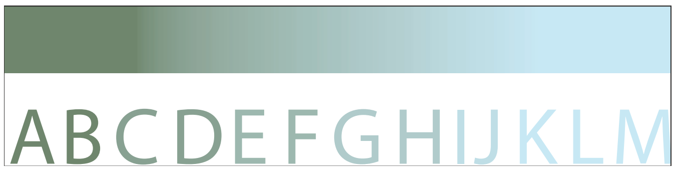


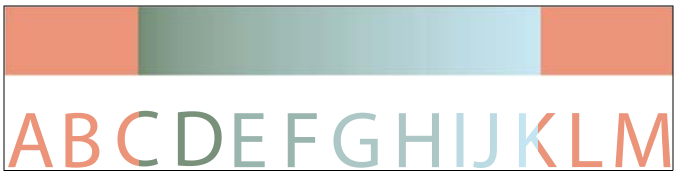
Figure L.9 – Uncoloured tiling pattern (8.7.3.3, "Uncoloured Tiling Patterns")



Extend = [**false false**], Background not specified



Extend = [**true true**], Background not specified



Extend = [**true true**], **Background** not specified

Figure L.10 – Axial shading (8.7.4.5.3, "Type 2 (Axial) Shadings")

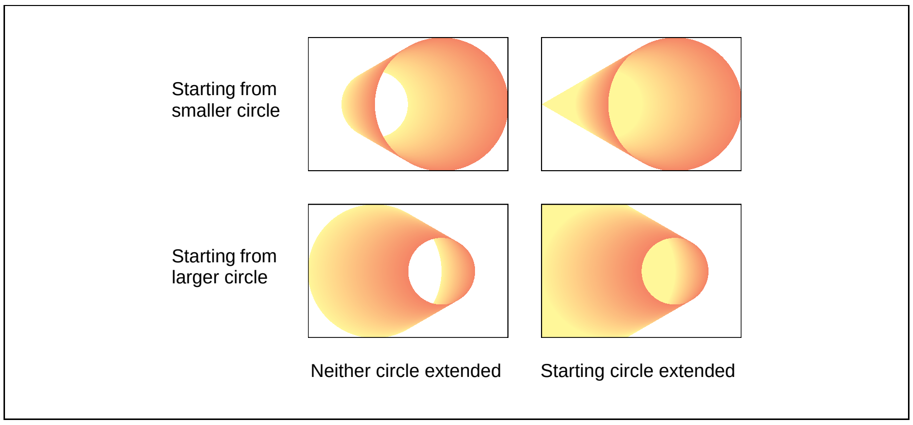


Figure L.11 – Radial shadings depicting a cone (8.7.4.5.4, "Type 3 (Radial) Shadings")

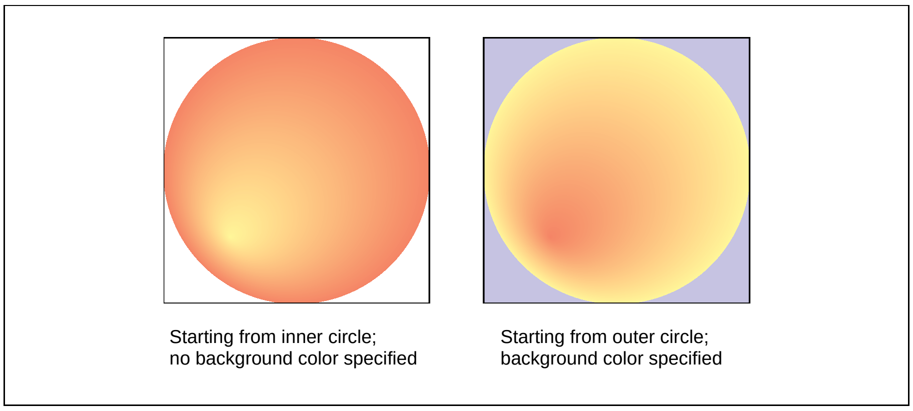

Starting from inner circle;
no background color specified
Starting from outer circle;
background color specified

Figure L.12 – Radial shadings depicting a sphere (8.7.4.5.4, "Type 3 (Radial) Shadings")

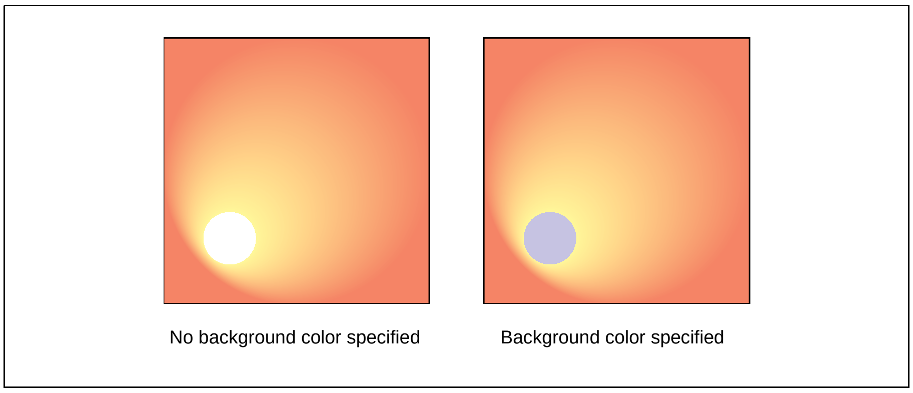


Figure L.13 – Radial shadings with extension (8.7.4.5.4, "Type 3 (Radial) Shadings")

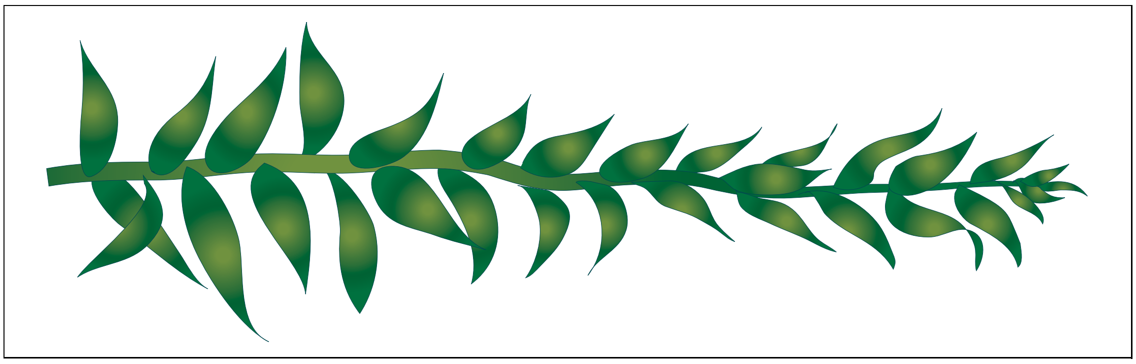


Figure L.14 – Radial shading effect (8.7.4.5.4, "Type 3 (Radial) Shadings")

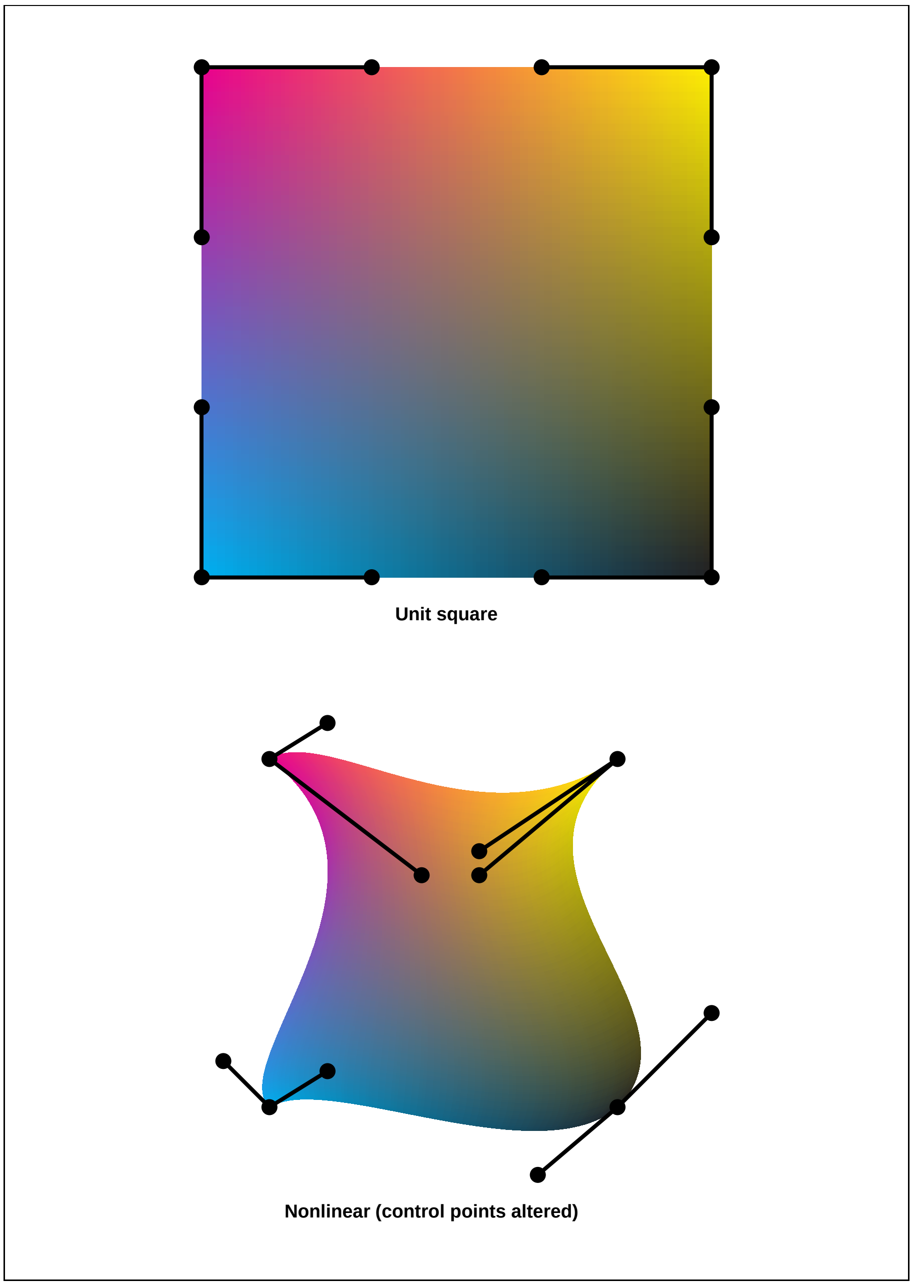


Figure L.15 – Coons patch mesh (8.7.4.5.7, "Type 6 Shadings (Coons Patch Meshes)")

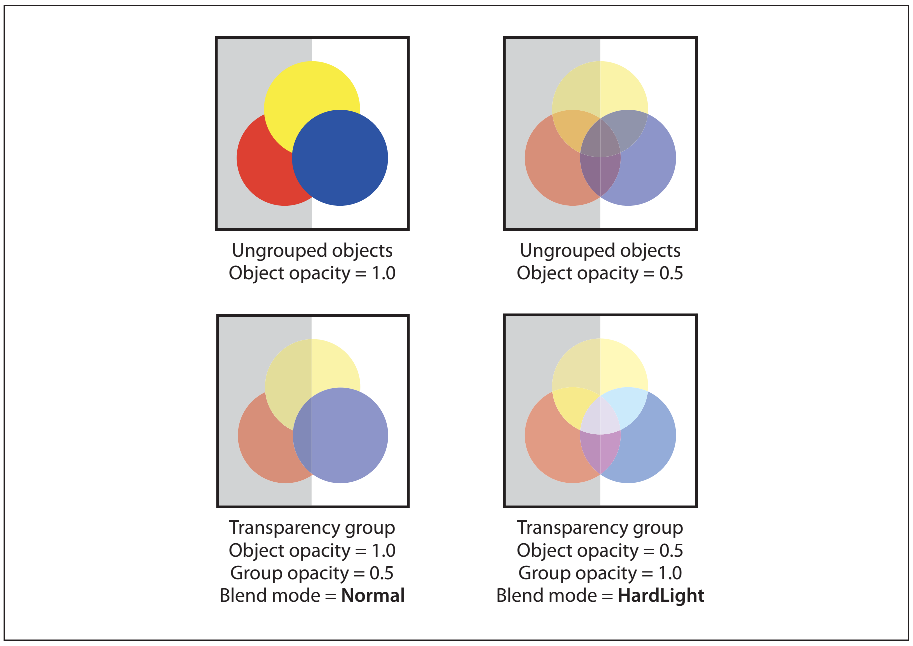


Figure L.16 – Transparency groups (11.2, "Overview of Transparency")

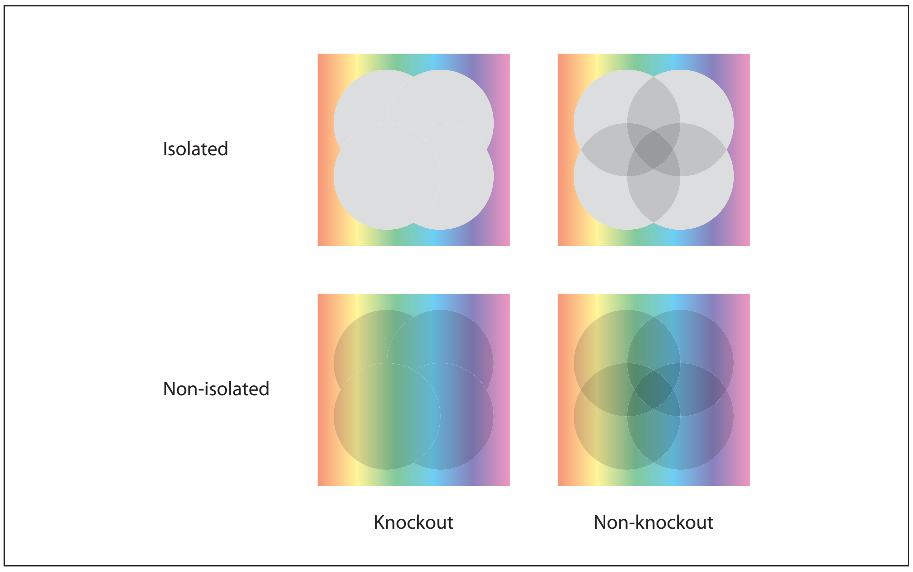


Figure L.17 – Isolated and knockout groups (11.4.5, "Isolated Groups" and 11.4.6, "Knockout Groups")

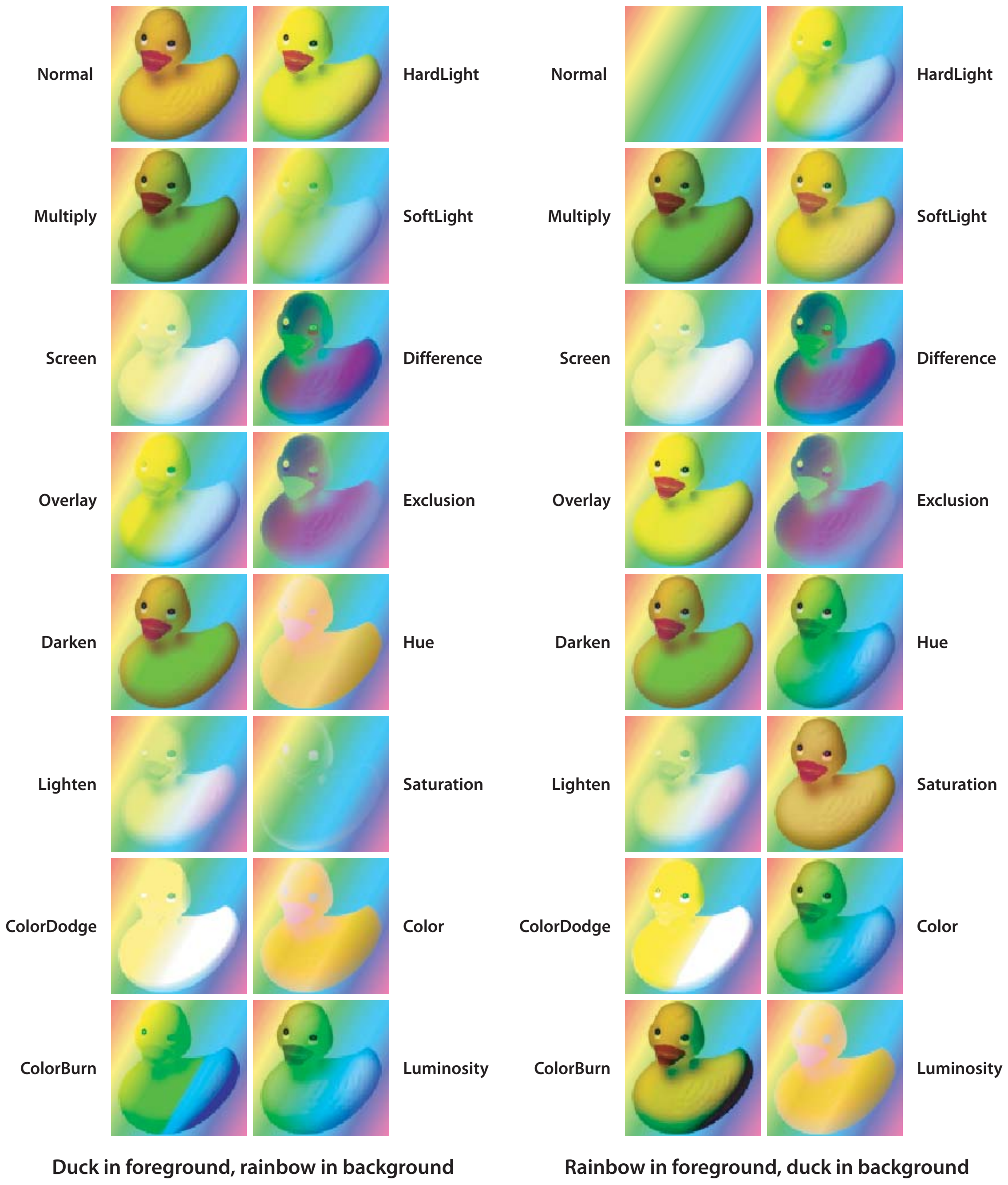
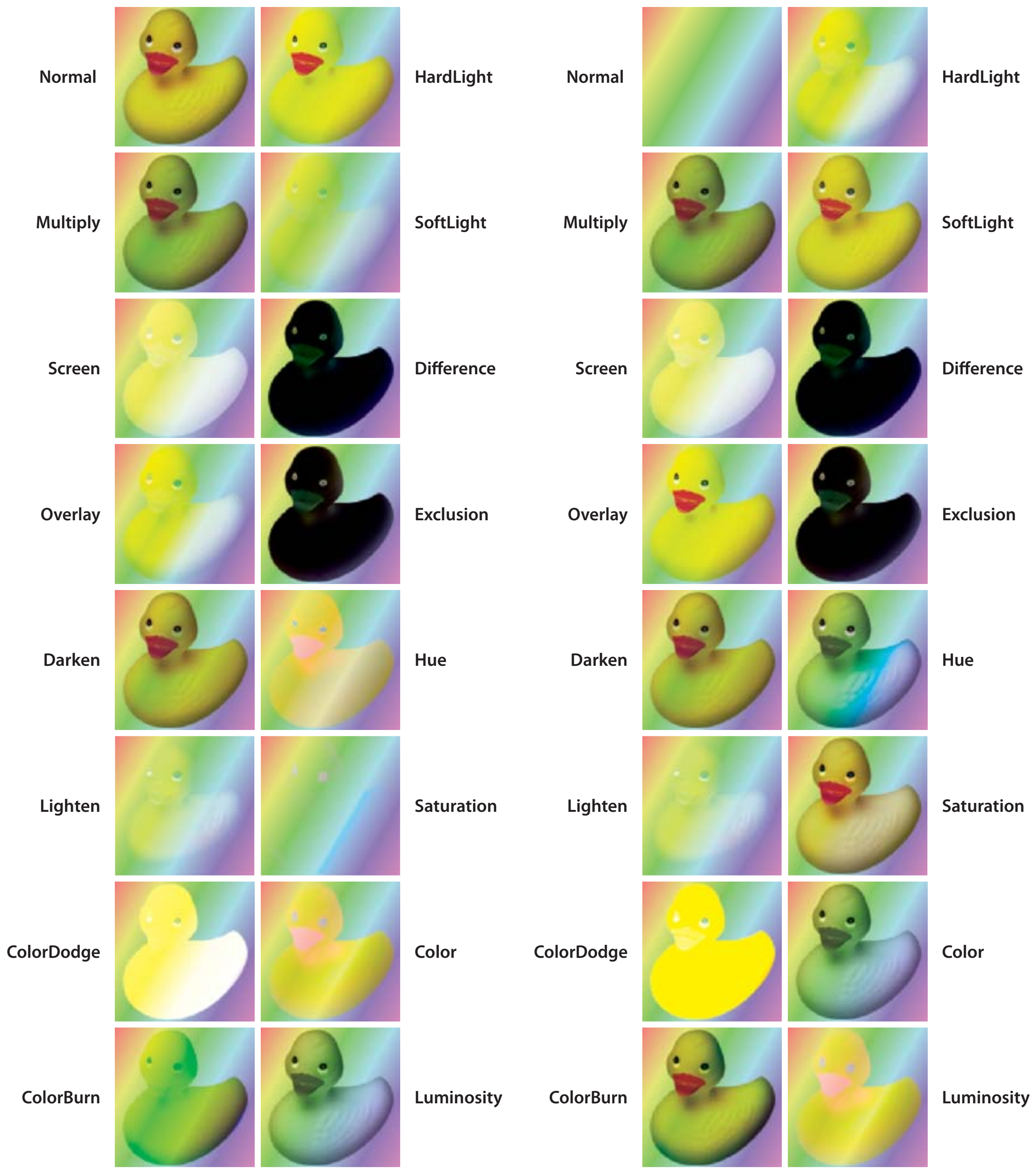


Figure L.18 – RGB blend modes (11.3.5, "Blend Mode")



Duck in foreground, rainbow in background

Rainbow in foreground, duck in background

Figure L.19 – CMYK blend modes (11.3.5, "Blend Mode")

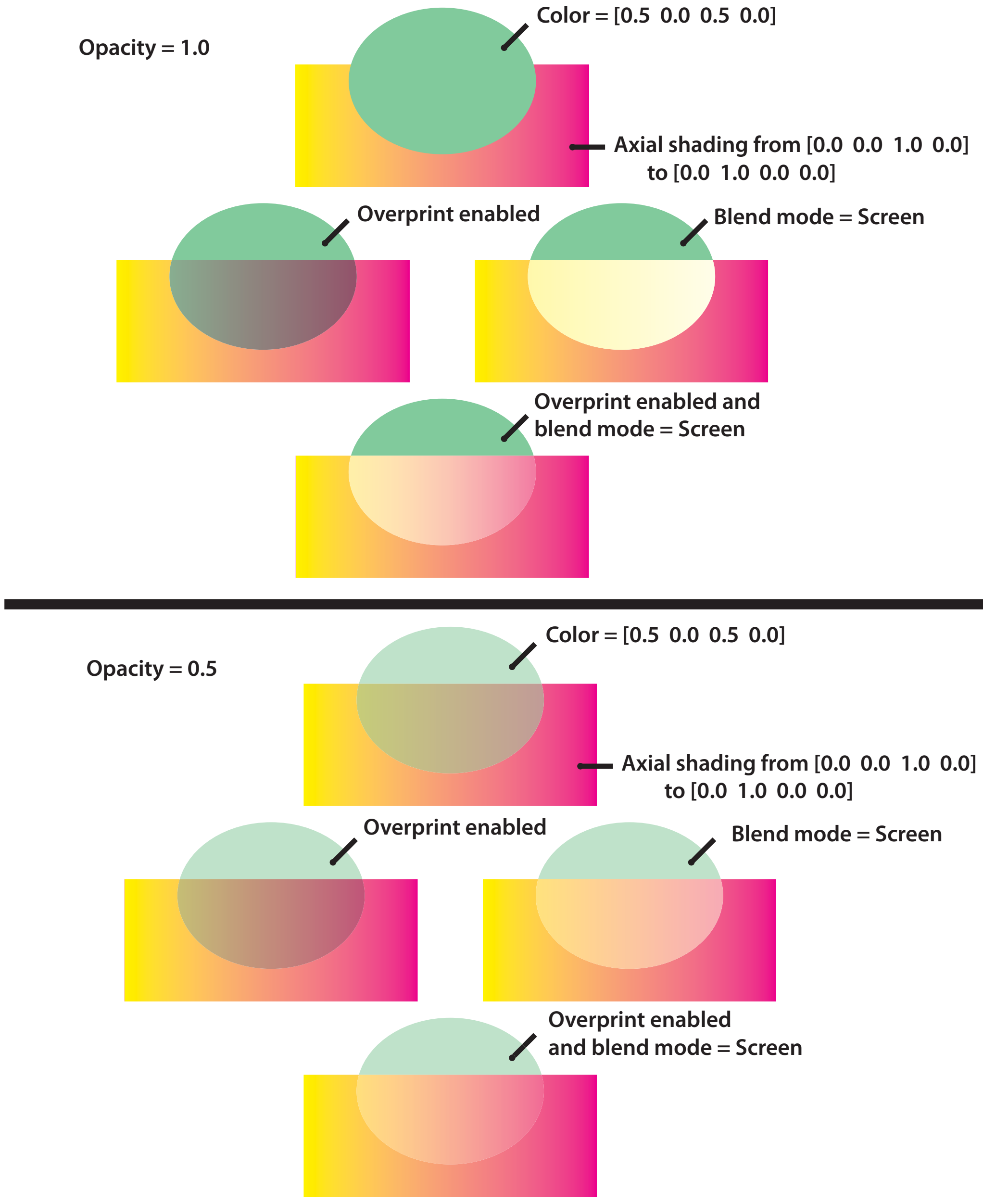

Opacity = 1.0
Color = [0.5 0.0 0.5 0.0]
Axial shading from [0.0 0.0 1.0 0.0] to [0.0 1.0 0.0 0.0]
Overprint enabled
Blend mode = Screen
Overprint enabled and blend mode = Screen
Opacity = 0.5
Color = [0.5 0.0 0.5 0.0]
Axial shading from [0.0 0.0 1.0 0.0] to [0.0 1.0 0.0 0.0]
Overprint enabled
Blend mode = Screen
Overprint enabled and blend mode = Screen

Figure L.20 – Blending and overprinting (11.7.4.3, "Compatibility with Opaque Overprinting")

Bibliography

This Bibliography provides details on books and documents, from ISO, AIIM and other sources, that pertain to this standard.

[1] ISO 15930-1:2001, *Graphic technology — Prepress digital data exchange — Use of PDF — Part 1: Complete exchange using CMYK data (PDF/X-1 and PDF/X-1a).*

[2] ISO 15930-3:2002, *Graphic technology — Prepress digital data exchange — Use of PDF — Part 3: Complete exchange suitable for colour-managed workflows (PDF/X-3).*

[3] ISO 15930-4:2003, *Graphic technology — Prepress digital data exchange using PDF — Part 4: Complete exchange of CMYK and spot colour printing data using PDF 1.4 (PDF/X-1a).*

[4] ISO 15930-5:2003, *Graphic technology — Prepress digital data exchange using PDF — Part 5: Partial exchange of printing data using PDF 1.4 (PDF/X-2)*.

[5] ISO 15930-6:2003, *Graphic technology — Prepress digital data exchange using PDF — Part 6: Complete exchange of printing data suitable for colour-managed workflows using PDF 1.4 (PDF/X-3).*

[6] ISO 19005-1:2005, *Document management — Electronic document file format for long-term preservation -- Part 1: Use of PDF 1.4 (PDF/A-1)* .

[7] ISO 24517-1:2007, *Document management — Engineering document format using PDF — Part 1: Use of PDF 1.6 (PDF/E-1).*

[8] *PDF Reference, First Edition*, version 1.0 (June 1993), Addison-Wesley, 0-201-62628-4.

[9] *PDF Reference, First Edition Revised*, version 1.1 (March 1996), Adobe Systems Incorporated.

[10] *PDF Reference, First Edition Revised*, version 1.2 (November 1996), Adobe Systems Incorporated.

[11] *PDF Reference, Second Edition*, version 1.3 (July 2000), Addison-Wesley, ISBN 0-201-61588-6.

[12] *PDF Reference, Third Edition*, version 1.4 (November 2001), Addison-Wesley, ISBN 0-201-75839-3.

[13] *PDF Reference, Fourth Edition*, version 1.5 (August 2003), Adobe Systems Incorporated (website only).

[14] *PDF Reference, Fifth Edition*, version 1.6 (December 2004), Adobe Press, ISBN 0-321-30474-8.

[15] *PostScript Language Reference*, Third Edition, Addison-Wesley, Reading, MA, 1999.

[16] Technical Note #5001, *PostScript Language Document Structuring Conventions Specification, Version 3.0,* Adobe Systems Incorporated.

[17] Technical Note #5044, *Color Separation Conventions for PostScript Language Programs,* Adobe Systems Incorporated.

[18] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983. Includes a discussion of balanced trees.

[19] Apple Computer, Inc., *TrueType Reference Manual*. Available on Apple’s Web site at <http://developer.apple.com/fonts/TTRefMan/>.

[20] Arvo, J. (ed.), *Graphics Gems II*, Academic Press, 1994. The section “Geometrically Continuous Cubic Bézier Curves” by Hans-Peter Seidel describes the mathematics used to smoothly join two cubic Bézier curves.

[21] *Cascading Style Sheets, level 2 (CSS2) Specification*, <http://www.w3.org/TR/REC-CSS2/>.

[22] *CIP4*. See International Cooperation for the Integration of Processes in Prepress, Press and Postpress.

[23] Ecma International, Standard ECMA-363, *Universal 3D File Format, 1st Edition*. This document is available at <http://www.ecma-international.org/>.

[24] *Extensible Stylesheet Language (XSL) 1.0,* <http://www.w3.org/TR/xsl/>.

[25] Fairchild, M. D., *Color Appearance Models*, Addison-Wesley, Reading, MA, 1997. Covers color vision, basic colorimetry, color appearance models, cross-media color reproduction, and the current CIE standards activities. Updates, software, and color appearance data are available at <http://www.cis.rit.edu/people/faculty/fairchild/CAM.html>.

[26] Foley, J. D. et al., *Computer Graphics: Principles and Practice,* Addison-Wesley, Reading, MA, 1996. (First edition was Foley, J. D. and van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.) Covers many graphics-related topics, including a thorough treatment of the mathematics of Bézier cubics and Gouraud shadings.

[27] Glassner, A. S. (ed.), *Graphics Gems*, Academic Press, 1993. The section “An Algorithm for Automatically Fitting Digitized Curves” by Philip J. Schneider describes an algorithm for determining the set of Bézier curves approximating an arbitrary set of user-provided points. Appendix 2 contains an implementation of the algorithm, written in the C programming language. Other sections relevant to the mathematics of Bézier curves include “Solving the Nearest-Point-On-Curve Problem” and “A Bézier Curve-Based Root-Finder,” both by Philip J. Schneider, and “Some Properties of Bézier Curves” by Ronald Goldman. The source code appearing in the appendix is available via anonymous FTP, as described in the preface to *Graphics Gems III* (Kirk, D. (ed.), *Graphics Gems III*, Academic Press, 1994).

[28] Hewlett-Packard Corporation, *PANOSE Classification Metrics Guide*. Available on the Agfa Monotype Web site at <http://www.agfamonotype.com/printer/pan1.asp>.

[29] *HTML 4.01 Specification,* <http://www.w3.org/TR/html401/>.

[30] Hunt, R. W. G., *The Reproduction of Colour*, 5th ed., Fisher Books, England, 1996. A comprehensive general reference on color reproduction; includes an introduction to the CIE system.

[31] Institute of Electrical and Electronics Engineers, *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754-1985).

[32] International Cooperation for the Integration of Processes in Prepress, Press and Postpress (CIP4), *JDF Specification, Version 1.2*. Available through the CIP4 Web site at <http://www.cip4.org>.

[33] Kirk, D. (ed.), *Graphics Gems III*, Academic Press, 1994. The section “Interpolation Using Bézier Curves” by Gershon Elber contains an algorithm for calculating a Bézier curve that passes through a user-specified set of points. The algorithm uses not only cubic Bézier curves, which are supported in PDF, but also higher-order Bézier curves. The appendix contains an implementation of the algorithm, written in the C programming language. The source code appearing in the appendix is available via anonymous FTP, as described in the book’s preface .

[34] Lunde, K., *CJKV Information Processing*, O’Reilly & Associates, Sebastopol, CA, 1999. Excellent background material on CMaps, character sets, encodings, and the like.

[35] Microsoft Corporation, *TrueType 1.0 Font Files Technical Specification*. Available at <http://www.microsoft.com/typography/tt/tt.htm>.

[36] Porter, T. and Duff, T., “Compositing Digital Images,” *Computer Graphics*, Vol. 18 No. 3, July 1984. *Computer Graphics* is the newsletter of the ACM’s special interest group SIGGRAPH; for more information, see <http://www.acm.org>.

[37] RSA Security, Inc. This document, among others related to encryption and digital signatures, is available at <http://www.rsasecurity.com>: *PKCS #1 - RSA Cryptography Standard* <http://www.rsasecurity.com/rsalabs/node.asp?id=2125>.

[38] *Scalable Vector Graphics (SVG) 1.0 Specification*, <http://www.w3.org/TR/2001/REC-SVG-20010904/>.

[39] *Synchronized Multimedia Integration Language (SMIL 2.0),* <http//:www.w3.org/TR/smil20/>.

[40] *Web Content Accessibility Guidelines 1.0*, <http://www.w3.org/TR/WAI-WEBCONTENT/>.

[41] *XHTML 1.0: The Extensible HyperText Markup Language*, <http://www.w3.org/TR/xhtml1/>.