INTERNATIONAL STANDARD

ISO 32000-2

Second edition
2020-12

Document management — Portable document format —

Part 2: PDF 2.0

Gestion de documents — Format de document portable —

Partie 2: PDF 2.0

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Reference number
ISO 32000-2:2020(E)

© ISO 2020

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

COPYRIGHT PROTECTED DOCUMENT

© ISO 2020

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO 2020 – All rights reserved

Contents

Page

Foreword vii

Introduction viii

1 Scope 1

2 Normative references 2

3 Terms and definitions 7

4 Notation 15
4.1 General 15
4.2 Established notations 15

5 Version designations 17

6 Conformance 18
6.1 General 18
6.2 Conforming PDF documents 18
6.3 PDF processors 18

7 Syntax 20
7.1 General 20
7.2 Lexical conventions 21
7.3 Objects 24
7.4 Filters 34
7.5 File structure 53
7.6 Encryption 71
7.7 Document structure 96
7.8 Content streams and resources 110
7.9 Common data structures 114
7.10 Functions 123
7.11 File specifications 132
7.12 Extensions dictionary 141

8 Graphics 145
8.1 General 145
8.2 Graphics objects 145
8.3 Coordinate systems 149
8.4 Graphics state 156
8.5 Path construction and painting 169
8.6 Colour spaces 177
8.7 Patterns 219

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

8.8 External objects 253
8.9 Images 254
8.10 Form XObjects 270
8.11 Optional content 276

9 Text 293
9.1 General 293
9.2 Organisation and use of fonts 293
9.3 Text state parameters and operators 300
9.4 Text objects 306
9.5 Introduction to font data structures 311
9.6 Simple fonts 313
9.7 Composite fonts 327
9.8 Font descriptors 343
9.9 Embedded font programs 351
9.10 Extraction of text content 355

10 Rendering 360
10.1 General 360
10.2 Raster output device native colour 361
10.3 CIE-Based colour to device colour 361
10.4 Conversions among device colour spaces 361
10.5 Transfer functions 364
10.6 Halftones 366
10.7 Scan conversion details 382
10.8 Rendering for separations 385

11 Transparency 387
11.1 General 387
11.2 Overview of transparency 387
11.3 Basic compositing computations 389
11.4 Transparency groups 402
11.5 Soft masks 414
11.6 Specifying transparency in PDF 415
11.7 Colour space and rendering issues 425

12 Interactive features 437
12.1 General 437
12.2 Viewer preferences 437
12.3 Document-level navigation 441
12.4 Page-level navigation 458
12.5 Annotations 465

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.6 Actions ... 506
12.7 Forms ... 528
12.8 Digital signatures ... 567
12.9 Measurement properties ... 595
12.10 Geospatial features ... 601
12.11 Document requirements ... 606

13 Multimedia features ... 614
13.1 General ... 614
13.2 Multimedia ... 614
13.3 Sounds ... 637
13.4 Movies ... 638
13.5 Alternate presentations ... 640
13.6 3D Artwork ... 642
13.7 Rich media ... 700

14 Document interchange ... 713
14.1 General ... 713
14.2 Procedure sets ... 713
14.3 Metadata ... 714
14.4 File identifiers ... 719
14.5 Page-piece dictionaries ... 719
14.6 Marked content ... 720
14.7 Logical structure ... 722
14.8 Tagged PDF ... 745
14.9 Repurposing and accessibility support ... 796
14.10 Web capture ... 802
14.11 Prepress support ... 814
14.12 Document parts ... 834
14.13 Associated files ... 838
Annex A (informative) Operator Summary ... 844
Annex B (informative) Operators in Type 4 Functions ... 848
Annex C (informative) Advice on maximising portability ... 850
Annex D (normative) Character sets and encodings ... 853
Annex E (normative) Extending PDF ... 877
Annex F (normative) Linearized PDF ... 879
Annex G (informative) Linearized PDF access strategies ... 901
Annex H (informative) Example PDF files ... 905
Annex I (normative) PDF versions and compatibility ... 936
Annex J (informative) XObject comparison ... 938

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Annex K (normative) XFA forms **944**

Annex L (normative) Parent-child relationships between the standard structure elements in the standard structure namespace for PDF 2.0 ..... **947**

Annex M (informative) Differences between the standard structure namespaces **973**

Annex N (informative) Best practice for halftones **974**

Annex O (normative) Fragment identifiers **977**

Annex P (informative) An algorithm to determine the actual blending colour space of a transparency group **980**

Annex Q (normative) Method for determining transparency on a page **982**

Bibliography **984**

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 171, *Document management applications*, Subcommittee SC 2, *Application issues*, in collaboration with Technical Committee ISO/TC 130, *Graphic technology*.

This second edition cancels and replaces the first edition (ISO 32000-2:2017), which has been technically revised.

A list of all the parts of ISO 32000 can be found on the ISO website. Changes from previous parts and editions are listed in the Introduction (clauses 0.3 and 0.4).

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Introduction

0.1 PDF

PDF enables 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®[1] page description language. This PDF Imaging Model enables the description of text and graphics in a device-independent and resolution-independent manner at a complete, precise and professional level. Unlike PostScript, which is a programming language, PDF is based on a structured binary file format that is optimised for high performance in interactive viewing.

PDF 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 also includes data structures such as tagged PDF, XMP and an associated files mechanism, that are useful for document management and content reuse.

PDF files can be created natively in PDF form, converted from other electronic formats. Since PDF supports a wide range of image and compression technologies, it is a suitable format for documents digitised from paper, microform, or other hard copy formats. Businesses, governments, libraries, archives and other institutions and individuals around the world use PDF to represent considerable bodies of important information. Since its introduction in 1993, 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 in which limiting the use of some features of PDF while requiring the use of others, enhances the usefulness of PDF. The following International Standards address specialised uses of PDF:

- PDF/X (ISO 15930) is the industry standard for the intermediate representation of printed material in electronic prepress systems for conventional printing applications.
- PDF/A (ISO 19005) is the industry standard for the archiving of digital documents.
- PDF/UA (ISO 14289) is the industry standard for accessible PDF documents and processors.
- PDF/E (ISO 24517) provides a mechanism for representing engineering documents and exchanging engineering data.
- PDF/VT (ISO 16612-2 and ISO 16612-3) is for high volume printing of personalised documents including variable data.
- ISO 19593 describes a method for storing data in a PDF file that correspond to the processing steps of printed products (such as cutting, folding, glueing, Braille, printed white, and printed varnish).
- ISO 21812 describes how document part metadata in a PDF file can be used to communicate the intended appearance of print products and their components.

As corporations, government agencies, and educational institutions streamline their operations by

[1] PostScript® is a trademark of Adobe Systems Incorporated. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

replacing paper-based workflows 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,
- an extensible metadata model at the document and object level,
- 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 and/or represent data within business systems.

0.2 ISO 32000 and PDF

PDF was developed and specified by Adobe Systems Incorporated beginning in 1993 and continuing until 2007 when ISO 32000-1 was first prepared. The Adobe Systems version PDF 1.7 was the basis for ISO 32000-1. The ISO 32000 series has been created as a multi-part document, of which this is Part 2. This allows future parts to be created without rendering ISO 32000, or applications based on it, obsolete. See clause 5, "Version designations" for how the version numbers of PDF (1.0, 1.1, 1.2, [...] 2.0) relate to one another.

The primary purpose of this document is to define well-formed PDF documents (conforming PDF files).

In carefully specifying what constitutes a well-defined PDF document, it is natural to describe why a particular feature is to be included in the PDF file and what effect it is designed to have on PDF processing software. So, although the primary objective of this document is to describe the content of conforming PDF files, it also serves secondary purposes of defining exactly how a PDF component is constructed, suggesting why a producer might choose to use the various PDF constructs, as well as what behaviour is elicited from software consuming that PDF file. The choice of which specific set of features a particular PDF processor supports is not specified.

PDF files represent electronic documents. Over time, it was natural to add features that take advantage of PDF's nature, and the power of computer viewing devices. The size of the PDF documentation has more than quadrupled since its first introduction, and the number of features that a PDF processor is expected to support has grown to be large.

0.3 Changes introduced in ISO 32000-2:2017

Starting with ISO 32000-2:2017 (PDF 2.0) the term "conforming reader" is no longer used. The terms "interactive PDF processor", "PDF reader" and "PDF writer" are used instead, and have a conditional

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

conformance definition. See 6, "Conformance" for further discussion of this change.

This document includes many changes from ISO 32000-1:2008, however only significant new features are marked as being new in PDF 2.0.

PDF 2.0 includes the following new features:

- 7.6.7, "Unencrypted wrapper document"
- 8.6.5.9, "Use of black point compensation";
- 12.5.6.24, "Projection annotations";
- 12.8.3.4, "CAdES signatures as used in PDF";
- 12.8.4, "Long term validation of signatures";
- 12.8.4.3, "Document Security Store (DSS)" and 12.8.5, "Document timestamp (DTS) dictionary";
- 12.10, "Geospatial features";
- 13.7, "Rich media" annotations;
- 14.7.4, "Namespaces" for tagged PDF;
- 14.9.6, "Pronunciation hints";
- 14.12, "Document parts";
- 14.13, "Associated files";
- Support for PRC (see 13.6, "3D Artwork");
- Support for UTF-8.

PDF 2.0 adds many new capabilities to existing features in PDF, including:

- Transparency and blend mode attributes for annotations;
- Stamp Annot intent;
- Polygon/Polyline real paths;
- 256-bit AES encryption;
- ECC-based certificates;
- Unicode-based passwords;
- Document requirement extensions;
- New value for tab order of fields and annotations;
- Page-level OutputIntents;
- Referenced (external) OutputIntents;
- Thumbnails for embedded files;
- Halftone Origin (HTO);
- Measurement & Point Data for image & form XObjects;
- L (length) key for inline image data;
- Viewer preferences enforcement (of print scaling);

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- 3D measurements;
- GoToDp action;
- RichMediaExecute action;
- Extension to GoTo and GoToR to support linking to a specific structure element;
- Extension to Signature Field Locks and Signature Seed Values;
- Extensions to 3D viewing conditions, incl. transparency;
- Ref (reference) structure element property;
- PageNum and Bates artifact types;
- New list types for structured lists;
- "Short" (short name) attribute for table header cells
- Extensions to OutputIntents (MixingHints and SpectralData).

The following clauses have been substantially rewritten for PDF 2.0:

- 7.4.7, "JBIG2Decode filter";
- 10.1 – 10.3, "Rendering";
- 11, "Transparency";
- 12.8, "Digital signatures";
- 14.3, "Metadata";
- 14.8, "Tagged PDF";
- 14.9, "Repurposing and accessibility support".

PDF 2.0 includes many important corrections, extensions and clarifications for existing features, including:

- Corrections for many typing errors including bad symbols and truncated formulae.
- Updates and changes in normative references and the bibliography.
- Improved cross referencing for clauses, tables and figures within this document.
- Clarification for processing dashed and degenerate lines, clarification for processing text objects and blending colour spaces within the transparency framework, clarifications and enhancements for annotation appearances, stamp annotations extension and polyline annotation enhancement.
- Strengthened encryption including introduction of elliptic curve cryptography, more control over forms tab ordering, enforced viewer preferences, rich text, improvements to digital signatures for long term signatures, 3D viewing improvements including 3D projections, revised blend formulae for **ColorBurn** and **ColorDodge**, additional structure tags to improve accessibility, requirement for metadata streams to be XMP and support for hyperlinks in rich text.
- Clarification for PDF version numbering, resource inheritance, required and optional signature dictionary **SubFilter** keys, artifacts, developer-defined extensions, word breaking and page sizes, which file to show when first opening a collection, scope of header attributes, precedence of CID font widths, when a **CIDToGIDMap** is used with Type 2 CID fonts, deprecate sound and movie actions and annotations in favour of newer methods, rendering

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

intent and ImageMask, precedence of Type 1 encoding methods, the wording used to define delimiters with respect to << and >>, Identity CMaps and CIDFonts, a special case when closing and filling a path, that clipping follows filling rules and that operating on an undefined path generates an error.

- Clarification and terminology improvements among Type 1, TrueType, CFF and OpenType fonts; thumbnails for embedded files.
- Specification of XFA used for rich text in annotations.
- The rewrite of 14.8, "Tagged PDF" includes clarification of the parent/child relationships between tags, simplifies and extends the standard tag set, and adds the use of namespaces for custom tag sets (see also 14.7.4, "Namespaces" for new namespace functionality).

Some features present in earlier versions of PDF have been deprecated in PDF 2.0, including:

- XFA (incl. NeedsRendering);
- Movie, Sound and TrapNet annotations;
- Movie and Sound actions;
- Info dictionary;
- Assistive technology restrictions via DRM;
- ProcSet;
- OS-specific file specifications;
- OS-specific additions to Launch actions;
- Names for XObjects;
- Names for Fonts;
- Arrays of Blend Modes;
- Alternate Presentations;
- Open prepress interface (OPI);
- CharSet (For Type 1 fonts);
- CIDSet (for CID fonts);
- Prepress viewer preferences (ViewArea, ViewClip, etc.);
- NeedAppearances;
- adbe.pkcs7.sha1;
- adbe.x509.rsa_sha1;
- Encryption of FDF files;
- Suspects flag in MarkInfo dictionary;
- UR signatures;
- Transfer functions in the graphics state.

0.4 Changes introduced in ISO 32000-2:2020

ISO 32000-2:2020 includes additional corrections and clarifications as indicated in this clause. Precise locations of key changes are also indicated by the text string “(2020)”. In addition, F.3 "Linearized PDF

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

document structure" and Annex L "Parent-child relationships between the standard structure elements in the standard structure namespace for PDF 2.0" have been significantly updated. PDF character collections were also updated as follows:

- Adobe-Japan1-6 becomes Adobe-Japan1-7;
- Adobe-CNS1-6 becomes Adobe-CNS1-7;
- Adobe-KR-9 is introduced;
- Adobe-Korea1-2 & Adobe-Japan2-0 were deprecated;
- Adobe-GB1-5 remains unchanged.

ISO 32000-2:2020 also makes several important Normative References changes due to various documents being withdrawn or obsoleted. Additionally some previous Normative References have been moved to the Bibliography:

- ISO 3166-1 is now an undated reference – see 7.9.2.2.2, "Text string language escape sequences";
- The ISO/IEC 14492 dated reference for JBIG2 was updated to the 2019 edition – see 7.4.7, "JBIG2Decode filter";
- The ISO/IEC 14496-22 dated reference for the Open Font format was updated to the 2019 edition – see 9.6.3, "TrueType fonts";
- ISO 15076-1:2010 dated reference for ICC.1 can be supplemented by the Errata list and approved revisions available from the ICC website (http://color.org/icc_specs2.xalter) – see 8.6.5.5, "ICCBased colour spaces";
- The ISO/IEC 15444-1 dated reference for JPEG 2000 was updated to the 2019 edition – see 7.4.9, "JPXDecode filter";
- The ISO/IEC 19444-1 dated reference for XFDF was updated to the 2019 edition;
- This document requires that RFC 3454 (“stringprep”) and RFC 4013 (“SASLprep”) continue to be used to maintain backward compatibility even though these RFCs are marked as obsolete by IETF;
- RFC 6234 (US Secure Hash Algorithms) is replaced by FIPS PUB 180-4;RFC 2083 (Portable Network Graphics) is replaced by ISO/IEC 15948:2004 for PNG predictors – see 7.4.4.4, "LZW and Flate predictor functions";
- ISO/DIS 21757-1 replaces several Adobe, ECMA and ISO publications related to ECMAScript in PDF 2.0 – see 12.6.4.17, "ECMAScript actions";
- This document makes explicit reference to ECMA-363 U3D 3rd edition and not the latest U3D 4th edition;
- PDF character collections are now all referenced to GitHub repositories;
- The TrueType Reference is now undated.

An attempt is being made to keep copies of all references without copyright restrictions available for free download on the following website: https://www.pdfa.org/iso-32000-normative-references/.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Document management — Portable document format —

Part 2:
PDF 2.0

IMPORTANT — The electronic file of this document contains colours which are considered to be useful for the correct understanding of the document. Users who need a paper copy of this document will therefore benefit from using a colour printer.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

1 Scope

This document 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 developers of software that creates PDF files (PDF writers), software that reads existing PDF files and (usually) interprets their contents for display (PDF readers), software that reads and displays PDF content and interacts with the computer users to possibly modify and save the PDF file (interactive PDF processors) and PDF products that read and/or write PDF files for a variety of other purposes (PDF processors). (PDF writers and PDF readers are more specialised classifications of interactive PDF processors and all are PDF processors).

This document does not specify the following:

- specific processes for converting paper or electronic documents to the PDF file format;
- specific technical design, user interface 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 PDF processors;
- required computer hardware and/or operating system.

© ISO 2020 – All rights reserved

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

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

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

ISO/IEC 10646, *Information technology – Universal Coded Character Set (UCS)*

ISO/IEC 10918 (all parts), *Information Technology – Digital Compression and Coding of Continuous-Tone Still Images: Requirements and guidelines* (informally known as the JPEG standard, for the Joint Photographic Experts Group, the ISO group that developed the standard)

ISO/IEC 14492:2019, *Information technology – Lossy/lossless coding of bi-level images*

ISO/IEC 14496-22:2019, *Information technology – Coding of audio-visual objects — Part 22: Open Font Format*

ISO 14739-1, *Document management – 3D use of Product Representation Compact (PRC) format — Part 1: PRC 10001*

ISO 15076-1:2010, *Image technology colour management – Architecture, profile format and data structure — Part 1: Based on ICC.1:2010*

ISO/IEC 15444-1:2019, *Information technology — JPEG 2000 image coding system: Core coding system*

ISO/IEC 15444-2:2004, *Information technology — JPEG 2000 image coding system: Extensions*

ISO/IEC 15948:2004, *Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification*

ISO 16684-1, *Extensible metadata platform (XMP) specification — Part 1: Data model, serialization and core properties*

ISO 17972-4, *Graphic technology — Colour data exchange format (CxF/X) — Part 4: Spot colour characterisation data (CxF/X-4)*

ISO 18619, *Image technology colour management – Black point compensation*

ISO 19162, *Geographic information – Well known text representation of coordinate reference systems*

ISO 19444-1:2019, *Document management – XML Forms Data Format – Part 1: Use of ISO 32000-2 (XFDF 3.0)*

ISO/DIS 21757-1, *Document management — ECMAScript for PDF — Part 1: Use of ISO 32000-2 (PDF 2.0)*

IEC 61966-2-1 ed1.0 (1999-10), *Multimedia systems and equipment — Colour measurement and*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

management — Part 2-1: Colour management — Default RGB colour space — sRGB (with Amendment 1 IEC 61966-2-1-am1 ed1.0 (2003-01))

ANSI X9.62-2005, *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American National Standards Institute*

INCITS 4-1986 (R2017), *Information Systems - Coded Character Sets - 7-Bit Standard Code for Information Interchange (7-Bit ASCII)*"

PostScript Language Third Edition, (February, 1999), Adobe Systems Incorporated[1]

Adobe Glyph List, Adobe Systems Incorporated[2]

Adobe Glyph List for New Fonts, Adobe Systems Incorporated[2]

Adobe PDF Signature Build Dictionary Specification v.1.4, (March 2008), Adobe Systems Incorporated[1]

Adobe TIFF Revision 6.0, Final, (June 1992), Adobe Systems Incorporated[1]

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

Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3, (January 2012), Adobe Systems Incorporated[1]

Adobe-Japan1-7 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe-GB1-5 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe-CNS1-7 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe-Korea1-2 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe-Japan2-0 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe-KR-9 Character Collection for CID-Keyed Fonts, https://github.com/adobe-type-tools/cmap-resources

Adobe Technical Note #5014, *Adobe CMap and CID Font Files Specification, Version 1.0*, (8 October 1996), Adobe Systems Incorporated[1]

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

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

[1] These documents can be found at the PDF Association at https://www.pdfa.org/iso-32000-normative-references/ as well as via the Adobe Systems Incorporated Web site http://www.adobe.com/.

[2] These documents can be found at https://github.com/adobe-type-tools/agl-aglfn.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Systems Incorporated[1]

Adobe Technical Note #5176, *The Compact Font Format Specification, version 1.0*, (December 2003), Adobe Systems Incorporated[1]

Adobe Technical Note #5177, *The Type 2 Charstring Format*, (March 2000), Adobe Systems Incorporated[1]

Adobe Technical Note #5620, *Portable Job Ticket Format, Version 1.1*, (April 1999), Adobe Systems Incorporated[1]

Adobe Technical Note #5660, *Open Prepress Interface (OPI) Specification, Version 2.0*, (January 2000), Adobe Systems Incorporated[1]

FIPS PUB 180-4, *Secure Hash Standard (SHS)*, (August 2015), Federal Information Processing Standards, https://doi.org/10.6028/NIST.FIPS.180-4 [3]

FIPS PUB 186-4, *Digital Signature Standard, describes DSA signatures*, (July 2013), Federal Information Processing Standards, https://doi.org/10.6028/NIST.FIPS.186-4 [3]

FIPS PUB 197, *Advanced Encryption Standard (AES)*, (November 2001), Federal Information Processing Standards, https://doi.org/10.6028/NIST.FIPS.197 [3]

RFC 1321, *The MD5 Message-Digest Algorithm*, (April 1992), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc1321 [4]

RFC 1950, *ZLIB Compressed Data Format Specification, Version 3.3*, (May 1996), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc1950 [4]

RFC 1951, *DEFLATE Compressed Data Format Specification, Version 1.3*, (May 1996), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc1951[4]

RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, (November 1996), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc2045 [4]

RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, (November 1996), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc2046 [4]

RFC 2315, *PKCS #7: Cryptographic Message Syntax Version 1.5* (March 1998), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc2315 [4]

RFC 3161, *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*, (August 2001), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc3161 [4]

RFC 3454, *Preparation of International Strings ("stringprep")*, (December 2002), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc3454 [4]

RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, (January 2005), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc3986 [4]

[3] These FIPS documents are also available from NIST at http://www.nist.gov/itl/fipscurrent.cfm.

[4] These documents are also available as Internet Engineering Task Force RFCs at https://www.ietf.org/

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

RFC 4013, *SASLprep: Stringprep Profile for User Names and Passwords*, (February 2005), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc4013 [4]

RFC 5035, *Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility*, (August 2007), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5035 [4]

RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,* (May 2008), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5280 [4]

RFC 5480, *Elliptic Curve Cryptography Subject Public Key Information*, (March 2009), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5480 [4]

BCP 47 (RFC 5646), *Tags for Identifying Languages*, (September 2009), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5646 [4]

RFC 5652, *Cryptographic Message Syntax (CMS)*, (September 2009), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5652 [4]

RFC 5755, *An Internet Attribute Certificate Profile for Authorization*, (January 2010), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5755 [4]

RFC 5816, *ESSCertIDv2 Update for RFC 3161*, (March 2010), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc5816 [4]

RFC 6960, X.509 *Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP*, (June 2013), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc6960 [4]

RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, (June 2014), https://dx.doi.org/10.17487/rfc7231

RFC 8018, *PKCS #5: Password-Based Cryptography Specification Version 2.1*, (January 2017), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc8018 [4]

RFC 8118, *The application/PDF Media Type, Internet Engineering Task Force*, (March 2017), Internet Engineering Task Force (IETF), https://dx.doi.org/10.17487/rfc8118 [4]

ETSI EN 319 122-1 V1.1.1 (2016-04) *Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 1: Building blocks and CAdES baseline signatures*, European Telecommunications Standards Institute[5]

ETSI EN 319 122-2 V1.1.1 (2016-04) *Electronic Signatures and Infrastructures (ESI); CAdES digital signatures; Part 2: Extended CAdES signatures*, European Telecommunications Standards Institute[5]

ETSI EN 319 142-1 V1.1.1 (2016-04) *Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 1: Building blocks and PAdES baseline signature*, European Telecommunications Standards Institute[5]

ETSI EN 319 142-2 V1.1.1 (2016-04) *Electronic Signatures and Infrastructures (ESI); PAdES digital signatures; Part 2: Additional PAdES signatures profiles*, European Telecommunications Standards Institute[5]

[5] These documents are available from http://www.etsi.org

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

EPSG *Geospatial Coordinate System Reference Codes*, administered by the International Association of Oil and Gas Producers (OGP), http://www.epsg.org

Extensible Markup Language (XML) 1.0, Fifth Edition, November 2008, World Wide Web Consortium (W3C), https://www.w3.org/TR/xml/

Accessible Rich Internet Applications (ARIA) 1.1, December 2017, World Wide Web Consortium (W3C), https://www.w3.org/TR/wai-aria-1.1/

Mathematical Markup Language (MathML) Version 3.0, 2nd Revision, April 2014, World Wide Web Consortium (W3C), https://www.w3.org/TR/MathML3/

Japanese Industrial Standard (JIS) X 4051:2004

PANOSE Classification Metrics Guide, (February 1997), Hewlett-Packard Corporation, *https://monotype.github.io/panose/pan1.htm*

Pronunciation Lexicon Specification (PLS) Version 1.0, October 2008, World Wide Web Consortium (W3C), https://www.w3.org/TR/pronunciation-lexicon/

ITU Recommendation T.4: Standardization of Group 3 facsimile terminals for document transmission, International Telecommunication Union (ITU), https://www.itu.int/rec/T-REC-T.4/en

ITU Recommendation T.6: Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus, International Telecommunication Union (ITU), https://www.itu.int/rec/T-REC-T.6/en

Standard ECMA-363, Universal 3D File Format, 3rd Edition (U3D), (June 2006), Ecma International

Synchronized Multimedia Integration Language (SMIL 3.0), December 2008, World Wide Web Consortium (W3C), http://www.w3.org/TR/SMIL3/

TrueType Reference Manual, Apple Inc., https://developer.apple.com/fonts/TrueType-Reference-Manual

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

3 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at http://www.iso.org/obp
- IEC Electropedia: available at http://www.electropedia.org/

3.1
approval signature
one or more digital signatures applied to a document, each one allowing the detection of changes to the document and which confirms a signer of the document

3.2
array object
one-dimensional collection of *objects* (3.44) arranged sequentially and implicitly numbered starting at 0

3.3
ASCII American Standard Code for Information Interchange
widely used convention for encoding a specific set of 128 characters as binary numbers defined in INCITS 4-1986 (R2017)

3.4
ASN.1
means of reference to ISO/IEC 8824-1

3.5
binary data
sequence of bytes

3.6
boolean objects
either the keyword **true** or the keyword **false**

3.7
byte
group of 8 binary digits (an 8-bit value) which collectively can be configured to represent one of 256 different values

3.8
CAdES
means of reference to ETSI EN 319 122-1 V1.1.1 and ETSI EN 319 122-2 V1.1.1

3.9
certification signature
one and only one digital signature applied to a document which allows the detection of changes to the

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

document and which confirms the origin or the author of the document and that is able to allow or to block specified actions to be performed when using conformant *signature handlers* (3.61) or *PDF processors* (3.49)

3.10
character
numeric code representing an abstract symbol according to some defined character encoding rule

Note 1 to entry: There are three manifestations of characters in PDF, depending on context:

- A PDF file is represented as a sequence of 8-bit *bytes* (3.7), some of which are interpreted as character codes in the *ASCII* (3.3) character set and some of which are treated as arbitrary *binary data* (3.5) depending upon the context.
- The contents (data) of a string or *stream object* (3.63) in some contexts are interpreted as character codes in the PDFDocEncoding, UTF-8 or UTF-16BE character set.
- The contents of a string within a PDF *content stream* (3.12) in some situations are interpreted as character codes that select *glyphs* (3.29) to be drawn on the page according to a character encoding that is associated with the text font.

3.11
character set
defined set of *characters* (3.10) each assigned a unique character value

3.12
content stream
stream object whose data consists of a sequence of instructions describing the graphical elements to be painted on a page

3.13
cross reference table
data structure that contains the *byte* (3.7) offset start for each of the *indirect objects* (3.33) within the file

3.14
developer
entity, including individuals, companies, non-profits, standards bodies, open source groups, etc., developing standards or software to use and extend this document

3.15
deprecated
a part of ISO 32000 that should not be written into a PDF 2.0 document, and should be ignored by a *PDF processor* (3.49)

Note 1 to entry: In some cases variations on these restrictions on continued use of a deprecated feature are explicitly stated in this document.

Note 2 to entry: Implementers are cautioned that some features that are deprecated in this document could have tighter constraints placed on them, or even be removed completely, in a later version of ISO 32000, or in subset standards such as PDF/X (ISO 15930), PDF/A (ISO 19005), PDF/E (ISO 24517),

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

PDF/VT (ISO 16612-2 and ISO 16612-3) and PDF/UA (ISO 14289).

3.16
dictionary object
associative table containing pairs of objects, the first object being a *name object* (3.39) serving as the key and the second object serving as the value and may be any kind of object including another dictionary

3.17
direct object
object that has not been made into an *indirect object* (3.33)

3.18
document part
set of related pages and/or related sets of pages

EXAMPLE The chapter pages of a book or all sets of pages intended for a recipient.

3.19
document part hierarchy
hierarchical data structure that specifies the organisation of *document parts* (3.18)

3.20
document part metadata
metadata associated with a *document part* (3.18)

3.21
DSA
Digital Signature Algorithm, and means of reference to FIPS PUB 186-4

3.22
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

3.23
end-of-line marker
EOL marker
one or two *character* (3.10) 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

3.24
FDF file
file conforming to the Forms Data Format containing form data or annotations that may be imported into a PDF file

Note 1 to entry: See 12.7.8, "Forms data format".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

3.25
filter
optional part of the specification of a *stream object* (3.63), indicating how the data in the stream should be decoded before it is used

3.26
font
identified collection of graphics that may be *glyphs* (3.29) or other graphical elements

3.27
font program
software program written in a special-purpose language, such as the Type 1, TrueType, or OpenType *font* (3.26) format, that is understood by a specialized font interpreter

3.28
function
special type of object that represents parameterised classes, including mathematical formulas and sampled representations with arbitrary resolution

3.29
glyph
recognizable abstract graphic symbol that is independent of any specific design

[SOURCE: ISO/IEC 9541-1:2012, 3.12]

3.30
graphics state
top of a push down stack of the graphics control parameters that defines the current global framework within which the graphics operators execute

3.31
ICC profile
ISO 15076-1:2010 or any of the ICC.1 specifications

3.32
ICC specification
cross-platform profile format for the creation and interpretation of colour data and means of reference to ISO 15076-1:2010 or any of the ICC.1 specifications

3.33
indirect object
object (3.44) that is labelled with a positive *integer object* (3.34) number followed by a non-negative integer generation number followed by keyword **obj** and having keyword **endobj** after it

3.34
integer object
mathematical integers with an implementation specified interval centred at 0 and written as one or more decimal digits optionally preceded by a sign

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

3.35
interactive PDF processor
special case of a *PDF processor* (3.49) that responds to interactive user controls defined in this document

3.36
JPEG
common image compression data format and means of reference to ISO/IEC 10918

3.37
job definition
information that specifies the production requirements and workflow of a unit of work involving purposing PDF content to one or more messaging channels

3.38
job ticket
electronic specification of process control for print production in either a published or proprietary format

3.39
name object
atomic symbol uniquely defined by a sequence of *characters* (3.10) introduced by a SOLIDUS (/) (2Fh), but the SOLIDUS is not considered to be part of the name

3.40
name tree
similar to a dictionary that associates keys and values but the keys in a name tree are strings and are ordered

3.41
null object
single *object* (3.44) of type null, denoted by the keyword **null**, and having a type and value that are unequal to those of any other object

3.42
number tree
similar to a dictionary that associates keys and values but the keys in a number tree are integers and are ordered

3.43
numeric object
either an *integer object* (3.34) or a *real object* (3.56)

3.44
object
basic data structure from which PDF files are constructed and includes these 9 types: array, boolean, dictionary, integer, name, null, real, stream and string

3.45
object reference

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

object (3.44) value used to allow one object to refer to another; that has the form "<n> <m> **R**" where <n> is an *indirect object* (3.33) number, <m> is its generation number and **R** is the uppercase letter R

3.46
object stream
stream that contains a sequence of PDF *objects* (3.44), except those of **Type** *stream*

3.47
PAdES
means of reference to ETSI EN 319 142-1 V1.1.1 and ETSI EN 319 142-2 V1.1.1

3.48
PDF imaging model
derived from the PostScript page description language and enables the description of high quality text and graphics in a device-independent and resolution-independent manner

3.49
PDF processor
software, hardware or any other active agent that writes, reads, updates or otherwise processes a PDF file which conforms to this document and does so in a manner that conforms to this document

3.50
PDF file
contiguous stream of *binary data* (3.5) that conforms to this document

3.51
PDF reader
PDF processor (3.49) that reads a PDF file

3.52
PDF writer
PDF processor (3.49) that writes a PDF file

3.53
PKCS #7
means of reference to Internet RFC 2315

3.54
PRC
means of reference to ISO 14739-1

3.55
RDF
means of reference to RDFa Core 1.1

3.56
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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

3.57
rectangle
specific *array object* (3.2) used to describe locations on a page and bounding boxes for a variety of *objects* (3.44) 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

3.58
resource dictionary
associates resource names, used in *content streams* (3.12), with the resource *objects* (3.44) themselves and is organised into various categories (e.g., Font, ColorSpace, Pattern)

3.59
running text
body of text, as distinct from headings, footnotes, diagrams, callouts and other non-textual material

3.60
SHA
Secure Hash Algorithms, and means of reference to FIPS 180-4

3.61
signature handler
software module that implements a specific form of signing and/or augmentation and/or verification of digital signatures

3.62
sRGB
means of reference to IEC 61966-2-1 ed1.0 (1999-10) with Amendment 1 IEC 61966-2-1-am1 ed1.0 (2003-01)

3.63
stream object
dictionary followed by zero or more *bytes* (3.7) bracketed between the keywords **stream** and **endstream**

3.64
string object
series of *bytes* (3.7) (unsigned integer values in the range 0 to 255) in which the bytes are not *integer objects* (3.34), but are stored in a more compact form

3.65
TIFF
means of reference to Adobe TIFF Revision 6.0

3.66
UCS
means of reference to ISO/IEC 10646

3.67
U3D

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

means of reference to ECMA-363, Universal 3D File Format, 3rd Edition (U3D)

3.68
white-space character
characters (3.10) that separate PDF syntactic constructs such as names and numbers from each other; white-space characters are HORIZONTAL TABULATION (09h), LINE FEED (0Ah), FORM FEED (0Ch), CARRIAGE RETURN (0Dh), SPACE (20h), NULL (00h); (see "Table 1 — White-space characters" in 7.2.3, "Character set")

3.69
XFDF
means of reference to ISO 19444-1:2019, *Document management – XML Forms Data Format – Part 1: Use of ISO 32000-2 (XFDF 3.0)*

3.70
XMP
means of reference to ISO 16684-1

Note 1 to entry: In addition to ISO 16684-1, Adobe Systems Inc. also publish *Adobe XMP: Extensible Metadata Platform, Part 2: Additional Properties* and *Adobe XMP: Extensible Metadata Platform, Part 3: Storage in Files*.

3.71
XMP packet
structured wrapper for serialised XML metadata and means of reference to ISO 16684-1 that can be embedded in a wide variety of file formats

Note 1 to entry: In addition to ISO 16684-1, Adobe Systems Inc. also publish *Adobe XMP: Extensible Metadata Platform, Part 2: Additional Properties* and *Adobe XMP: Extensible Metadata Platform, Part 3: Storage in Files*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

4 Notation

4.1 General

PDF operators, PDF keywords, the names of keys in PDF dictionaries, and other predefined names are written in bold font; words that denote operands of PDF operators or values of dictionary keys are written in italic font. An italic font is also used to introduce key concepts or reference specific terms of importance.

NOTE 1 If a word or phrase is used as both a keyword and a value, the notation used in this document will default to italic font unless doing so would cause undue confusion.

NOTE 2 For backwards compatibility, words denoting predefined names, operands or values and other specific terms of importance use US English spelling.

Token characters used to delimit objects and describe the structure of PDF files, as defined in 7.2, "Lexical conventions", shall be identified by their INCITS 4-1986 (R2017) (ASCII 7-bit USA codes) character name written in upper case followed by a parenthetic two digit hexadecimal character value with the suffix "h".

Ellipses (...) are used within PDF examples to indicate omitted detail. Pairs of ellipses are also used to bracket comments about such omitted detail.

Characters in text streams, as defined by 7.9.2, "String object types", are identified by their Unicode character name written in uppercase in font followed by a parenthetic four digit hexadecimal character code value with the prefix "U+" as shown in the example in this clause:

EXAMPLE EN SPACE (U+2002)

4.2 Established notations

Due to the historical development of PDF and the requirement to remain backward compatible with existing PDF files, the notations used for keys and numbers in PDF files use United States English conventions, while the rest of the language in this document conforms to Oxford spelling in most cases. For example, within a PDF you will encounter the key **Alternate** whereas Oxford spelling would use the term Alternative. Likewise PDF files use the key **Color** rather than the Oxford spelling Colour. Exceptions to this are found where switching between the two spellings could cause confusion, such as with the term *artifact* and the word *artefact*. In cases such as these, this document will use the spelling of the term of art throughout the entirety of the document rather than switching between two spellings. Real numbers that contain a decimal radix separator use the PERIOD (2Eh) character and not the COMMA (2Ch) as used in Oxford spelling. Numbers do not include spaces.

Any use of the terms *UCS* or *UCS-2* throughout this document shall be inferred as referring to ISO/IEC 10646.

Any use of the term *JPEG* throughout this document shall be inferred as referring to compressed image data which is in accordance with ISO/IEC 10918 (all parts and amendments). ISO/IEC 10918 is informally known as the JPEG standard, for the Joint Photographic Experts Group, the ISO group that developed the standard.

Any use of the term *XFDF* throughout this document shall be inferred as referring to data which is in

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

accordance with ISO 19444-1:2019.

Any use of the term *RDF* throughout this document shall be inferred as referring to data which is in accordance with *RDFa Core 1.1, Third Edition*.

Any use of the term *sRGB* throughout this document shall be inferred as referring to data in accordance with IEC 61966-2-1 ed1.0 (1999-10) with Amendment 1 IEC 61966-2-1-am1 ed1.0 (2003-01).

Any use of the phrases *Group 3, Group 4,* or *CCITT facsimile* shall be inferred as referring to data in accordance with *ITU Recommendation T.4: Standardization of Group 3 facsimile terminals for document transmission* and *ITU Recommendation T.6: Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus*, International Telecommunication Union (ITU).

Any use of the phrase *TIFF* throughout this document shall be inferred as referring to data in accordance with *Adobe TIFF Revision 6.0, Final, (June 1992)*.

Any use of the term *XMP* throughout this document shall be inferred as referring to data in accordance with ISO 16684-1.

Any use of the term *Adobe-Japan1* throughout this document shall be inferred as referring to data in accordance with Adobe-Japan1-7 Character Collection for CID-Keyed Fonts.

Any use of the term *Adobe-GB1* throughout this document shall be inferred as referring to data in accordance with Adobe-GB1-5 Character Collection for CID-Keyed Fonts.

Any use of the term *Adobe-CNS1* throughout this document shall be inferred as referring to data in accordance with Adobe-CNS1-7 Character Collection for CID-Keyed Fonts .

Any use of the term *Adobe-Korea1* throughout this document shall be inferred as referring to data in accordance with Adobe-Korea1-2 Character Collection for CID-Keyed Fonts. Adobe-Korea1 was deprecated in this document (2020).

Any use of the term *Adobe-Japan2* throughout this document shall be inferred as referring to data in accordance with Adobe-Japan2-0 Character Collection for CID-Keyed Fonts. Adobe-Japan2 was deprecated in this document (2020).

Any use of the term *DSA* throughout this document shall be inferred as referring to the digital signature algorithm defined by *FIPS PUB 186-4.*

Any use of the term *PKCS #7* throughout this document shall be inferred as referring to cryptographic data in accordance with *Internet RFC 2315.*

Any use of the term *SHA* throughout this document shall be inferred as referring to the Secure Hash Algorithms defined in *FIPS 180-4*.

Any use of the term *CAdES* throughout this document shall be inferred as referring to data in accordance with *ETSI EN 319 122-1 V1.1.1* and *ETSI EN 319 122-2 V1.1.1*.

Any use of the term *PAdES* throughout this document shall be inferred as referring to data in accordance with *ETSI EN 319 142-1 V1.1.1* and *ETSI EN 319 142-2 V1.1.1*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

5 Version designations

For the convenience of the reader of this document, the PDF versions in which various features were introduced are provided informatively within the 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 in June 1993. Since then, PDF has gone through seven Adobe 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.

ISO 32000-1 defines a PDF version matching 1.7. This document defines a new version of PDF designated 2.0. This document is also suitable for interpretation of files made to conform to any of the previous Adobe PDF specifications 1.0 through 1.7 and ISO 32000-1.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

6 Conformance

6.1 General

This clause describes conformance with this document.

6.2 Conforming PDF documents

Conforming PDF files shall adhere to all requirements of this document. A conforming PDF file is not obligated to contain any feature other than those explicitly required by this document.

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

6.3 PDF processors

6.3.1 General

A PDF processor is any software, hardware or any other active agent that writes, reads, updates or otherwise processes a PDF file which conforms to this document, and does so in a manner that conforms to this document. Two specialised classifications of PDF processors are PDF writers, which create PDF documents, and PDF readers, which interpret PDF documents. The purpose of a PDF reader can be to display or print a document, or to extract data from a document. An interactive PDF processor interacts with a user while processing the PDF file and as a consequence may read or write PDF files. In this document, when it is important to distinguish what is written into a PDF file, the more specialised term PDF writer will be used, when it is important to distinguish the act of reading from the contents of a PDF file, the term PDF reader will be used and when it is important to distinguish the act of user interaction while processing a PDF file, the term interactive PDF processor will be used. Otherwise, the more general term PDF processor will be used.

Some PDF processors may also choose to support fragment identifiers (https://www.w3.org/DesignIssues/Fragment.html) which are added at the end of a URI to provide a reference to subordinate content within the target of the URI. These fragment identifiers anchor onto specific content or characteristics of the PDF document to which the URI refers, and represent either an action to be performed on the document when it is opened, or an object within the document (see Annex O, "Fragment identifiers").

6.3.2 Conformance of PDF processors

6.3.2.1 General

If a PDF writer creates a PDF file, the file created shall conform to this document. If a PDF writer adds or amends objects in a pre-existing PDF file then the added or amended elements shall conform to the file format requirements of this document, and shall be consistent with related existing elements.

With the exception of linearized PDF files, all PDF files should be read using the trailer and cross-reference table as described in 7.5, "File structure". Linearized files should be read as specified in Annex F, "Linearized PDF" and Annex G, "Linearized PDF access strategies". Reading a non-linearized file in a serial manner is not reliable because of the way objects are to be processed after an

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

incremental update. A PDF processor shall respect and honour standard security (see 7.6.4, "Standard security handler") when it opens a document for processing.

Each PDF processor chooses which subsets of PDF functionality to support. For each subset that the processor chooses to support, the processor shall comply with the applicable provisions in this document.

NOTE The subset standards for PDF, such as PDF/X (ISO 15930), PDF/A (ISO 19005), PDF/E (ISO 24517), PDF/VT (ISO 16612-2) and PDF/UA (ISO 14289), define the term "conforming reader" because they restrict the processing of documents conforming to those standards to strict processing rules. PDF 2.0 is highly generalized to accommodate various processing needs, so the notion of a “conforming reader” is not useful for this document.

6.3.2.2 PDF processors providing rendering

When a PDF processor renders a PDF page, it shall render the page contents as defined in this document. When doing so, the PDF processor shall respect the default (or any other user-specified) optional content definitions (8.11, "Optional content"), if any. A PDF processor shall also render the appropriate appearance stream for all annotations (12.5.5, "Appearance streams") which have appearance streams designated for this purpose as indicated by the annotation flags (see 12.5.3, "Annotation flags"), unless otherwise instructed.

6.3.2.3 Interactive PDF processors

An interactive PDF processor shall follow all the requirements in 6.3.2.2, "PDF processors providing rendering" for rendering. In addition, such a processor shall support all of the interactive aspects of optional content (8.11, "Optional content").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

7 Syntax

7.1 General

This clause covers everything about the syntax of a PDF file 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 file syntax is best understood by considering it as four parts, as shown in "Figure 1 — PDF file components":

- *Objects*. A PDF file is a data structure composed from a small set of basic types of data objects. Subclause 7.2, "Lexical conventions" describes the character set used to write objects and other syntactic elements. Subclause 7.3, "Objects" describes the syntax and essential properties of the objects. Subclause 7.3.8, "Stream objects" provides complete details of the most complex data type, the stream object. Subclause 7.4, "Filters" introduces stream filters primarily used to make PDF files smaller by applying compression technology to stream data.
- *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. Subclause 7.5, "File structure" describes the file structure. Subclause 7.6, "Encryption" describes a file-level mechanism for protecting a document's content from unauthorised access.
- *Document structure*. The PDF file structure specifies how the basic object types are used to represent components of a PDF file: pages, fonts, annotations, and so forth. Subclause 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. Subclause 7.8, "Content streams and resources" discusses PDF content streams and their associated resources.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

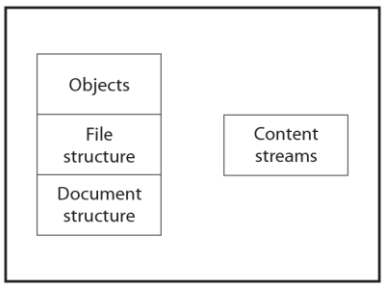


Figure 1 — PDF file components

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". Subclause 7.12, "Extensions dictionary" describes a dictionary used to identify developer-defined extensions to this document that are used in the file.

© ISO 2020 – All rights reserved

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 12.7.8, "Forms data format", and the Portable Job Ticket Format (PJTF), described in Adobe Technical Note #5620, *Portable Job Ticket Format*.

Annex H, "Example PDF file" presents several examples showing the structure of PDF files.

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 subclauses 7.2.2**,** "Representation**"** through 7.2.4, "Comments". One or more tokens are assembled to form higher-level syntactic entities, principally objects, which are the basic data values from which a PDF file is constructed.

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, UTF-8 or UTF-16 character set as described in 7.9.2, "String object types" and 7.3.8, "Stream objects" respectively.

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 as described in clause 9, "Text".

7.2.2 Representation

A non-encrypted PDF file can be entirely represented using byte values corresponding to the visible printable subset of the ASCII character set defined in INCITS 4-1986 (R2017), 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 ensure 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 can have when it is treated as data in specific contexts, such as representing human-readable text or selecting a glyph from a font.

7.2.3 Character set

The PDF character set is divided into three classes referred to as regular, delimiter, and white-space

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

characters. This classification enables the grouping of characters into tokens including separating syntactic constructs such as names and numbers from each other. The rules defined in this subclause apply to all characters in the file except within strings, streams, and comments.

NOTE 1 See 7.3.4, "String objects", 7.3.8, "Stream objects" and 7.2.4, "Comments".

PDF syntax treats any sequence of consecutive white-space characters, not inside of a string or stream, as one character. The characters that are considered white-space characters are shown in "Table 1 — White-space characters" and can be used to separate syntactic constructs such as names and numbers from each other. In all other contexts, PDF syntax 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 — that is, preceding a token that shall appear at the beginning of a line.

NOTE 2 The examples in this document 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), SOLIDUS (2Fh) and PERCENT SIGN (25h), respectively). They delimit syntactic entities such as arrays, names, and comments. The delimiter characters { and } (LEFT CURLY BRACE (7Bh) and RIGHT CURLY BRACE (7Dh)) are additional delimiter characters within Type 4 PostScript calculator functions (see 7.10.5 "Type 4 (PostScript calculator) functions"). In addition, the double character constructs << (3C3Ch) and >> (3E3Eh) are used to delimit dictionary objects (see 7.3.7, "Dictionary objects"). Any of these delimiters 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" shows the delimiter characters.

© ISO 2020 – All rights reserved

Table 2 — Delimiter characters

Glyph	Decimal	Hexadecimal	Octal	Name
(	40	28	050	LEFT PARENTHESIS
)	41	29	051	RIGHT PARENTHESIS
<	60	3C	074	LESS-THAN SIGN
>	62	3E	076	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	057	SOLIDUS
%	37	25	045	PERCENT SIGN

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 syntax is case-sensitive; corresponding uppercase and lowercase letters are distinct.

7.2.4 Comments

Any occurrence of the PERCENT SIGN (25h) outside a string or inside a content stream (see 7.8.2, "Content streams") introduces a comment. The comment consists of all characters after the PERCENT SIGN and up to but not including the end-of-the-line marker. A PDF processor should ignore comments. PDF processors shall treat comments as single white-space characters for the purposes of lexical conversion (see 7.2, "Lexical conventions"). That is, a comment separates the token preceding it from the one following it.

EXAMPLE The PDF syntax 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 defined by this document. PDF writers do not need to preserve comments.

NOTE 1 Using comments to communicate information to PDF processors is discouraged. The preferred and documented mechanism is to use PDF names (see Annex E, "Extending PDF").

NOTE 2 This clause was clarified in this document (2020).

© ISO 2020 – All rights reserved

7.3 Objects

7.3.1 General

PDF syntax includes nine basic types of objects: boolean values, integers, 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**. See 4, "Notation", for more information regarding the notation of these keywords throughout this document.

7.3.3 Numeric objects

PDF syntax provides two types of basic 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 PDF processor is running; Annex C, "Advice on maximising portability", 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. A real number shall not be present when an integer is expected. 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.

EXAMPLE 2 Real objects

```
34.5  -3.62 +123.6   4. -.002 0
```

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

Throughout this document, the term *number* refers to an object whose type can be either integer or real.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

7.3.4 String objects

7.3.4.1 General

A string object shall consist of a series of zero or more bytes. The maximum length of a string used in the computer on which the PDF processor is running may be subject to implementation limits; see Annex C, "Advice on maximising portability".

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 subclause 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.

Subclause 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 (LEFT PARENHESIS (28h) and RIGHT PARENTHESIS (29h)). Any characters may appear in a string except unbalanced parentheses and the backslash (REVERSE SOLIDUS (5Ch)), which shall be treated specially as described in this subclause. Balanced pairs of parentheses within a string require no special treatment.

EXAMPLE 1 The following are valid literal strings:

```
(This is a string)
(Strings can contain newlines
and such.)
(Strings can 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 — Escape sequences in literal strings". If the character following the REVERSE SOLIDUS is not one of those shown in "Table 3 — Escape sequences in literal strings", the REVERSE SOLIDUS shall be ignored.

Table 3 — Escape sequences in literal strings

Sequence	Meaning
\n	LINE FEED (0Ah) (LF)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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

A PDF 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 PDF processor 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
(\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 PDF 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 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 (see “Table 1 — White-space characters”) shall be ignored. The empty string can be represented in a hexadecimal string by a LESS-THAN SIGN (3Ch) immediately followed by a GREATER-THAN SIGN (3Eh).

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 that, after all escaping is expanded (see below), and the resulting sequences of bytes are not an exact binary match denote different objects. 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

name.

Even though a name may contain any characters except null, in order to express a name following the general rules for parsing PDF files, when written into a PDF file names should be encoded as described in the succeeding paragraphs.

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 can be coded in either of two ways.

No token delimiter (such as white-space) occurs between the SOLIDUS and the encoded name. White-space used as part of a name shall always be coded using the 2-digit hexadecimal notation.

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 — Examples of literal names" 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Syntax for Literal name	Resulting Name
/A#42	AB

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

NOTE 3 This document 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 document) as /Type and /FullScreen.

The maximum length of a name used in the computer on which the PDF processor is running may be subject to an implementation limit; see Annex C, "Advice on maximising portability". 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 PDF processor. 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 109 — Entries in a Type 1 font dictionary"), a colourant 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 making up the name object 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 syntax 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 can represent the same logical text. PDF name objects are considered distinct objects if, after all escaping is expanded, the resulting sequences of bytes are not an exact binary match.

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 indirect reference (see 7.3.10, "Indirect objects") shall be used to refer to a stream object, and may also be used to refer to any other kind of object. 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 syntax directly supports only one-dimensional arrays. Arrays of higher dimension can be constructed by using arrays as elements of arrays, nested to any depth.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 the PostScript language, which may be objects of any type). All keys shall be direct objects. The value may be any kind of object, including another dictionary. An indirect reference (see 7.3.10, "Indirect objects") shall be used as the value to refer to a stream object, and may also be used to refer to any other kind of object. 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 the PostScript language, where null behaves like any other object as the value of a dictionary entry.) The maximum number of entries in a dictionary used in the computer on which the PDF processor is running may be subject to implementation limits; see Annex C, "Advice on maximising portability". A dictionary may have zero entries.

NOTE 1 The statement requiring dictionary keys to always be direct objects was added in this document (2020).

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 2 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 file. 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 specialised 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 document or a registered name; see Annex E, "Extending PDF", 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 subclause 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 represents.

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

```
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**.

NOTE 2 Without the restriction against following the keyword **stream** by a CARRIAGE RETURN alone, differentiating 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 is not possible.

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 PDF processor.

"Table 5 — Entries common to all stream dictionaries" 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 needs to 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.

Streams shall also not contain 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, or an array of zero, one or several names, of filter(s) that shall be applied in processing the stream data found between the keywords **stream** and **endstream**. Multiple filters shall be specified in the order in which they are to be applied. NOTE It is not recommended to include the same filter more than once in a **Filter** array.
DecodeParms	dictionary or array	(*Optional*) A parameter dictionary or an array of such dictionaries, used by the filters specified by **Filter**, respectively. 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 in the same order as the **Filter** array: 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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
FDecodeParms	dictionary or array	(*Optional; PDF 1.2*) A parameter dictionary, or an array of such dictionaries, used by the filters specified by **FFilter**, respectively. The same rules apply as for **DecodeParms**.
DL	integer	(*Optional; PDF 1.5*) A non-negative integer representing the number of bytes in the decoded (defiltered) stream. This value is only a hint; 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. Non-zero generation numbers may be introduced when the file is later updated; see 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 string 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE (2020) Any object outside of an object stream (see 7.5.7, "Object streams") can consist solely of an object reference. PDF syntax thus permits chains of such objects.

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 PDF processor; 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 where 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 files 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 are 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 will need to transform the ASCII hexadecimal-encoded data in that stream in order to obtain the original binary data.

A PDF 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 PDF 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:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 files support 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 in 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 are lost during the encoding, resulting in a loss of quality when the data are decompressed. Compression in which no loss of data occurs is called lossless. Though somehow obvious it is 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 summarised in "Table 6 — Standard filters", which also indicates whether they accept any optional parameters. The following subclauses 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

FILTER name	Parameters	Description
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 (ISO/IEC 10918), reproducing image sample data that approximates the original data.
JPXDecode	no	(*PDF 1.5*) Decompresses data encoded using the wavelet-based JPEG 2000 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 Flate compression method and then encoded in ASCII base-85 representation. The optional entry, **DecodeParms** is not used in this example, resulting in each filter using its default parameter values.

```
1 0 obj
      <</Length 447
      /Filter [/ASCII85Decode /FlateDecode]
      >>
stream
GhQ\j_/=ii'LfceJ%/-bPR=PCD-LeMX!Z7pmG\.BSJGfG1MH^``^(F$rh*;h>h/i!XMIl:?"
o%Tk8W%<tN0"eRE==1.4N>=Cc@SiF+*s\3`snB>Y$n=^$I[!5nkk"\Wp1;WBoI"]#[bZ=Ou#c
QqtW?@3^VI[CftgH<HWngT*U:T12>8lm%+Q=gI^i0+\=:@fc@rHeGG!&rPQYfZ>YdpV`eM._@ Z]46KR;P?sT-tkq-
Xn4&]=EE2[h<FPBAePZJ9inNNF4R*)W-?\NH1C-GG&?=IC_>b%&++C2jo
*B+a7eU&`hee<0<F%-Y]De8<t.o"7p,rtNedLjK.J`kSl@c;BDEq+;F1Y?O)>keNVh+-(1*S=
CQ?fQ_I]jWNc*Qe2XR^R#;3:<$\m;6d>@C3egX,+W+VfZ]X4*3r5*tiejhQe:!'jq$^jfSIj%
13FL
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 562>>
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*
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        (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 (End Of Data). 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

$$(b_1 \times 256^3) + (b_2 \times 256^2) + (b_3 \times 256^1) + b_4 = (c_1 \times 85^4) + (c_2 \times 85^3) + (c_3 \times 85^2) + (c_4 \times 85^1) + c_5$$

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 232 - 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 subclause. 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 *Adobe TIFF Revision 6.0*; Final (TIFF). 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 RFC 1950*, and *Internet RFC 1951*.

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 were text.

The LZW and Flate compression methods can discover and exploit many patterns in the input data, whether the data are 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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) realise 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.

b) For maximum compression, the encoder looks for the longest such sequence.

c) Emit the code corresponding to that sequence.

d) 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 — Typical LZW encoding sequence".

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
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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

(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 — Optional parameters for LZWDecode and FlateDecode filters" 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 were 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 were 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*.
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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 are 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 neighbouring samples. If the predictor function works well, the postprediction data clusters toward 0.

PDF files support two groups of predictor functions. The first, the TIFF group, shall be the single function that is Predictor 2 as defined in the *Adobe TIFF Revision 6.0* specification.

NOTE 1 In the Adobe TIFF Revision 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, shall be the filters of the World Wide Web Consortium's Portable Network Graphics recommendation, as defined in *ISO/IEC 15948:2004*.

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

"Table 9 — PNG predictor algorithms" lists the 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 — Optional parameters for LZWDecode and FlateDecode filters"), whose value shall be one of

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

those listed in "Table 10 — Predictor values".

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 were 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 post-prediction 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

© ISO 2020 – All rights reserved

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 in accordance with *ITU Recommendation T.4: Standardization of Group 3 facsimile terminals for document transmission* and *ITU Recommendation T.6: Facsimile coding schemes and coding control functions for Group 4 facsimile apparatus,* International Telecommunication Union (ITU).

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 document but can be found in ITU Recommendations T.4 and T.6. For historical reasons, these documents are referred to 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 syntactic convention for sampled image data.
- Encoded data shall ordinarily be treated as a continuous, unbroken bit stream. The **EncodedByteAlign** parameter (described in "Table 11 – Optional parameters for the CCITTFaxDecode filter") 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 – Optional parameters for the CCITTFaxDecode filter".

"Table 11 – Optional parameters for the CCITTFaxDecode filter" 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 were encoded.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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*.
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 bits with a value of 1 shall be interpreted as black pixels and 0 bits as white pixels, the reverse of the normal PDF syntactic 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 bitonal (1 bit per pixel) image data that has been encoded according to the JBIG2 standard as defined by ISO 14492:2019 excluding colour palette coding. While the JBIG2 compression method may also be used for encoding colour palette images into a JBIG2 bit stream, use of JBIG2 for PDF image XObjects shall be limited to using only bitonal (1 bit per pixel) in a JBIG2 coded bit stream. Specifically segments of type "colour palette" (type 54) shall not be present, and any segment and file header colour extension flag (COLEXTFLAG) shall be 0.

NOTE 1 The JBIG2 standard defines a coding method for bi-level images (e.g., black and white pages). These are images consisting of a single rectangular bit plane, with each pixel taking on one of just two colours. It has been drafted by the Joint Bi-level Image Experts Group (JBIG), a committee established in 1988 that reports both to ISO/IEC JTC 1/SC29/WG1 and to ITU-T.

JBIG2, which supports both lossy and lossless compression, is useful for bitonal image data. JBIG2 explicitly defines the requirements of a compliant bitstream, and thus defines decoder behaviour. JBIG2 does not explicitly define a standard encoder, but instead is flexible enough to allow various approaches to encoder design.

In general, JBIG2 provides considerably better compression of bitonal images than the 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. Where JBIG2 is used for lossy compression based on symbol dictionaries, care should be taken to avoid mismatches leading to character substitution (see 0.2.1 "Symbol coding" in ISO/IEC 14492:2019 for more details). While best compression is achieved for images of text, the JBIG2 standard also includes algorithms for representing regions of an image that contain dithered halftone images (for example, photographs) or line art.

While the JBIG2 compression method may also be used for encoding multiple images into a single JBIG2 bit stream, use of JBIG2 for PDF image XObjects shall be limited to using only page 1 in a JBIG2 coded bit stream. This does not preclude the shared use of global segments across several JBIG2 encoded PDF image XObjects.

NOTE 2 Typically, these images are scanned pages of a multiple-page document. Since a single table of symbol bitmaps can be 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 can be specified in PDF files 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.

Both single-page and multiple-page JBIG2 bit streams can be represented in PDF files by encoding each JBIG2 page as one PDF image XObject, as follows:

- The filter shall use the embedded file organisation of JBIG2 as defined in ISO/IEC 14492:2019, Annex D.3, "Embedded organization".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- The optional 2-byte combination (marker) defined in ISO/IEC 14492:2019, D.3, "Embedded organization" shall not be present.
- JBIG2 bit streams shall be represented in sequential organisation as defined in ISO/IEC 14492:2019, D.1, "Sequential organisation".
- The JBIG2 file header, end-of-page segments, and end-of-file segment shall not be present.

NOTE 3 It is recommended that segments appear in increasing order.

- 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 shall always be 1; that is, when each such segment is written to the XObject, the value of its segment page association field shall 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 — Optional parameter for the JBIG2Decode filter" shall refer to that stream. The stream may be shared by multiple image XObjects whose JBIG2 encoded streams 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.

The following shows an image that was compressed according to the JBIG2 standard 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
00000001300001000000130000003400000042000000000
00000040000000000002062000010000001e000000340000
00420000000000000000020010000000231db51ce51ffac>
endstream
endobj

6 0 obj
       <</Length 126
         /Filter /ASCIIHexDecode
      >>
stream
00000000001000000003200000 3fffdff02fefefe000000
01000000012ae225aea9a5a538b4d9999c5c8e56ef0f872
7f2b53d4e37ef795cc5506dffac>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
endstream
endobj
```

The original JBIG2 bit stream from which the image XObject for this example was constructed is as follows:

```
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 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 is not allowed in a JBIG2 encoded image XObject in PDF files, this header is not placed in the JBIG2 stream object and is discarded.

b) The first JBIG2 segment (segment 0) is a 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 thus is placed in the **JBIG2Globals** stream.

c) The next segments are 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 are placed in the image XObject representing this image.

d) The last segments are 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 are not allowed in a JBIG2 encoded image XObject in PDF files, they are discarded.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

7.4.8 DCTDecode filter

The **DCTDecode** filter decodes grayscale or colour image data that has been encoded in the JPEG baseline format in accordance with ISO/IEC 10918 (all parts). Refer also to Adobe Technical Note #5116, *Supporting the DCT Filters in PostScript Level 2*, 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, 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 – Optional parameter for the DCTDecode filter".

The details of the encoding algorithm are not presented here but are in the ISO JPEG standard. 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 quantised. Each coefficient can be quantised with a different step size. It is this quantization that results in the loss of information in the JPEG algorithm. The quantised coefficients are then compressed.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 *YCbCr* before encoding and from *YCbCr* to *RGB* after decoding. If the image has four components, *CMYK* values shall be transformed to *YCbCrK* before encoding and from *YCbCrK* 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 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 (APP14) 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. Parameters that control the decoding process as well as other metadata are embedded within the encoded data stream using a notation referred to as "markers". When Adobe Systems Incorporated defined the use of JPEG images within PostScript language data streams, Adobe defined a particular set of rules pertaining to which markers are to be recognised, which are to be ignored and which are considered errors. A specific Adobe-defined marker (X'FFEE, sometimes called 'APPE' or 'APP14') was also introduced. The exact rules for producing and consuming DCT encoded data within PostScript language are provided in *Adobe Technical Note #5116* and PDF DCT encoding shall exactly follow all those rules established by Adobe for the PostScript language.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

NOTE 2 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 3 Better compression is often available for colour spaces that treat luminance and chrominance separately than for those that do not. The RGB-to-YCbCr 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, can 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

© ISO 2020 – All rights reserved

dictionary; the distinction between baseline and progressive JPEG shall be represented in the encoded data.

NOTE 4 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 JPEG 2000 compression method, an International Standard for the compression and packaging of image data. See ISO/IEC *15444-1, Information technology — JPEG 2000 image coding system: Core coding system* and ISO/IEC *15444-2, Information Technology — JPEG 2000 image coding system: Extensions*, available from http://www.jpeg.org/jpeg2000/.

NOTE 1 JPEG 2000 defines a wavelet-based method for image compression that gives somewhat better size reduction than other methods such as regular JPEG or CCITT. Also, 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, however bits per sample shall be between 1 to 38 inclusive.

NOTE 2 From a single JPEG 2000 data stream, multiple versions of an image can 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 can 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 can gain performance benefits by using the resolution progression. If the full-resolution image is densely sampled, an application can 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 can 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 can simply have performance consequences.

The JPEG 2000 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.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

of JPX but not JPX baseline, shall be supported in a PDF file. JPX file structures used in PDF files shall conform to the JPEG 2000 specification.

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.

JPX opacity and premultiplied opacity channels are 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 available method shows a normal blending mode.

NOTE 7 As specified below, if JPX opacity is used in PDF files, then the JPX data are only allowed to have one opacity channel, and this single channel is required to be associated with all colour channels.

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. For JPX files used in PDF files, the chroma-key transparency shall be ignored.

NOTE 8 For images in PDF files a similar mechanism exists, which allows to specify a range of colours to be considered transparent, see 8.9.6.4, "Colour key masking".

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 syntax.

There are other features in the JPX format beyond describing a simple image; they are not supported in PDF files. These include provisions for describing layering and giving instructions on composition and specifying simple animation. 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

entries):

- **Width** and **Height** shall match the corresponding width and height values in the JPEG 2000 data.
- **ColorSpace** shall be optional since JPEG 2000 data contain colour space specifications. If present, it shall determine how the image samples are interpreted, and the colour space specifications in the JPEG 2000 data shall be ignored. The number of ordinary colour channels in the JPEG 2000 data shall match the number of components in the colour space; a PDF 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 syntactic 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 JPEG 2000 data shall be used. A JPEG 2000 image within a PDF file shall have one of: the baseline JPX colour spaces excluding enumerated colour space 19 (CIEJab); or enumerated colour space 12 (CMYK); or at least one ICC profile that is valid within PDF files. PDF processors 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.

NOTE 9 The above bullet point was clarified in this document (2020).

- If multiple colour space specifications are given in the JPEG 2000 data, a PDF processor 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 ordinary channels in the JPEG 2000 data is 1, 3, or 4.
- **SMaskInData** specifies whether soft-mask information packaged with the image samples shall be used (see 11.6.5.2, "Soft-mask images"); if it is, the **SMask** entry shall not be present. If **SMaskInData** is non-zero, there shall be only one opacity channel in the JPEG 2000 data and it shall apply to all colour channels.
- If **ColorSpace** is absent, then the **Decode** array shall be ignored unless **ImageMask** is *true*.
- If **ImageMask** is *true*, the JPEG 2000 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 — Optional parameters for Crypt filters") shall specify which of the named crypt filters in the document (see 7.6.6, "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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 — Entries common to all encryption dictionaries") or one of the standard crypt filters (see "Table 26 — Standard crypt filter names"). 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 — Entries common to all crypt filter dictionaries") when decrypting data or providing a key to decrypt data.

NOTE When adding private data to the decode parameters dictionary, security handlers are required to name these entries as defined by Annex E, "Extending PDF".

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 7.6.3, "General encryption algorithm" to the key prior to decrypting the stream. Instead, the security handler shall decrypt the stream using the key as is. 7.4, "Filters" explains how a stream specifies filters.

7.5 File structure

7.5.1 General

This subclause describes how objects are organised in a PDF file for efficient random access and incremental update. A basic conforming PDF file shall be constructed of the following four elements (see "Figure 2 — Initial structure of a PDF file"):

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

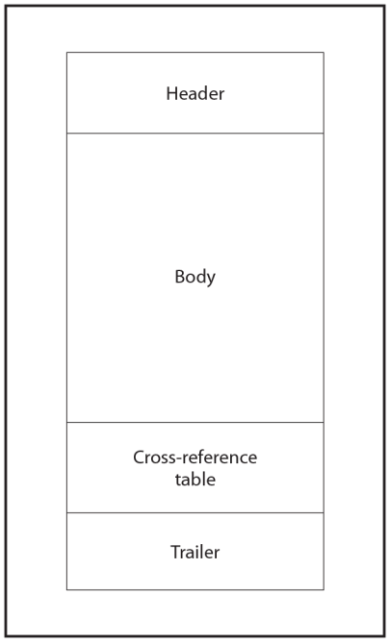


Figure 2 — Initial structure of a PDF file

As a matter of convention, the tokens in a PDF file are arranged into lines; see 7.2, "Lexical conventions". PDF files with binary data may have arbitrarily long lines.

The rules described here are sufficient to produce a basic conforming PDF file. However, additional rules apply to organising a PDF file to enable efficient incremental access to a document's components in a network environment. This form of organisation, called Linearized PDF, is described in Annex F, "Linearized PDF" and Annex G, " Linearized PDF access strategies". With the exception of linearized PDF files, all PDF files should be read using the trailer and cross-reference table as described in the following subclauses. Reading a non-linearized file in a serial manner is not reliable because of the way objects are to be processed after an incremental update. (See 6.3.2, "Conformance of PDF processors".)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

7.5.2 File header

The PDF file begins with the 5 characters "%PDF–" and byte offsets shall be calculated from the PERCENT SIGN (25h).

NOTE 1 This provision allows for arbitrary bytes preceding the %PDF- without impacting the viability of the PDF file and its byte offsets.

A PDF 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
```


© ISO 2020 – All rights reserved

```
%PDF-1.7

%PDF-2.0
```

The file header shall consist of “%PDF–1.*n*” or “%PDF–2.*n*” followed by a single EOL marker, where ‘*n*’ is a single digit number between 0 (30h) and 9 (39h).

NOTE 2 The above paragraph was added in this document (2020).

A PDF processor that writes a file that conforms to this document shall identify the version (either in the header or as the value of the **Version** entry in the document’s catalog dictionary (see 7.7.2, "Document catalog dictionary") as 2.0.

NOTE 3 The **Version** entry in the document’s catalog dictionary is used to update the version using an incremental update (see 7.5.6, "Incremental updates").

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

This document 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 PDF file so that the entire PDF 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 PDF 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, "Linearized PDF" and Annex G, "Linearized PDF access strategies"). One additional section shall be added each time the file is incrementally

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF file that has never been incrementally updated, the cross-reference section shall contain only one subsection, whose object numbering begins at 0. PDF comments shall not be included in a cross-reference table between the keywords **xref** and **trailer**.

NOTE 2 (2020) This document clarified the above paragraph in relation to comments not being permitted in cross-reference tables, as this is also stated in 7.5.8.4, "Compatibility with applications that do not support compressed reference streams".

NOTE 3 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, modified or deleted. This also means that cross reference subsections of incremental updates can never have an object number of zero.

Each cross-reference subsection shall contain entries for a contiguous range of object numbers. Each cross-reference subsection shall contain entries for a contiguous range of object numbers. The subsection shall begin with a line containing only two integers separated by a SPACE (20h) and terminated by an end-of-line marker (see 7.2.3, "Character set"). The two integers denote (respectively) the object number of the first object in this subsection and the number of entries in the subsection.

NOTE 4 The number of entries in a subsection for a file that has never been incrementally updated can never be zero.

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:

nnnnnnnnn ggggg **n** *eol*

where:

nnnnnnnnn 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 PDF 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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; this entry 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 PDF file, even if one or more of the object numbers in this range do not actually occur in the PDF 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF processor to quickly find the cross-reference table and certain special objects. PDF processors 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 PDF 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
   <<key1 value1
      key2 value2
      …
      keyn valuen
   >>
startxref
Byte_offset_of_last_cross-reference_section
%%EOF
```

"Table 15 — Entries in the file trailer dictionary" lists the contents of the trailer dictionary. The PDF file trailer dictionary may also contain any second-class name as described in Annex E, "Extending PDF".

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 PDF 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 PDF 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 PDF reader.
Prev	integer	(*Optional, present only if the file has more than one cross-reference section; shall be a direct object*) The byte offset from the beginning of the PDF file to the beginning of the previous cross-reference stream.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Root	dictionary	(*Required; shall be an indirect reference*) The catalog dictionary for the PDF file (see 7.7.2, "Document catalog dictionary").
Encrypt	dictionary	(*Required if document is encrypted; PDF 1.1*) The PDF file's encryption dictionary (see 7.6, "Encryption").
Info	dictionary	(*Optional; shall be an indirect reference*) The PDF file's information dictionary. As described in 14.3.3, "Document information dictionary", this method for specifying document metadata has been deprecated in PDF 2.0 and should therefore only be used to encode information that is stated as required elsewhere in this document. NOTE 1 The **ModDate** key within the **Info** dictionary is required if Page-Piece dictionaries (see 14.5, "Page-piece dictionaries") are used.
ID	array	(*Required in PDF 2.0 and later, or if an **Encrypt** entry is present; optional otherwise; PDF 1.1*) An array of two byte-strings constituting a PDF file identifier (See 14.4, "File identifiers") for the PDF file. Each PDF file identifier byte-string shall have a minimum length of 16 bytes. If there is an **Encrypt** entry, this array and the two byte-strings shall be direct objects and shall be unencrypted. NOTE 2 Because the **ID** entries are not encrypted, the **ID** key can be checked to assure that the correct PDF file is being accessed without decrypting the PDF file. The restrictions that the objects all be direct objects and not be encrypted ensure this. NOTE 3 Although this entry is optional prior to PDF 2.0, its absence can prevent the PDF file from functioning in some workflows that depend on PDF files being uniquely identified. NOTE 4 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 PDF reader: the **ID** strings need 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 — Additional entries in a hybrid-reference file's trailer dictionary" 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 PDF 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF file in this way is that small changes to a large document can be saved quickly. There are additional advantages such as when editing a document across an HTTP connection or using OLE embedding (a Microsoft Windows™ specific technology), a PDF processor cannot overwrite the contents of the original PDF file. Incremental updates are used to save changes to documents in these contexts.

NOTE 2 The resulting PDF file has the structure shown in "Figure 3 — Structure of an updated PDF file". 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 PDF 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 — Entries in the file trailer dictionary"). Each trailer shall be terminated by its own end-of-file (%%EOF) marker.

NOTE 3 As shown in "Figure 3 — Structure of an updated PDF file", a PDF file that has been updated several times contains several trailers. Because updates are appended to PDF files, a PDF file can 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 PDF 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 PDF 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 PDF reader reads the PDF 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 PDF file.

In versions of PDF 1.4 or later a PDF writer may use the **Version** entry in the document's catalog dictionary (see 7.7.2, "Document catalog dictionary") to override the version specified in the header. A PDF 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

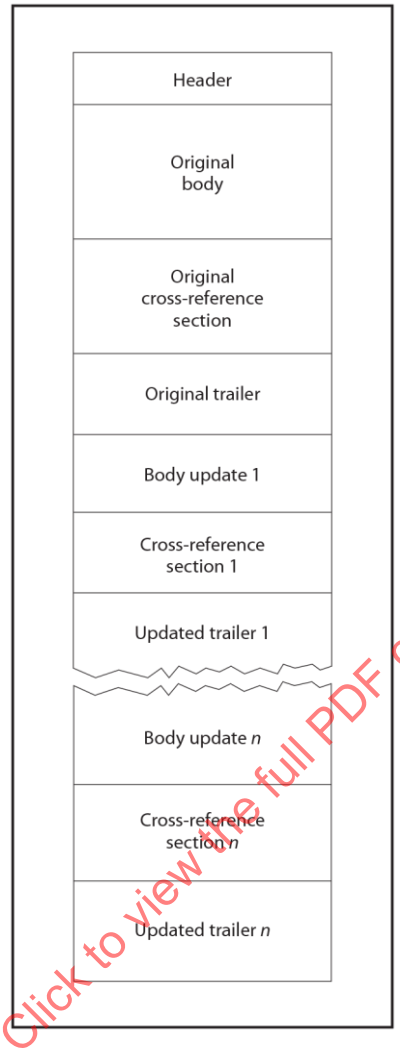


Figure 3 — Structure of an updated PDF file

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 PDF 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, "Linearized PDF" and Annex G, "Linearized PDF access strategies"), the document catalog dictionary, the linearization dictionary, and page objects shall

© ISO 2020 – All rights reserved

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 PDF reader. However, compressed objects can be stored in a manner that a PDF 1.4 PDF reader can ignore.

In addition to the regular keys for streams shown in "Table 5 — Entries common to all stream dictionaries", the stream dictionary describing an object stream contains the entries specified in “Table 16 — Additional entries specific to an object stream dictionary”.

Table 16 — Additional entries specific to an object stream dictionary

Key	Type	Value
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 is 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 PDF 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 needs to be limited. This can 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** can 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 offset for which is the value of the stream's **First** entry. The byte offsets shall be in increasing order. The pairs, themselves, shall also be separated by white-space.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- Following the **N** pairs of integers, the **N** objects are stored consecutively. Only the object values are stored in the stream; the **obj** and **endobj** keywords shall not be used.

NOTE7 (2020) Previous editions of this specification incorrectly stated that white-space was required between objects. This document corrects the text to clarify that processing of each object in an object stream starts at the specified byte offset in the decompressed stream and ends prior to the byte offset of the next object or when the end of stream is encountered.

NOTE 8 A compressed dictionary or array can 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 PDF file (i.e., entire object stream is encrypted), strings occurring anywhere in an object stream shall not be separately encrypted.

A PDF writer shall store the first object immediately after the last byte offset separated by white-space. A PDF processor 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

…
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 9 For readability, the object stream has been shown unencoded. In a PDF 2.0 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, along with a cross-reference table 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 & 2 bytes in width
respectively
         /Filter /ASCIIHexDecode      %For readability in this example
         /Size 32
         …
      >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 10 The number 54321 in Example 4 is the byte 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 (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 byte offset of the cross-reference stream rather than the **xref** keyword. For PDF files that use cross-reference streams entirely (that is, PDF 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,

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

with the exception of the **startxref** address, %%EOF segment and comments, a PDF file may be entirely a sequence of objects.

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

7.5.8.2 Cross-reference stream dictionary

Cross-reference streams shall contain the required entries and may contain the optional entries shown in "Table 17 — Additional entries specific to a cross-reference stream dictionary" in addition to the entries common to all streams ("Table 5 — Entries common to all stream dictionaries") and trailer dictionaries ("Table 15 — Entries in the file trailer dictionary"). Since some of the information in the cross-reference stream is needed by the PDF processor 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 — Additional entries specific to a cross-reference stream dictionary" 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 — Entries common to all stream dictionaries" shall also be direct objects.
- Other cross-reference stream entries not listed in "Table 17 — Additional entries specific to a cross-reference stream dictionary" may be indirect; in fact, some (such as **Root** in "Table 15 — Entries in the file trailer dictionary") 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	Value
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 shall have no more than one entry in a section. Default value: [0 *Size*].

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Prev	integer	(*Required, if any cross reference streams are already present in the file*) The byte offset from the beginning of the PDF file to the beginning of the previous cross-reference stream. The value is meaningful if the PDF file has more than one cross-reference stream. It is not meaningful in hybrid-reference files; see 7.5.8.4, "Compatibility with applications that do not support compressed reference streams". This entry has the same function as the **Prev** entry in the trailer dictionary ("Table 15 — Entries in the file trailer dictionary").
W	array	(*Required*) An array of integers representing the size of the fields in a single cross-reference entry. "Table 18 — Entries in a cross-reference stream" 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. A value of zero shall not be used for the second element of the array. 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 — Entries in a cross-reference stream"). In PDF 1.5 through PDF 2.0, 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 — Additional entries specific to a cross-reference stream dictionary"). Fields requiring more than one byte are stored with the high-order byte first.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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). Default value: *0*.
	2	The object number of the next free object.

© ISO 2020 – All rights reserved

Type	Field	Description
	3	The generation number to use if this object number is used again. Default value: *0*.
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 PDF file. Default value: *0*.
	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. NOTE This index value will be between zero and the value of **N** minus 1 from the associated object stream dictionary.

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

A hybrid-reference PDF file is readable by PDF processors designed only to support versions of PDF before PDF 1.5. Such a PDF 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 PDF files, the trailer dictionary may contain, in addition to the entry for trailers shown in "Table 15 — Entries in the file trailer dictionary", an entry, as shown in “Table 19 — Additional entries in a hybrid-reference file’s trailer dictionary”.

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 PDF file of a cross-reference stream.

When a PDF reader opens a hybrid-reference PDF file, objects with entries in cross-reference streams are not hidden. When the PDF reader searches for an object, if an entry is not found in any given

© ISO 2020 – All rights reserved
STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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).

NOTE 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 PDF reader shall look in the cross-reference stream first, find the object there, and shall ignore the free entry in the previous section.

EXAMPLE 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 PDF 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 dictionary (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 PDF reader.

One choice for a PDF 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 PDF 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
<< /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 00
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 00
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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
<< /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 PDF 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 PDF file, but otherwise, there are no ordering restrictions on any of the objects or on the main cross-reference section. However, a PDF file that uses both the hybrid-reference format and the linearized format has ordering requirements (see Annex F, "Linearized PDF" and Annex G, "Linearized PDF access strategies").

7.6 Encryption

7.6.1 General

A PDF file can be encrypted (*PDF 1.1*) to protect its contents from unauthorised access.

7.6.2 Application of encryption

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
- Any hexadecimal strings representing the value of the **Contents** key in a Signature dictionary

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

encrypted like any other stream in the file. Beginning with PDF 1.5, embedded files can be encrypted in an otherwise unencrypted document (see 7.6.6, "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 — Entries in the file trailer dictionary"). The absence of this entry from the trailer dictionary means that a PDF processor shall consider the document to be not encrypted. The entries shown in "Table 20 — Entries common to all encryption dictionaries" are common to all encryption dictionaries. For document with multiple versions the **Encrypt** entry should be identical in all copies of the trailer.

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 PDF processors shall support, but PDF processors 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 file 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 file 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 with a key length of 128 bits (16 bytes) to use on specific streams (see 7.6.6, "Crypt filters"). A value of 5 for **V** permits the specification of crypt filters with a file encryption key length of 256 bits (32 bytes). Values less than 5 for the **V** entry are deprecated in PDF 2.0.

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 7.6.4, "Standard security handler". Entries for public-key security handlers are described in 7.6.5, "Public-key security handlers".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 PDF processor 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, "Extending PDF".
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.

© ISO 2020 – All rights reserved

Key	Type	Value
V	integer	(*Required*) 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* (*Deprecated in PDF 2.0*) Indicates the use of 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) with a file encryption key length of 40 bits; see below. *2* (*PDF 1.4; deprecated in PDF 2.0*) Indicates the use of 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) but permitting file encryption key lengths greater than 40 bits. *3* (*PDF 1.4; deprecated in PDF 2.0*) An unpublished algorithm that permits file encryption key lengths ranging from 40 to 128 bits. This value shall not appear in a conforming PDF file. *4* (*PDF 1.5; deprecated in PDF 2.0*) The security handler defines the use of encryption and decryption in the document, using the rules specified by the **CF**, **StmF**, and **StrF** entries using 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) with a file encryption key length of 128 bits. *5* (*PDF 2.0*) The security handler defines the use of encryption and decryption in the document, using the rules specified by the **CF**, **StmF**, **StrF** and **EFF** entries using 7.6.3.3, "Algorithm 1.A: Encryption of data using the AES algorithms" with a file encryption key length of 256 bits.
Length	integer	(*Optional; PDF 1.4; only if V is 2 or 3; deprecated in PDF 2.0*) The length of the file 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) or 5 (PDF 2.0)*) A dictionary whose keys shall be crypt filter names and whose values shall be the corresponding crypt filter dictionaries (see "Table 25 — Entries common to all crypt filter dictionaries"). Every crypt filter used in the document shall have an entry in this dictionary, except for the standard crypt filter names (see "Table 26 — Standard crypt filter names"). Any keys in the **CF** dictionary that are listed in "Table 26 — Standard crypt filter names" shall be ignored by a PDF processor. Instead, the PDF processor shall use properties of the respective standard crypt filters.
StmF	name	(*Optional; meaningful only when the value of **V** is 4 (PDF 1.5) or 5 (PDF 2.0)*) 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 — Standard crypt filter names". 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 — Standard filters"), shall be decrypted by the security handler, using this crypt filter. Default value: *Identity*.
StrF	name	(*Optional; meaningful only when the value of V is 4 (PDF 1.5) or 5 (PDF 2.0)*) 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 — Standard crypt filter names". Default value: *Identity*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
EFF	name	(*Optional; meaningful only when the value of* ***V*** *is 4* (*PDF 1.6*) *or 5* (*PDF 2.0*)) 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 — Standard crypt filter names". This entry shall be provided by the security handler. PDF 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 values of the keys defined in "Table 20 — Entries common to all encryption dictionaries" shall not be encrypted. However, a security handler may choose to encrypt any objects that are private to itself.

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

7.6.3 General encryption algorithm

7.6.3.1 General

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. The use of this algorithm in PDF 2.0 is deprecated.

NOTE 1 The name RC4™ is a registered trademark of RSA Security Inc. and cannot be used by third parties creating implementations of the algorithm. Proprietary implementations of the RC4 encryption algorithm are available under license from RSA Security Inc. For licensing information, contact: RSA Security Inc. 2955 Campus Drive, Suite 400, San Mateo, CA 94403-2507, USA, or http://www.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).*

Strings and streams encrypted with AES shall use a padding scheme that is described in *Internet RFC 8018*. For an original message length of M, the pad shall consist of 16 - (M modulo 16) bytes whose value shall also be 16 - (M modulo 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.

NOTE 2 Since **Encrypt** versions 1 to 4 are deprecated in PDF 2.0, this has the effect of implicitly deprecating the use of MD5 for key generation purposes.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

For **Encrypt** versions 1-4, as defined by the **V** key of the encryption dictionary, PDF's standard encryption methods also make use of the MD5 message-digest algorithm for key generation purposes (described in *Internet RFC 1321)*. **Encrypt** version 5 does not use MD5.

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. Encryption of data uses one of two algorithms. When the value of the **V** key of the encryption dictionary is 1, 2, 3 or 4 (*PDF 1.7*), 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" (*deprecated in PDF 2.0*) shall be used. When the value of the **V** key of the encryption dictionary is 5 (*PDF 2.0*) 7.6.3.3, "Algorithm 1.A: Encryption of data using the AES algorithms" shall be used. The difference is that Algorithm 1.A uses the starting key directly and does not modify the file encryption key at all. Algorithm 1.A is used only with the AES algorithm and 256-bit file encryption keys.

Algorithms in 7.6, "Encryption" are uniquely numbered within this subclause in a manner that maintains compatibility with previous documentation.

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

This algorithm is deprecated in PDF 2.0.

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 file 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.) For example, for object number 258 and generation number 7, the hexadecimal values 0x02 0x01 0x00 0x07 0x00 would be appended to the file encryption key.

   If using the AES algorithm, extend the file 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) Initialise 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.

7.6.3.3 Algorithm 1.A: Encryption of data using the AES algorithms

a) Use the 32-byte file encryption key for the AES-256 symmetric key algorithm, along with the string or stream data to be encrypted.

   Use the AES algorithm in Cipher Block Chaining (CBC) mode, which requires an initialization vector. The block size parameter is set to 16 bytes, and the initialization vector is a 16-byte random

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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".

7.6.4 Standard security handler

7.6.4.1 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 PDF processor might have a security settings dialogue box that the user can invoke before saving the 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 *user* and *owner* passwords.

NOTE 1 A PDF processor can 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 PDF reader shall first try to authenticate the encrypted document using the padding string defined in 7.6.4.3, "File encryption key algorithm" (default user password):

- If this authentication attempt is successful, the PDF reader may open, decrypt, render and otherwise provide access to the document.
- If this authentication attempt fails, the interactive PDF processor should prompt for a password. Correctly supplying either password (*owner* or *user* password) should enable the user to gain access to the document.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

operations to which user access can be controlled shall be as follows:

- Modifying the document's contents
- Copying or otherwise extracting text, images and graphics from the document.
- Adding or modifying text annotations (see 12.5.6.4, "Text annotations") and interactive form fields (see 12.7, "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 document outline items 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, images and graphics (in support of accessibility, or for other purposes) to be controlled separately.

If a security handler of revision 4 or 5 is specified, the standard security handler shall support crypt filters (see 7.6.6, "Crypt filters"). The support shall be limited to the **Identity** crypt filter (see "Table 26 — Standard crypt filter names") and crypt filters named **StdCF** whose dictionaries contain an **AuthEvent** value of *DocOpen*. For revision 4, the filter **CFM** value shall be *V2* (RC4) or **AESV2** (AES-128). For revision 6, the filter **CFM** value shall be *AESV3* (AES-256). 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 an indicator of the type of the security handler or encryption. Use of security handler revisions 1, 2, 3, 4 and 5 is deprecated in PDF 2.0.

Once the document has been opened and decrypted successfully, a PDF 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. PDF 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 12.8.6, "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 can be non-interactive PDF readers that do not have a person running them such as printing off-line or on a server.

All passwords for revision 6 shall be based on Unicode. Preprocessing of a user-provided password consists first of normalizing its representation by applying the "SASLPrep" profile (*Internet RFC 4013*) of the "stringprep" algorithm (*Internet RFC 3454*) to the supplied password using the Normalize and BiDi options. Next, the password string shall be converted to UTF-8 encoding, and then truncated to the first 127 bytes if the string is longer than 127 bytes (see 7.6.4.3.3, "Algorithm 2.A: Retrieving the file encryption key from an encrypted document in order to decrypt it (revision 6 and later)", steps (a, b)).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 3 The use of the “SASLPrep” profile (*Internet RFC 4013*) and the “stringprep” algorithm (*Internet RFC 3454*) places limitations on the set of Unicode characters which can be used in PDF passwords.

7.6.4.2 Standard encryption dictionary

"Table 21 — Additional encryption dictionary entries for the standard security handler" shows the encryption dictionary entries for the standard security handler (in addition to those in "Table 20 — Entries common to all encryption dictionaries").

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

Key	Type	Value
R	integer	(*Required*) A number specifying which revision of the standard security handler shall be used to interpret this dictionary: *2* (*Deprecated in PDF 2.0*) if the document is encrypted with a **V** value less than *2* (see "Table 20 — Entries common to all encryption dictionaries") 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 — Standard security handler user access permissions". *3* (*Deprecated in PDF 2.0*) 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* (*Deprecated in PDF 2.0*) if the document is encrypted with a **V** value of *4*. *5* (*PDF 2.0; deprecated in PDF 2.0*) Shall not be used. This value was used by a deprecated proprietary Adobe extension. *6* (*PDF 2.0*) if the document is encrypted with a **V** value of *5*.
O	byte string	(*Required*) A byte string, 32 bytes long if the value of **R** is *4* or less and 48 bytes long if the value of **R** is *6*, based on both the owne r and user passwords, that shall be used in computing the file encryption key and in determining whether a valid owner password was entered. For more information, see 7.6.4.3, "File encryption key algorithm" and 7.6.4.4, "Password algorithms".
U	byte string	(*Required*) A byte string, 32 bytes long if the value of **R** is *4* or less and 48 bytes long if the value of **R** is *6*, based on the owner and 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 7.6.4.4, "Password algorithms".
OE	byte string	(*Required if R is 6 (PDF 2.0)*) A 32-byte string, based on the owner and user password, that shall be used in computing the file encryption key. For more information, see 7.6.4.4, "Password algorithms".
UE	byte string	(*Required if **R** is 6 (PDF 2.0)*) A 32-byte string, based on the user password, that shall be used in computing the file encryption key. For more information, see 7.6.4.4, "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 — Standard security handler user access permissions").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Perms	byte string	(*Required if **R** is 6 (PDF 2.0)*) A 16-byte string, encrypted with the file encryption key, that contains an encrypted copy of the permissions flags. For more information, see 7.6.4.4, "Password algorithms".
EncryptMetadata	boolean	(*Optional; meaningful only when the value of **V** is 4 (PDF 1.5) or 5 (PDF 2.0)*) Indicates whether the document-level metadata stream (see 14.3.2, "Metadata streams") shall be encrypted. 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 — Standard security handler user access permissions" 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).

PDF 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** is always 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 22 — Standard security handler user access permissions

Bit position	Meaning
1 - 2	Reserved. Must be zero (0).
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	Copy or otherwise extract text and graphics from the document. However, for the limited purpose of providing this content to assistive technology, a PDF reader should behave as if this bit was set to 1. NOTE For accessibility, ISO 32000-1 had this option restricted by bit 10, but that exception has been deprecated in PDF 2.0.
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).
7 - 8	Reserved. Must be 1.

© ISO 2020 – All rights reserved

Bit position	Meaning
9	(*Security handlers of revision 3 or greater*) Fill in existing interactive form fields (including signature fields), even if bit 6 is clear.
10	Not used. This bit was previously used to determine whether content could be extracted for the purposes of accessibility, however, that restriction has been deprecated in PDF 2.0. PDF readers shall ignore this bit and PDF writers shall always set this bit to 1 to ensure compatibility with PDF readers following earlier specifications.
11	(*Security handlers of revision 3 or greater*) Assemble the document (insert, rotate, or delete pages and create document outline items 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, based on an implementation-dependent algorithm. When this bit is clear (and bit 3 is set), printing shall be limited to a low-level representation of the appearance, possibly of degraded quality.
13 - 32	(*Security handlers of revision 3 or greater*) Reserved. Must be 1.

NOTE The above table was re-titled and corrected in this document (2020).

7.6.4.3 File encryption key algorithm

7.6.4.3.1 General

As noted earlier, one function of a security handler is to generate a file encryption key for use in encrypting and decrypting the contents of a document. Given a password string, the standard security handler computes a file encryption key. For revision 4 and earlier, the algorithm is as shown in 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)" and for revision 6, the algorithm is as shown in 7.6.4.3.3, "Algorithm 2.A: Retrieving the file encryption key from an encrypted document in order to decrypt it (revision 6 and later)".

7.6.4.3.2 Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)

This algorithm is deprecated in PDF 2.0.

a) The password string is generated from host system codepage characters (or system scripts) by first converting the string to **PDFDocEncoding**. If the input is Unicode, first convert to a codepage encoding, and then to **PDFDocEncoding** for backward compatibility. Pad or truncate the resulting 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) Initialise 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. (7.6.4.4.2, "Algorithm 3:

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Computing the encryption dictionary's O-entry value (revision 4 and earlier)" 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 — Entries in the file trailer dictionary") to the MD5 hash function.

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.

NOTE 1 This provision pertains only to document-level XMP metadata.

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 file encryption key as defined by the value of the encryption dictionary's **Length** entry.

i) Set the file 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.

NOTE 2 The first element of the **ID** array, as used in 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)", step e, generally remains unchanged across revisions of a given document. However, since this is not guaranteed, use of the **ID** in computation of the file encryption key, as required when using 7.6.4.3.3, "Algorithm 2.A: Retrieving the file encryption key from an encrypted document in order to decrypt it (revision 6 and later)Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)", can complicate updates to the document. For this reason, security handlers are encouraged to use Algorithm 2.A or higher, which do not use the **ID** in file encryption key computation. This algorithm, when applied to the user password string, produces the file encryption key used to encrypt or decrypt string and stream data according to 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms". Parts of this algorithm are also used in the algorithms described below.

7.6.4.3.3 Algorithm 2.A: Retrieving the file encryption key from an encrypted document in order to decrypt it (revision 6 and later)

To understand the algorithm below, it is necessary to treat the 48-bytes of the **O** and **U** strings in the **Encrypt** dictionary as made up of three sections, as described in Algorithms 8 and 9. The first 32 bytes are a hash value (explained below). The next 8 bytes are called the Validation Salt. The final 8 bytes are called the Key Salt. Whenever UTF-8 password is used below, steps (a) and (b) are to be applied to the relevant password string to generate the UTF-8 password.

a) The UTF-8 password string shall be generated from Unicode input by processing the input string with the SASLprep (*Internet RFC 4013*) profile of stringprep (*Internet RFC 3454*) using the Normalize and BiDi options, and then converting to a UTF-8 representation.

b) Truncate the UTF-8 representation to 127 bytes if it is longer than 127 bytes.

c) Test the password against the owner key by computing a hash using algorithm 2.B with an input string consisting of the UTF-8 password concatenated with the 8 bytes of owner Validation Salt, concatenated with the 48-byte **U** string. If the 32-byte result matches the first 32 bytes of the O string, this is the owner password.

d) Compute an intermediate owner key by computing a hash using algorithm 2.B with an input string consisting of the UTF-8 owner password concatenated with the 8 bytes of owner Key Salt, concatenated with the 48-byte **U** string. The 32-byte result is the key used to decrypt the 32-byte **OE** string using AES-

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

256 in CBC mode with no padding and an initialization vector of zero. The 32-byte result is the file encryption key.

e) Compute an intermediate user key by computing a hash using algorithm 2.B with an input string consisting of the UTF-8 user password concatenated with the 8 bytes of user Key Salt. The 32-byte result is the key used to decrypt the 32-byte **UE** string using AES-256 in CBC mode with no padding and an initialization vector of zero. The 32-byte result is the file encryption key.

f) Decrypt the 16-byte **Perms** string using AES-256 in ECB mode with an initialization vector of zero and the file encryption key as the key. Verify that bytes 9-11 of the result are the characters "a", "d", "b". Bytes 0-3 of the decrypted **Perms** entry, treated as a little-endian integer, are the user permissions. They shall match the value in the **P** key.

7.6.4.3.4 Algorithm 2.B: Computing a hash (revision 6 and later)

Take the SHA-256 hash of the original input to the algorithm and name the resulting 32 bytes, K. Perform the following steps (a)-(d) 64 times:

a) Make a new string, **K1**, consisting of 64 repetitions of the sequence: input password, **K**, the 48-byte user key. The 48 byte user key is only used when checking the owner password or creating the owner key. If checking the user password or creating the user key, **K1** is the concatenation of the input password and **K**.

b) Encrypt **K1** with the AES-128 (CBC, no padding) algorithm, using the first 16 bytes of **K** as the key and the second 16 bytes of **K** as the initialization vector. The result of this encryption is **E**.

c) Taking the first 16 bytes of **E** as an unsigned big-endian integer, compute the remainder, modulo 3. If the result is 0, the next hash used is SHA-256, if the result is 1, the next hash used is SHA-384, if the result is 2, the next hash used is SHA-512.

d) Using the hash algorithm determined in step c, take the hash of **E**. The result is a new value of **K**, which will be 32, 48, or 64 bytes in length.

Repeat the process (a-d) with this new value for **K**. Following 64 rounds (round number 0 to round number 63), do the following, starting with round number 64:

NOTE 2 The reason for multiple rounds is to defeat the possibility of running all paths in parallel. With 64 rounds (minimum) there are 3^64 paths through the algorithm.

e) Look at the very last byte of **E**. If the value of that byte (taken as an unsigned integer) is greater than the round number - 32, repeat steps (a-d) again.

f) Repeat from steps (a-e) until the value of the last byte is ≤ (round number) - 32.

NOTE 3 Tests indicate that the total number of rounds will most likely be between 65 and 80.

The first 32 bytes of the final **K** are the output of the algorithm.

7.6.4.4 Password algorithms

7.6.4.4.1 General

In addition to the file encryption key, the standard security handler shall provide the contents of the encryption dictionary ("Table 20 — Entries common to all encryption dictionaries" and "Table 21 — Additional encryption dictionary entries for the standard security handler"). The values of the **Filter**, **V**, **Length**, **R**, and **P** entries are straightforward. The computation of the values for the **O** (owner password) and the **U** (user password) entries for encryption revision 4 and earlier, and the **O**, **U**, **OE** (owner encryption), **UE** (user encryption) and **Perms** (permissions) values for encryption revision 6

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

require more explanation.

Algorithms 3 through 5 show how the values of the owner password and user password are computed for revision 4 and earlier. Algorithm 6 and 7 show how to determine if a password is valid. Algorithms 8 through 10 show how to compute the values stored in the PDF (by writers) to determine if a revision 6 password is valid. Algorithms 11 through 13 describe how to validate a password provided by a user or owner to read the PDF.

Passwords for revision 4 and earlier are up to 32 characters in length, and are limited to characters in the **PDFDocEncoding** character set (see Annex D, "Character sets and encodings").

In revision 4 and earlier, the result of running the password algorithm was the file encryption key. In revision 6, the file encryption key is decoupled from the password algorithm to make the owner and user keys independent. For algorithms 8 to 13, the file encryption key shall be a 256-bit (32-byte) value generated with a strong random number generator.

7.6.4.4.2 Algorithm 3: Computing the encryption dictionary's O-entry value (revision 4 and earlier)

This algorithm is deprecated in PDF 2.0.

a) Pad or truncate the owner password string as described in step (b) of 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)". If there is no owner password, use the user password instead.

b) Initialise 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 file 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 (b) of 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)".

f) Encrypt the result of step (e), using an RC4 encryption function with the file 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 a file 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.

7.6.4.4.3 Algorithm 4: Computing the encryption dictionary's U-entry value (Security handlers of revision 2)

This algorithm is deprecated in PDF 2.0.

a) Create a file encryption key based on the user password string, as described in 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

b) Encrypt the 32-byte padding string shown in step (b) of 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)", using an RC4 encryption function with the file encryption key from the preceding step.

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

7.6.4.4.4 Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or 4)

This algorithm is deprecated in PDF 2.0.

a) Create a file encryption key based on the user password string, as described in 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)".

b) Initialise the MD5 hash function and pass the 32-byte padding string shown in step (b) of 7.6.4.3.2, "Algorithm 2: Computing a file encryption key in order to encrypt a document (revision 4 and earlier)" 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 — Entries in the file trailer dictionary") 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 a file encryption key generated by taking each byte of the original file 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.

7.6.4.4.5 Algorithm 6: Authenticating the user password (Security handlers of revision 4 and earlier)

This algorithm is deprecated in PDF 2.0.

a) Perform all but the last step of 7.6.4.4.3, "Algorithm 4: Computing the encryption dictionary's U-entry value (Security handlers of revision 2)" or 7.6.4.4.4, "Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or 4)" 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 file encryption key obtained in step (a) (that is, in the first step of 7.6.4.4.3, "Algorithm 4: Computing the encryption dictionary's U-entry value (Security handlers of revision 2)" or 7.6.4.4.4, "Algorithm 5: Computing the encryption dictionary's U (user password) value (Security handlers of revision 3 or 4)") shall be used to decrypt the document.

7.6.4.4.6 Algorithm 7: Authenticating the owner password (Security handlers of revision 4 and earlier)

This algorithm is deprecated in PDF 2.0.

a) Compute a file encryption key from the supplied password string, as described in step (a) to step (d) of

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

"Algorithm 3: Computing the encryption dictionary's O-entry value (revision 4 and earlier)".

b) (*Security handlers of revision 2 only*) Decrypt the value of the encryption dictionary's **O** entry, using an RC4 encryption function with the file 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 7.6.4.4.5, "Algorithm 6: Authenticating the user password (Security handlers of revision 4 and earlier)". If it is correct, the password supplied is the correct owner password.

7.6.4.4.7 Algorithm 8: Computing the encryption dictionary's U (user password) and UE (user encryption) values (Security handlers of revision 6)

a) Generate 16 random bytes of data using a strong random number generator. The first 8 bytes are the User Validation Salt. The second 8 bytes are the User Key Salt. Compute the 32-byte hash using algorithm 2.B with an input string consisting of the UTF-8 password concatenated with the User Validation Salt. The 48- byte string consisting of the 32-byte hash followed by the User Validation Salt followed by the User Key Salt is stored as the **U** key.

b) Compute the 32-byte hash using algorithm 2.B with an input string consisting of the UTF-8 password concatenated with the User Key Salt. Using this hash as the key, encrypt the file encryption key using AES-256 in CBC mode with no padding and an initialization vector of zero. The resulting 32-byte string is stored as the **UE** key.

7.6.4.4.8 Algorithm 9: Computing the encryption dictionary's O (owner password) and OE (owner encryption) values (Security handlers of revision 6)

a) Generate 16 random bytes of data using a strong random number generator. The first 8 bytes are the Owner Validation Salt. The second 8 bytes are the Owner Key Salt. Compute the 32-byte hash using algorithm 2.B with an input string consisting of the UTF-8 password concatenated with the Owner Validation Salt and then concatenated with the 48-byte U string as generated in Algorithm 8. The 48-byte string consisting of the 32-byte hash followed by the Owner Validation Salt followed by the Owner Key Salt is stored as the **O** key.

b) Compute the 32-byte hash using 7.6.4.3.4, "Algorithm 2.B: Computing a hash (revision 6 and later)" with an input string consisting of the UTF-8 password concatenated with the Owner Key Salt and then concatenated with the 48-byte U string as generated in 7.6.4.4.7, "Algorithm 8: Computing the encryption dictionary's U (user password) and UE (user encryption) values (Security handlers of revision 6)". Using this hash as the key, encrypt the file encryption key using AES-256 in CBC mode with no padding and an initialization vector of zero. The resulting 32-byte string is stored as the **OE** key.

7.6.4.4.9 Algorithm 10: Computing the encryption dictionary's Perms (permissions) value (Security handlers of revision 6)

Fill a 16-byte block as follows:

a) Extend the permissions (contents of the P integer) to 64 bits by setting the upper 32 bits to all 1's.

NOTE This allows for future extension without changing the format.

b) Record the 8 bytes of permission in the bytes 0-7 of the block, low order byte first.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

c) Set byte 8 to the ASCII character "T" or "F" according to the **EncryptMetadata** boolean.

d) Set bytes 9-11 to the ASCII characters '"a", "d", "b".

e) Set bytes 12-15 to 4 bytes of random data, which will be ignored.

f) Encrypt the 16-byte block using AES-256 in ECB mode with an initialization vector of zero, using the file encryption key as the key. The result (16 bytes) is stored as the **Perms** string, and checked for validity when the file is opened.

7.6.4.4.10 Algorithm 11: Authenticating the user password (Security handlers of revision 6)

a) Test the password against the user key by computing the 32-byte hash using 7.6.4.3.4, "Algorithm 2.B: Computing a hash (revision 6 and later)" with an input string consisting of the UTF-8 password concatenated with the 8 bytes of User Validation Salt (see 7.6.4.4.7, "Algorithm 8: Computing the encryption dictionary's U (user password) and UE (user encryption) values (Security handlers of revision 6)"). If the 32- byte result matches the first 32 bytes of the **U** string, this is the user password.

7.6.4.4.11 Algorithm 12: Authenticating the owner password (Security handlers of revision 6)

a) Test the password against the owner key by computing the 32-byte hash using algorithm 2.B with an input string consisting of the UTF-8 password concatenated with the 8 bytes of Owner Validation Salt and the 48 byte **U** string. If the 32 byte result matches the first 32 bytes of the **O** string, this is the owner password.

7.6.4.4.12 Algorithm 13: Validating the permissions (Security handlers of revision 6)

a) Decrypt the 16 byte **Perms** string using AES-256 in ECB mode with an initialization vector of zero and the file encryption key as the key. Verify that bytes 9-11 of the result are the characters "a", "d", "b". Bytes 0-3 of the decrypted **Perms** entry, treated as a little-endian integer, are the user permissions. They should match the value in the **P** key. Byte 8 should match the ASCII character "T" or "F" according to the boolean value of the **EncryptMetadata** key.

7.6.5 Public-key security handlers

7.6.5.1 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 — Public-key security handler user access permissions" in 7.6.5.2, "Public-key encryption dictionary".

Public-key security handlers use the industry standard *Public Key Cryptographic Standard Number 7* (CMS) binary encoding syntax to encode recipient list, decryption key, and access permission information. The CMS specification is in *Internet RFC 5652*.

When encrypting the data, each recipient's X.509 public key certificate (as described in ITU-T Recommendation X.509) shall be available. When decrypting the data, the PDF 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

shall be used to decrypt the encrypted data.

7.6.5.2 Public-key encryption dictionary

Encryption dictionaries for public-key security handlers contain the common entries shown in "Table 20 — Entries common to all encryption dictionaries", whose values are described above. In addition, they may contain the entry shown in "Table 23 — Additional encryption dictionary entries for public-key security handlers" as described below.

The **Filter** entry shall be the name of a public-key security handler. This filter entry shall be the name of the public key handler used to encrypt the document. See "Table 20 — Entries common to all encryption dictionaries".

NOTE Examples of existing security handlers that support public-key encryption are Entrust.PPKEF, Adobe.PPKLite, and Adobe.PubSec.

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 7.6.6, "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” defines additional encryption dictionary entries for public-key security handlers.

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 CMS object listing recipients who have been granted equal access rights to the document. The data contained in the CMS object shall include both a cryptographic key that shall be used to decrypt the encrypted data and the access permissions (see "Table 24 — Public-key security handler user access permissions") that apply to the recipient list. There shall be only one CMS 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 — Additional crypt filter dictionary entries for public-key security handlers".

NOTE (2020) "Table 23 — Additional encryption dictionary entries for public-key security handlers" previously specified a **P** key which was removed in this document due to errors that prohibited interoperable implementations.

The access permissions 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 — Public-key security handler user access permissions" 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.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. However, for the limited purpose of providing this content to assistive technology, a PDF reader shall behave as if this bit was set to 1. NOTE ISO 32000-1 had this option restricted by bit 10, for accessibility, but that exception has been deprecated in PDF 2.0.
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	Not used. This bit was previously used to determine whether content could be extracted for the purposes of accessibility, however, that restriction has been deprecated in PDF 2.0. PDF readers shall ignore this bit and PDF writers shall always set this bit to 1 to ensure compatibility with PDF readers supporting older specifications.
11	Assemble the document (insert, rotate, or delete pages and create document outline items 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, based on an implementation-dependent algorithm. When this bit is clear (and bit 3 is set), printing shall be limited to a low-level representation of the appearance, possibly of degraded quality.

NOTE "Table 24 — Public-key security handler user access permissions" is different to "Table 22 — Standard security handler user access permissions".

7.6.5.3 Public-key encryption algorithms

"Figure 4 — Public-key encryption algorithm" illustrates how CMS objects shall be used when encrypting PDF files. A CMS object is designed to encapsulate and encrypt what is referred to as the enveloped data.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

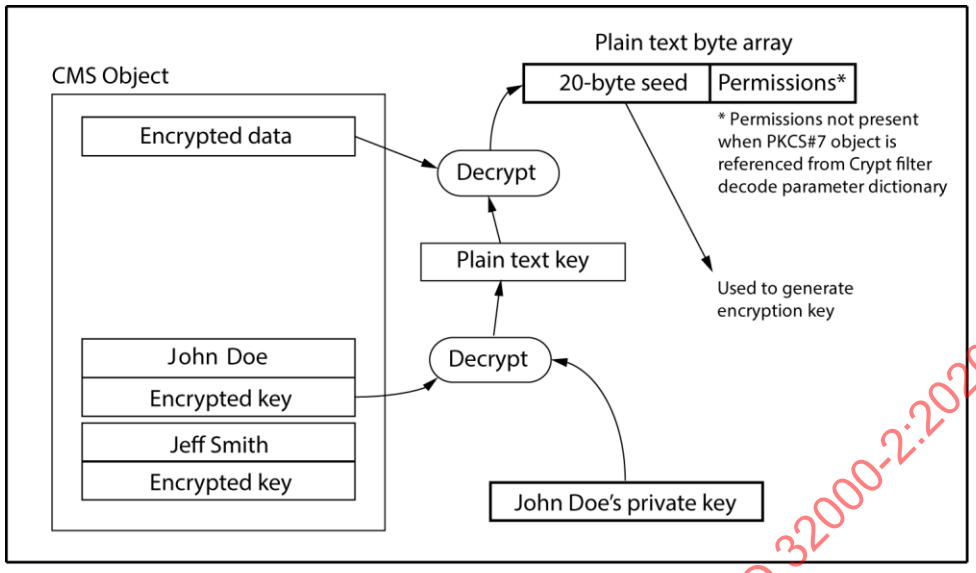


Figure 4 — Public-key encryption algorithm

The enveloped data in the CMS 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 7.6.6, "Crypt filters"). A key shall be used to encrypt (and decrypt) the enveloped data. This key (the plaintext key in "Figure 4 — Public-key encryption algorithm") shall be encrypted for each recipient, using that recipient's public key, and shall be stored in the CMS 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 CMS object, resulting in a byte array that includes the following information:

- A 20-byte seed that shall be used to create the file encryption key that is used by "Algorithm 1: Encryption of data using the RC4 or AES algorithms" or "Algorithm 1.A: Encryption of data using the 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, most significant byte first. See "Table 24 — Public-key security handler user access permissions" for the possible permission values.

NOTE The above bullet was corrected in this document (2020).

- 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 — Optional parameters for Crypt filters"), 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 CMS object are:

- RC4 with key lengths up to 256-bits (deprecated in PDF 2.0);
- DES, Triple DES, RC2 with key lengths up to 128 bits (deprecated in PDF 2.0);
- 128-bit AES in Cipher Block Chaining (CBC) mode (deprecated in PDF 2.0);
- 192-bit AES in CBC mode (deprecated in PDF 2.0);

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 256-bit AES in CBC mode.

The CMS specification is in *Internet RFC 5652, Cryptographic Message Syntax*.

The file encryption key used by 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms" shall be calculated by means of an SHA-1 message digest operation, for a key length of 128 bits. For the file encryption key used by 7.6.3.3 "Algorithm 1.A: Encryption of data using the AES algorithms", a SHA-256 digest operation shall be used for a key length of 256 bits. These operations digest the following data, in order:

a) The 20 bytes of seed.

b) The bytes of each item in the **Recipients** array of CMS 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.

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

7.6.6 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 — Entries common to all encryption dictionaries") 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 — Entries common to all crypt filter dictionaries".
- 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 PDF processor shall provide a standard Identity filter which shall pass the data unchanged (see "Table 26 — Standard crypt filter names") 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 — Optional parameters for Crypt filters") whose **Name** entry specifies the particular crypt filter that shall 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. PDF 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 PDF 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* (*Deprecated in PDF 2.0*) The application shall ask the security handler for the file encryption key and shall implicitly decrypt data with 7.6.3.2, "Algorithm 1: Encryption of data using the RC4 or AES algorithms", using the RC4 algorithm. *AESV2* (*PDF 1.6; deprecated in PDF 2.0*) The application shall ask the security handler for the file encryption key and shall implicitly decrypt data with 7.6.3.2, "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. The key size (**Length**) shall be 128 bits. *AESV3* (*PDF 2.0*) The application shall ask the security handler for the file encryption key and shall implicitly decrypt data with 7.6.3.3, "Algorithm 1.A: Encryption of data using the AES algorithms", using the AES-256 algorithm in Cipher Block Chaining (CBC) with padding mode with a 16-byte block size and an initialization vector that is randomly generated and placed as the first 16 bytes in the stream or string. The key size (**Length**) shall be 256 bits. When the value is *V2*, *AESV2* or *AESV3*, the application may ask once for this file 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 file 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 that shall be used to trigger the authorization that is required to access file 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 — Entries common to all encryption dictionaries"), the PDF reader shall ignore this key and behave as if the value is *DocOpen*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Length	integer	*(Required; deprecated in PDF 2.0)* Security handlers may define their own use of the **Length** entry and should use it to define the bit length of the file encryption key. The bit length of the file encryption key shall be a multiple of 8 in the range of 40 to 256. The standard security handler expresses the **Length** entry in bytes (e.g., 32 means a length of 256 bits) and public-key security handlers express it as is (e.g., 256 means a length of 256 bits). When **CFM** is *AESV2*, the **Length** key shall have the value of 128. When **CFM** is *AESV3*, the **Length** key shall have a value of 256. NOTE (2020) The **Length** key was corrected to be required and the descriptive text updated.

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, "Extending PDF").

Table 26 — Standard crypt filter names

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

"Table 27 — Additional crypt filter dictionary entries for public-key security handlers" lists the additional crypt filter dictionary entries used by public-key security handlers (see 7.6.5, "Public-key security handlers"). When these entries are present, the value of **CFM** shall be *V2* or *AESV2* or *AESV3*.

NOTE The allowed values of **CFM** were corrected in this document (2020).

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

Key	Type	Value
Recipients	string or array	(*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 CMS object that shall list recipients that have been granted equal access rights to the document. The enveloped data contained in the CMS object shall include both a 20-byte seed value that shall be used to compute the file encryption key (see 7.6.5.3, "Public-key encryption algorithms") followed by 4 bytes of permissions settings (see "Table 24 — Public-key security handler user access permissions") 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. NOTE (2020) The cross-reference to Table 24 in the above paragraph was corrected. If the crypt filter is referenced from a **Crypt** filter decode parameter dictionary (see "Table 14 — Optional parameters for Crypt filters"), this entry shall be a string that shall be a binary-encoded CMS object that shall contain a list of all recipients who are permitted to access the corresponding encrypted stream. The enveloped data contained in the CMS

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
		object shall be a 20-byte seed value that shall be used to create the file 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. PDF processors 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 PDF processor. 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-2.0
                                        %… at least four binary characters > 127
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                                 %1st 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
   … 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
   /ID [<1379F4B0AA2C1B40B3F1A63E4B8C7810> <4C9F4FF0E656CF40AD95CDEA3E720E1B> ]
      %ID is not encrypted
>>
startxref
495
%%EOF
```

7.6.7 Unencrypted wrapper document

This document allows PDF writers to use custom security handlers that encrypt the strings and streams within a PDF file using algorithms that are not specified in this standard. This allows PDF producers the flexibility to adapt to changes in encryption technology and respond to customer needs. However, when PDF files are encrypted using such non-standard security handlers, users attempting to open such documents may not have enough information to identify a security handler for the user's PDF processor that would enable the document to be decrypted.

To enable PDF producers using non-standard security handlers to provide users with information to help them identify and install necessary security handler(s) the producer may embed the encrypted PDF file within an unencrypted PDF file (an *unencrypted wrapper*). An unencrypted wrapper should provide guidance informing users of the security handler that is needed to decrypt the embedded encrypted PDF file (*encrypted payload*). This subclause specifies use of the collection dictionary and associated files to identify the encrypted payload in a way that allows PDF processors that already have the necessary security handler to immediately present the encrypted payload, although PDF processors without the custom security handler will present the unencrypted wrapper document with helpful instructions to the user.

To enable automatic display of an encrypted payload within an unencrypted wrapper the PDF producer shall include a **Collection** dictionary (as described in 12.3.5, "Collections") identifying the encrypted payload as the initial document in the collection and setting the collection **View** to *H* (hidden). In addition, the PDF producer shall include the file specification dictionary for the encrypted

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

payload in the **EmbeddedFiles** name tree, and as an entry in the **AF** array in the document catalog. The file specification dictionary for the encrypted payload shall include the **AFRelationship** key with a value of *EncryptedPayload*, and shall include an encrypted payload dictionary (see "Table 28 — Entries in an encrypted payload dictionary") with details of the cryptographic filter needed to decrypt the encrypted payload. For a PDF file that is an unencrypted wrapper for an encrypted payload document, the **EmbeddedFiles** name tree shall contain exactly one entry, for the encrypted payload document.

Table 28 — Entries in an encrypted payload dictionary

Key	Type	Value
Type	name	*(Optional)* The type of PDF object that this dictionary describes; if present, shall be *EncryptedPayload* for an encrypted payload file specification.
Subtype	name	*(Required)* The name of the cryptographic filter used to encrypt the encrypted payload document. This allows a PDF processor to easily determine whether it has the appropriate cryptographic filter.
Version	name	*(Optional)* The version number of the cryptographic filter used to encrypt the encrypted payload referenced by this dictionary. NOTE The value of **Version** is not to be interpreted as a real number but as integers with a PERIOD (2Eh) between them.

EXAMPLE This example shows details of an unencrypted wrapper document containing an encrypted payload document.

```
%PDF-2.0
15 0 obj
      <</Length 377 /Filter /FlateDecode>>
stream
   …
endstream
endobj

17 0 obj                                %File Specification Dictionary
      <<
         /Desc (This embedded file is encrypted using the Acme Custom Crypto filter)
         /EF <</F 18 0 R>>
         /F (AcmeCustomCrypto Protected PDF.pdf)
         /Type /Filespec
         /UF (AcmeCustomCrypto Protected PDF.pdf)
         /EP
         <<
            /Type /EncryptedPayload
            /Subtype /AcmeCustomCrypto
            /Version /1.0
         >>
         /AFRelationship /EncryptedPayload
      >>
endobj

18 0 obj                                %Embedded File stream
      <<
         /Type /EmbeddedFile
         /DL 123456
         /Filter /FlateDecode
         /Length 123456
         /Params
         <<
            /CheckSum <01234567890123456789012345678901>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            /CreationDate (D:20130228170141-08'00)
            /ModDate (D:20130228170141-08'00)
            /Size 123456
         >>
         /Subtype /application#2Fpdf
      >>
stream
… Flate-encoded contents of encrypted payload document …
endstream
endobj

19 0 obj                                %Document Catalog
      <<
         /Type/Catalog
         /Collection
         <<
            /D (AcmeCustomCrypto Protected PDF.pdf)
            /View /H
         >>
         /Names 20 0 R
         /Pages 12 0 R
         …
         /AF [17 0 R]
      >>
endobj

20 0 obj                                %Document Name tree
      <</EmbeddedFiles 21 0 R>>
endobj

21 0 obj                                %Embedded Files Name dictionary
      <</Names [(AcmeCustomCrypto Protected PDF.pdf) 17 0 R]>>
endobj

12 0 obj %Page tree
      … Page tree for wrapper cover page …
endobj

…
xref
0 21
0000000000 65536 f
…
0000036212 00000 n
trailer
      <<
         /Root 19 0 R
         /Info 3 0 R
         /Size 21
      >>
startxref
37873
%%EOF
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 dictionary").

© ISO 2020 – All rights reserved

NOTE Most of the objects in the hierarchy are dictionaries. "Figure 5 — Structure of a PDF document" 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 dictionary. 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 subclause, 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 subclauses.

7.7.2 Document catalog dictionary

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 dictionary 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 29 — Entries in the catalog dictionary" shows the entries in the catalog dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

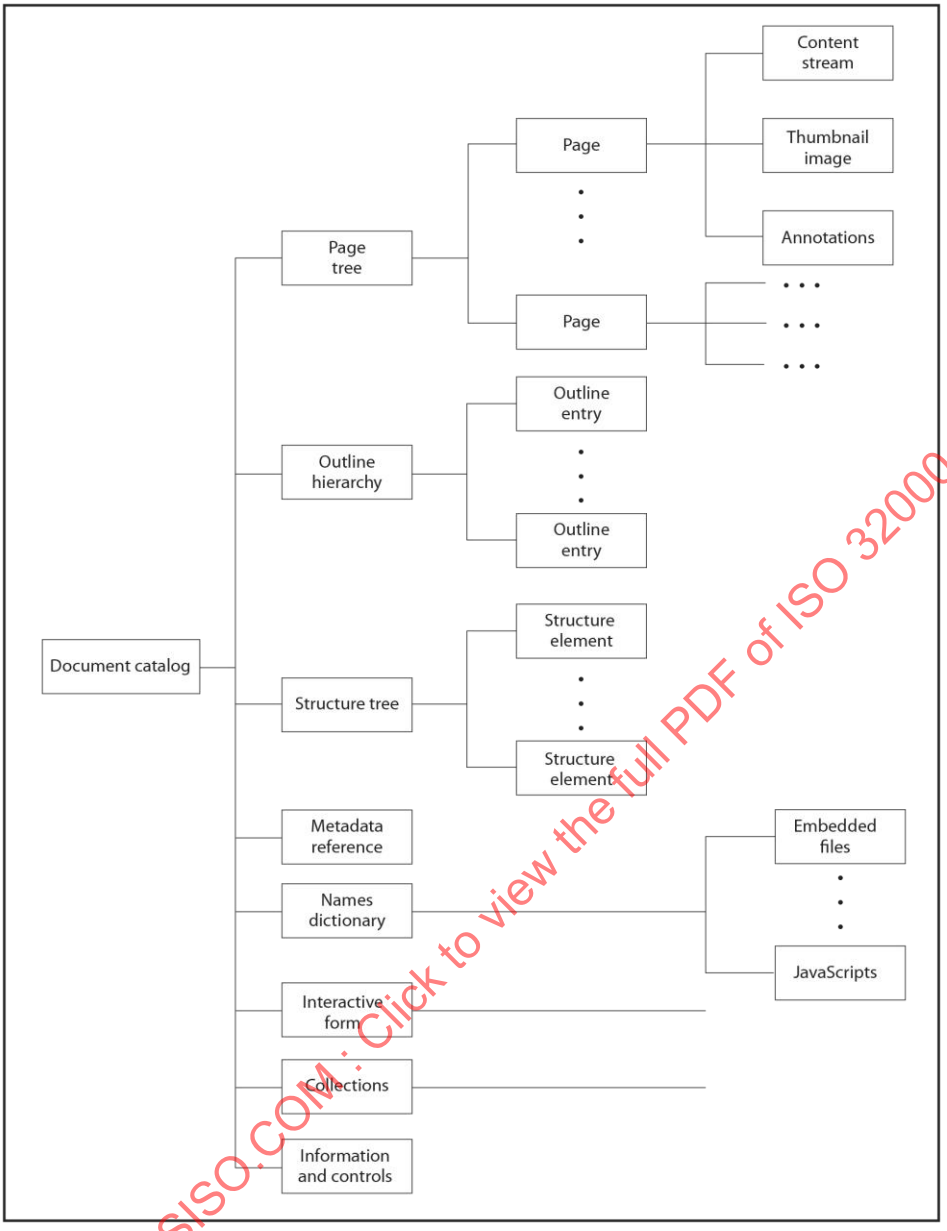


Figure 5 — Structure of a PDF document

Table 29 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 PDF processor 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 32000-1*) 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"). *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **Names** dictionary is required and shall contain the **EmbeddedFiles** name tree.
Dests	dictionary	(*Optional; PDF 1.1; shall be an indirect reference*) A dictionary of names and corresponding destinations (see 12.3.2.4, "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, PDF readers shall use their own current user preference settings.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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*.
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	dictionary or array	(*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 (12.6.2, "Action dictionaries"). 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 1.2*) 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 12.6.4.8, "URI actions").
AcroForm	dictionary	(*Optional; PDF 1.2*) The document's interactive form (AcroForm) dictionary (see 12.7.3, "Interactive form dictionary").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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").
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. 12.8.6, "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 12.8.7, "Legal content attestations").
Requirements	array	(*Optional; PDF 1.7*) An array of requirement dictionaries that shall represent requirements for the document. Subclause 12.11, "Document requirements", describes this dictionary and how it shall be used.
Collection	dictionary	(*Optional; PDF 1.7*) A collection dictionary that an interactive PDF processor shall use to enhance the presentation of file attachments stored in the PDF document. (see 12.3.5, "Collections"). (PDF 2.0) For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document ") the **Collection** is required and shall identify the encrypted payload as the default document (as indicated by the **D** entry) of the collection and shall further specify that the collection **View** should be initially H (hidden).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
NeedsRendering	boolean	(*Optional; deprecated in PDF 2.0*) 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 Annex K, "XFA forms". Default value: *false*.
DSS	dictionary	(*Optional; PDF 2.0*) A DSS dictionary containing document-wide security information. See 12.8.4.3, "Document Security Store (DSS)".
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this PDF document. See 14.13, "Associated files" and 14.13.3, "Associated files linked to the PDF document's catalog" for more details. For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **AF** key is required and shall include a reference to the file specification dictionary for the encrypted payload document.
DPartRoot	dictionary	(*Optional; PDF 2.0*) A DPartRoot dictionary used to describe the document parts hierarchy for this PDF document. See 14.12, "Document parts".

EXAMPLE The following shows a sample catalog dictionary 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, PDF 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 subclauses. Compliant PDF processors 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 optimise application performance, a PDF 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.

7.7.3.2 Page tree nodes

"Table 30 — Required entries in a page tree node" shows the entries in a page tree node that shall always be present (*Required*).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 30 — 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; not permitted 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 Since the number of pages descendent from a Pages dictionary can be accurately determined by examining the tree itself using the **Kids** arrays, the **Count** entry is redundant. A PDF writer shall ensure that the value of the **Count** key is consistent with the number of entries in the **Kids** array and its descendants which definitively determines the number of descendant pages.

NOTE 1 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".

PDF processors 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

In addition to the entries shown in "Table 30 — Required entries in a page tree node", 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"). A page tree shall not contain multiple indirect references to the same page tree node.

NOTE 2 This requirement prevents multiple copies of the same node being present in the tree.

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 31 — Entries in a page object" 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 31 — 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 or *Template* for an invisible Template page (see 12.7.7, "Named pages").
Parent	dictionary	(*Required; shall be an indirect reference*) The page tree node that is the immediate parent of this page object. Objects of Type *Template* shall have no **Parent** key.
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 contents (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, but PDF writers should not use this method of sharing resources as described in 7.8.3, "Resource dictionaries".
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 (see 14.11.2, "Page boundaries"). Default value: the value of **MediaBox**. NOTE 1 This clipped page output will often be placed (imposed) on the output medium in some implementation-defined manner.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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.
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 with at least one white-space character added between the streams' data, in order, to form a single stream. PDF 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 organisation. Applications that consume or produce PDF files need not preserve the existing structure of the Contents array. PDF 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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. Objects of Type *Template* shall have no **B** key. NOTE 2 The information in this entry can be created or recreated from the information obtained from the **Threads** key in the catalog dictionary.
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").
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 14.7.5.4, "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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Tabs	name	(*Optional; PDF 1.5*) A name specifying the tab order that shall be used for annotations on the page (see 12.5 "Annotations"). If present, the values shall be one of *R* (row order), *C* (column order), and *S* (structure order). Beginning with PDF 2.0, additional values also include *A* (annotations array order) and *W* (widget order). Annotations array order refers to the order of the annotation enumerated in the **Annots** entry of the Page dictionary (see "Table 31 — Entries in a page object"). Widget order means using the same array ordering but making two passes, the first only picking the widget annotations and the second picking all other annotations.
TemplateInstantiated	name	(*Required if this page was created from a named page object; PDF 1.5*) The name of the originating page object (see 12.7.7, "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	array	(*Optional; PDF 1.6*) An array of viewport dictionaries (see "Table 265 — Entries in a viewport dictionary") that shall specify rectangular regions of the page.
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this page. See 14.13, "Associated files" and 14.13.8, "Associated files linked to DParts" for more details.
OutputIntents	array	(*Optional; PDF 2.0*) An array of output intent dictionaries that shall specify the colour characteristics of output devices on which this page might be rendered (see 14.11.5, "Output intents").
DPart	dictionary	(*Required, if this page is within the range of a DPart, not permitted otherwise; PDF 2.0*) An indirect reference to the DPart dictionary whose range of pages includes this page object (see 14.12.3, "Connecting the DPart tree structure to pages"). NOTE 3 The **DPart** key in a page object allows a PDF processor to directly retrieve the section of the document part hierarchy that applies to this page object. This also allows for ready access of DPM data in PDF processors.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

EXAMPLE The following shows the definition of a page object with 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
                             >>
                     >>
         /Contents 12 0 R
         /Annots [23 0 R
                  24 0 R
                 ]
      >>
endobj
```

A page tree shall not contain multiple indirect references to the same page object.

NOTE This requirement prevents multiple copies of the same page being present in the tree.

7.7.3.4 Inheritance of page attributes

Some of the page attributes shown in "Table 31 — Entries in a page object" 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 might 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 can override this inherited value with a **MediaBox** entry of its own.

All values shall be inherited as-is, without merging, even for composite data types such as arrays and dictionaries. Thus (at least conceptually) the **Resources** dictionary for a page shall be found by searching the Page object and then each Pages object encountered by following Parent links up the pages tree from the Page towards the Catalog object. When the first **Resources** dictionary is found the search shall be stopped and that **Resources** dictionary shall be used in its entirety.

In a document conforming to Linearized PDF (see Annex F, "Linearized PDF" and Annex G, "Linearized PDF access strategies") 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 — Inheritance of attributes" 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

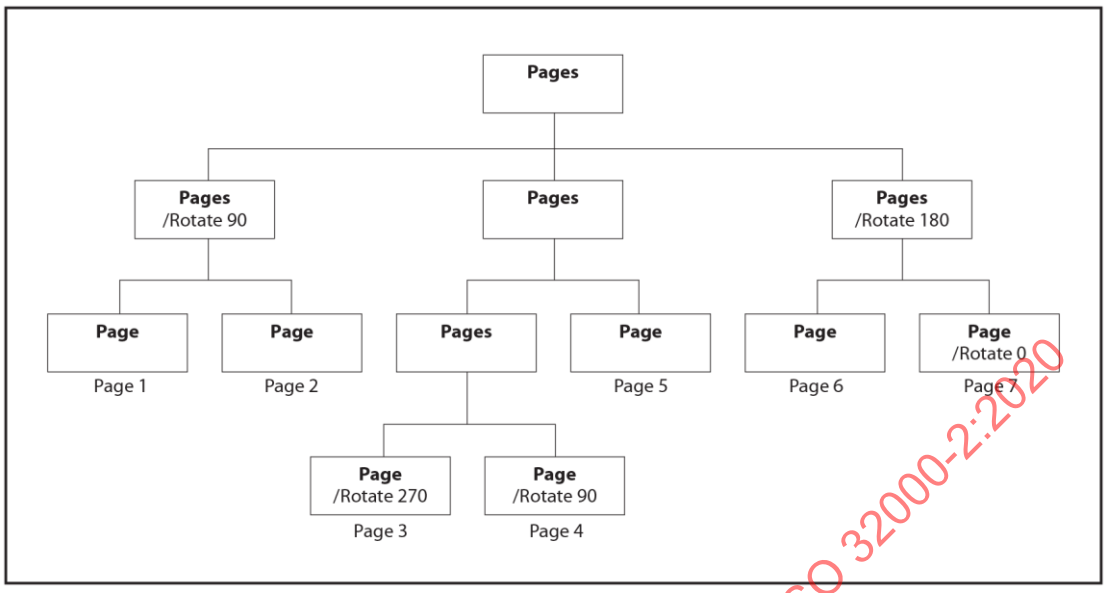


Figure 6 — Inheritance of attributes

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 dictionary (see 7.7.2, "Document catalog dictionary"). 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 32 — Entries in the name dictionary" shows the contents of the name dictionary.

Table 32 — Entries in the name dictionary

Key	Type	Value
Dests	name tree	(*Optional; PDF 1.2*) A name tree mapping name strings to destinations (see 12.3.2.4, "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").
JavaScript	name tree	(*Optional; PDF 1.3*) A name tree mapping name strings to document-level ECMAScript actions (see 12.6.4.17, "ECMAScript actions").
Pages	name tree	(*Optional; PDF 1.3*) A name tree mapping name strings to visible pages for use in interactive forms (see 12.7.7, "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 12.7.7, "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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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"). All File Specification dictionaries referenced from this name tree shall contain an **EF** key whose value is a dictionary which contains either an **F** or **UF** key whose value is an embedded file stream. *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the name strings provided in this tree shall not contain or be derived from the encrypted payload document's actual file name. This is to avoid potential disclosure of sensitive information in the original filename. The name string should match the value of **F** or **UF** in the referenced File Specification dictionary.
AlternatePresentations	name tree	(*Optional; PDF 1.4; deprecated in PDF 2.0*) 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 subclause 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 file. However, whereas the file 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 (9.6.4, "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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 Annex A, "Operator Summary" provides a summary of all PDF operators.

NOTE 2 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 the PostScript language 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 3 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 PDF reader encounters an operator in a content stream that it does not recognise, an error shall occur. A pair of compatibility operators, **BX** and **EX** (*PDF 1.1*), shall modify this behaviour (see "Table 33 — Compatibility operators"). These operators shall occur in pairs and may be nested. They bracket a compatibility section, a portion of a content stream within which unrecognised operators shall be ignored without error. This mechanism enables a PDF processor 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 33 — Compatibility operators

Operands	Operator	Description
—	**BX**	(*PDF 1.1*) Begin a compatibility section. Unrecognised 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 unrecognised operands and operators from previous matching **BX** onward.

7.8.3 Resource dictionaries

As stated in 7.8.2, "Content streams" 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** entry or is inherited, as described under 7.7.3.4, "Inheritance of page attributes" from some ancestor node of the page object. PDF writers should not use this inheritance feature of PDF as its use can cause undue complexity for a PDF reader. A PDF writer should only include resource definitions for resources that are actually referenced by the content streams of the associated page in the **Resources** dictionary. If the content streams of multiple pages require exactly the same set of resources, a single **Resources** dictionary may be shared between them by using indirect references. If each page requires different sets of resources, then each should be written with its own **Resources** dictionary.
- Content streams that define the glyph descriptions of a Type 3 font shall include a **Resources** entry in the Type 3 font dictionary specifying all the resources used by all the content streams in the **CharProcs** dictionary of a Type 3 font.
- For other content streams, a PDF writer shall include a **Resources** entry in the stream's dictionary specifying the resource dictionary which contains all the resources used by that

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

content stream. This shall apply to content streams that define form XObjects, patterns, and annotation appearances.

- 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.

NOTE The above bulleted list was clarified in this document (2020).

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 34 — Entries in a resource dictionary".

The corresponding values shall be as follows:

- For all 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 34 — 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 9, "Text").
ProcSet	array	(*Optional; deprecated in PDF 2.0*) An array of predefined procedure set names (see 14.2, "Procedure sets").
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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

EXAMPLE 2 The following shows a resource dictionary containing fonts, and external objects. 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.

```
<</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 used throughout PDF technology. This subclause 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 35 — PDF data types" summarises the basic and higher-level data types that are used throughout this document to describe the values of dictionary entries and other PDF data values.

Table 35 — PDF data types

Type	Description	Subclause
ASCII string	Bytes containing ASCII characters	7.9.2
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 7.9.2.4
date	Date (ASCII string)	7.9.4
dictionary	Dictionary object	7.3.7
file specification	File specification (string or dictionary)	7.11
function	Function (dictionary or stream)	7.10

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Type	Description	Subclause
integer	Integer number	7.3.3
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)	7.3.3
number tree	Number tree (dictionary)	7.9.7
PDFDocEncoded string	Bytes containing a string that shall be encoded using PDFDocEncoding	7.9.2 7.9.2.3
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: ASCII string and byte string.	7.9.2
text string	Bytes that represent characters that shall be encoded using either PDFDocEncoding, UTF-16BE or UTF-8 (as defined in 7.9.2.2, "Text string type".)	7.9.2 7.9.2.2
text stream	Text stream	7.9.3

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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, ASCII string, or byte string. The further qualification reflects the encoding used to represent the characters or glyphs described by the string.

The string types described in "Table 35 — PDF data types" specify increasingly specific encoding schemes, as shown in "Figure 7 — Relationship between string types".

© ISO 2020 – All rights reserved

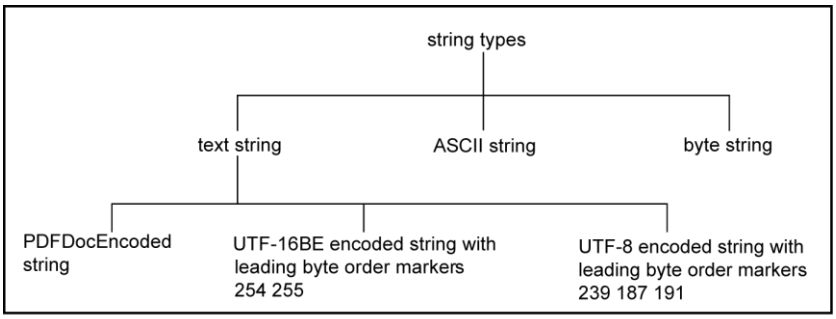


Figure 7 — Relationship between string types

7.9.2.2 Text string type

7.9.2.2.1 General

The text string type shall be used for character strings that contain information intended to be human-readable, such as text annotations, document outline item names, article names, 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 **PDFDocEncoding,** the UTF-16BE Unicode character encoding scheme, or (PDF 2.0) the UTF-8 Unicode character encoding scheme. **PDFDocEncoding** can encode all of the ISO Latin 1 character set and is documented in Annex D, "Character sets and encodings". UTF-16BE and UTF-8 can encode all Unicode characters. UTF-16BE, UTF-8 and Unicode character encoding are described in *The Unicode Standard* by the Unicode Consortium.

EXAMPLE 1 A PDF dictionary containing key 'Key' with the value that is the text string "text‰" will look like

```
<</Key(text?)>>
```

where the character '?' after the 'text' is represented by the hex code 8B (octal code 213 - that is according to "D.2 Latin character set and encodings").

NOTE 2 **PDFDocEncoding** does not support all Unicode characters whereas UTF-16BE and UTF-8 do.

For text strings encoded in UTF-16BE, the first two bytes shall be 254 followed by 255. These two bytes represent the Unicode byte order marker, ZERO WIDTH NO-BREAK SPACE (U+FEFF), indicating that the string is encoded in the UTF-16BE (big-endian) encoding scheme specified in Unicode.

EXAMPLE 2 A PDF dictionary containing key 'Key' with the value that is the text string "тест" (that is what the word in Russian with the translation to English as 'test') will look like

```
<</Key(??????????)>>
```

where the characters in parentheses is the sequence of bytes with hex codes FE, FF, 04, 42, 04, 35, 04, 41, 04, 42.

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.

For text strings encoded in UTF-8, the first three bytes shall be 239 followed by 187, followed by 191. These three bytes represent the Unicode byte order marker indicating that the string is encoded in the UTF-8 encoding scheme specified in Unicode.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

NOTE 4 This mechanism precludes beginning a string using **PDFDocEncoding** with the three characters dieresis, guillemotright, questiondown, which is unlikely to be a meaningful beginning of a word or phrase.

PDF 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.

NOTE 5 It is important not to confuse UTF-16BE with UCS2 (i.e. wchar_t). UTF-16 is not a fixed width encoding scheme.

7.9.2.2.2 Text string language escape sequences

Escape sequences may appear anywhere in a Unicode text string to indicate the language in which subsequent text shall be written.

NOTE 1 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 ESCAPE (U+001B) (that is, for strings encoded in UTF-16BE, the byte sequence 0 followed by 27; for strings encoded in UTF-8 the byte value 27).

b) A 2- byte BCP 47 language code.

EXAMPLE 1 en for English or ja for Japanese encoded as ASCII characters.

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

EXAMPLE 2 US for the United States or JP for Japan encoded as ASCII characters.

d) The Unicode value ESCAPE (U+001B).

NOTE 2 The complete list of codes defined by BCP 47 and ISO 3166 can be obtained from the Internet Engineering Task Force and the International Organization for Standardization.

NOTE 3 Since 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.

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 or UTF-8 do.

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. For example, byte strings are used to define a file identifier (see 14.4, "File identifiers") that is specified in ID entry of PDF file trailer (see "Table 15 — Entries in the file trailer dictionary"). In such case byte string is written in hexadecimal form (see 7.3.4.3, "Hexadecimal strings") and looks like

<B6FB54F3F8554D478DC874F11DAD0F11>

NOTE The string can represent characters but the encoding is not known. The bytes of the string do not have to represent characters.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 8824-1. A date shall be a text string value containing no white-space, 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 now and 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
```

NOTE 1 A date string can be any valid PDF string object as described in 7.3.4, "String objects". The description above relates to the text string value after appropriate processing.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

NOTE 2 PDF versions up to and including 1.7 defined a date string to include a terminating apostrophe. PDF processors are recommended to accept date strings that still follow that convention.

NOTE 3 The letter Z can optionally be followed by hour and minute offsets, which are zero in this case.

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.

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).

NOTE Rectangles can have a width of zero or height of zero.

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 (null objects, 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 — Entries in a name tree node dictionary" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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	(*Required for intermediate and leaf nodes; not permitted in root nodes*) 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            …
            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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 — Entries in a number tree node dictionary" 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".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 parameterised classes of functions, including mathematical formulas and sampled representations with arbitrary resolution.

NOTE 1 Functions can 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 valid values for the input. Some types of functions also define a range, the set of valid 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause 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 — Entries common to all function dictionaries".

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 × m numbers, where m shall be the number of input values. For each i from 0 to m - 1, **Domain**$_{2i}$ shall be less than or equal to **Domain**$_{2i+1}$, and the i^{th} input value, x_i, shall lie in the interval **Domain**$_{2i} \le x_i \le$ **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 × *n* numbers, where *n* shall be the number of output values. For each *j* from 0 to *n* - 1, **Range**$_{2j}$ shall be less than or equal to **Range**$_{2j+1}$, and the j^{th} output value, y_j, shall lie in the interval **Range**$_{2j} \le y_j \le$ **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 (subject to implementation limits).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.

$$y = \text{Interpolate}(x, x_{\min}, x_{\max}, y_{\min}, y_{\max})$$
$$= y_{\min} + \left((x - x_{\min}) \times \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \right)$$

For a given value of *x*, Interpolate calculates the *y* value on the line defined by the two points ($x_{\min}$, $y_{\min}$) and ($x_{\max}$, $y_{\max}$).

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 organised 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 — Entries common to all function dictionaries", a Type 0 function dictionary includes those shown in "Table 39 — Additional entries specific to a Type 0 function dictionary".

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.

The interpolate function (7.10.1, "General") explains the relationship between **Domain**, **Encode**, **Size**, **Decode**, and **Range**.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 — Entries common to all stream dictionaries").

When a sampled function is called, each input value x_i, for $0 \le i < m$, 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 $0 \le j < n$, 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 ÷ 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 — Mapping with the Decode array"). 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 ÷ 14) and the sample values supplied are in the interval [0 14], the

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

effective range of [-1.0 1.0] is achieved, and the range value 0.0 is represented by the sample value 7.

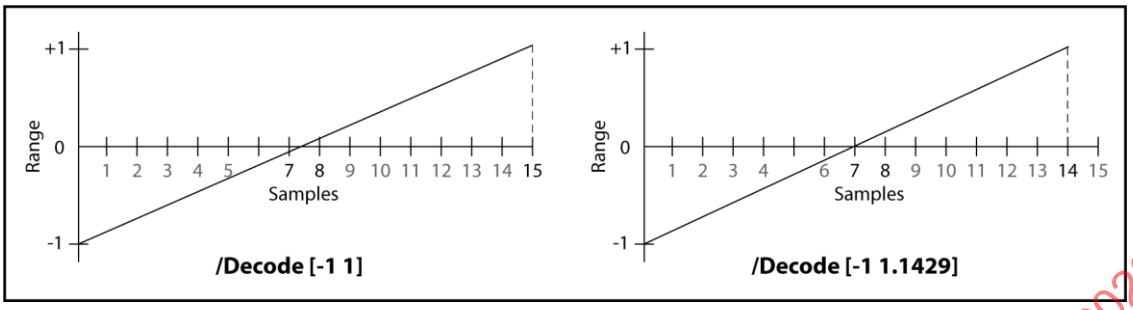


Figure 8 — Mapping with the Decode array

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.

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 — Entries common to all function dictionaries", a Type 2 function dictionary shall include those listed in "Table 40 — Additional entries specific to a Type 2 function dictionary".

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 $y_j = C0_j + x^N \times (C1_j - C0_j)$, for $0 \leq j < n$.

Values of **Domain** shall constrain *x* in such a way that if **N** is not an integer, all values of *x* will be non-negative, and if **N** is negative, no value of *x* will 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

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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, [**Domain**$_0$ **Domain**$_1$].

In addition to the entries in "Table 38 — Entries common to all function dictionaries", a Type 3 function dictionary shall include those listed in "Table 41 — Additional entries specific to a Type 3 function dictionary".

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 **Domain**$_0$ shall be strictly less than **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):

$$Domain_0 \leq Bounds_0 < Bounds_1 < \cdots < Bounds_{k-2} \leq Domain_1$$

The **Bounds** array shall describe a series of k half-open intervals, closed on the left and open on the right with the following exceptions:

- the last interval, shall always be closed on the right,
- if $Domain_0 = Bounds_0$ then the first interval shall be closed on both the left and right and the second (next) interval shall be open on the left.

EXAMPLE 1 Using the normal notation where (and) mean open on the left and right, respectively, and [and] mean close on the left and right, the following cases are depicted:

$k = 1$ and $Domain_0 \leq Domain_1$ results in one interval $[Domain_0, Domain_1]$.

$k = 2$ and $Domain_0 < Bounds_0 \leq Domain_1$ results in two intervals $[Domain_0, Bounds_0)$; $[Bounds_0, Domain_1]$.

$k = 2$ and $Domain_0 = Bounds_0 < Domain_1$ results in two intervals $[Domain_0, Bounds_0]$; $(Bounds_0, Domain_1]$.

$k = 3$ and $Domain_0 < Bounds_0 < Bounds_1 < Domain1$ results in three intervals $[Domain_0, Bounds_0)$; $[Bounds_0, Bounds_1)$; $[Bounds_1, Domain_1]$.

$k = 3$ *and* $Domain_0 = Bounds_0 < Bounds_1 < Domain_1$ results in three intervals $[Domain_0, Bounds_0]$;

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$(Bounds_0, Bounds_1)$; $[Bounds_1, Domain_1]$.

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 (interval), defined by **Domain**$_0$ and **Bounds**$_0$ as defined just above; the second function shall apply to *x* values in the second subdomain, defined by **Bounds**$_0$ and **Bounds**$_1$; and so on. The last function shall apply to *x* values in the last subdomain, which includes the upper bound defined by **Bounds**$_{k-2}$ and **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, **Domain**$_0$ ≤ *x* ≤ **Domain**$_1$.

The **Encode** array contains $2 \times k$ 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 $0 \leq i < k$. In this equation, **Bounds**$_{-1}$ means **Domain**$_0$, and **Bounds**$_{k-1}$ means **Domain**$_1$. If the last bound, **Bounds**$_{k-2}$, is equal to **Domain**$_1$, then *x*′ shall be defined to be **Encode**$_{2(k-1)}$. For the degenerate case, if the first bound, **Bounds**$_0$, is equal to **Domain**$_0$ then *x*′ shall be defined to be **Encode**$_0$.

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 2 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. This subset is comprised of the following PostScript language features:

- Expressions involving only integers, real numbers, and boolean values
- Comments
- No composite data structures (such as strings or arrays)
- No procedures
- No variables or names

NOTE 1 Although any function can be sampled (in a Type 4 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 alternative 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, $4 \times m^6$, could be prohibitively large. In practice, such functions can often be written as short, simple PostScript language functions.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

NOTE 2 Type 4 functions also facilitate inclusion of 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 10.6.3, "Spot functions"). All of the predefined spot functions can be written as Type 4 functions.

7.10.5.2 Operators and operands

"Table 42 — Operators in Type 4 functions" lists the operators that can be used in this type of function. The PostScript Language Reference, Third Edition shall define the semantics of these operators and all other syntax rules of the PostScript language. Although the semantics are those of the corresponding PostScript language operators, a full PostScript language compatible interpreter is not required.

Table 42 — Operators in Type 4 functions

Operator Type	Operators				
Arithmetic operators	**abs**	**cvi**	**floor**	**mod**	**sin**
	add	**cvr**	**idiv**	**mul**	**sqrt**
	atan	**div**	**ln**	**neg**	**sub**
	ceiling	**exp**	**log**	**round**	**truncate**
	cos				
Relational, boolean, and bitwise operators	**and**	**false**	**le**	**not**	**true**
	bitshift	**ge**	**lt**	**or**	**xor**
	eq	**gt**	**ne**		
Conditional operators	**if**	**ifelse**			
Stack operators	**copy**	**exch**	**pop**		
	dup	**index**	**roll**		

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The operand syntax for Type 4 functions shall follow PDF conventions rather than PostScript language 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 the PostScript language, no "procedure objects" shall be involved.

7.10.5.3 Type 4 function dictionary

A Type 4 function dictionary shall include the entries in "Table 38 — Entries common to all function dictionaries", as well as other appropriate stream attributes (see "Table 5 — Entries common to all stream dictionaries"). The following example shows a Type 4 function equivalent to the predefined

© ISO 2020 – All rights reserved

spot function **DoubleDot** (see 10.6.3, "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.

A PDF writer shall not use a nesting depth for { } greater than 255 in order to prevent unwanted stack overflows. Annex B, "Operators in Type 4 Functions", contains a summary of these operators.

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 is considered 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 SOLIDIUS character, 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 SOLIDIUS character 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 3986* 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 *Internet RFC 3986* to represent any bytes that would be either unsafe according to *Internet RFC 3986* 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 *Internet RFC 3986*. 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

from the file specification and the process is repeated.

EXAMPLE 2 The relative file specification from Example 1 in this subclause using the .. (two PERIODs) special component

```
../../ArtFiles/Figure1.pdf
```

would yield the absolute specification

```
/HardDisk/ArtFiles/Figure1.pdf
```

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. "Table 43 — Entries in a file specification dictionary" shows the entries in a file specification dictionary. PDF writers should use the **F** entries to specify files. The **UF** entry is optional, but should be included because it enables cross-platform and cross-language compatibility.

Table 43 — Entries in a file specification dictionary

Key	Type	Value
Type	name	(*Required if an **EF**, **EP** 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, "Extending PDF"). This entry shall be independent of the **F** and **UF** 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. A PDF reader shall use the value of the **UF** key, when present, instead of the **F** key. *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **F** entry shall not contain or be derived from the encrypted payload's actual file name. This is to avoid potential disclosure of sensitive information in the original filename. The value of **F** for encrypted payload documents should include the name of the cryptographic filter needed to decrypt the document. See the example in 7.6.7, "Unencrypted wrapper document".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 as defined in 7.9.2.2, "Text string type". The **F** entry should be included along with this entry for backwards compatibility reasons. A PDF reader shall use the value of the **UF** key, when present, instead of the **F** key. (*PDF 2.0*) For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **UF** entry shall not contain or be derived from the encrypted payload's actual file name. This is to avoid potential disclosure of sensitive information in the original filename.
DOS	byte string	(*Optional; deprecated in PDF 2.0*) A file specification string (see 7.11.2, "File specification strings") representing a DOS file name.
Mac	byte string	(*Optional; deprecated in PDF 2.0*) A file specification string (see 7.11.2, "File specification strings") representing a Mac OS file name.
Unix	byte string	(*Optional; deprecated in PDF 2.0*) A file specification string (see 7.11.2, "File specification strings") representing a UNIX file name.
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 1 The use of this entry improves a PDF processor's chances of finding the intended file and allows a PDF processor 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 PDF 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 **F** and **UF** keys 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. *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **EF** dictionary is required for the file stream containing the encrypted payload.
RF	dictionary	(*Optional; PDF 1.3*) A dictionary with the same structure as the **EF** dictionary, which shall be present. For each key in this dictionary, the same key shall appear in the **EF** dictionary of this file specification 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Desc	text string	(*Optional; PDF 1.6*) Descriptive text associated with the file specification. It shall be used for file specification dictionaries referenced from 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").
Thumb	stream	(*Optional; PDF 2.0*) A stream object defining the thumbnail image for the file specification. (See 12.3.4, "Thumbnail images")
EP	dictionary	*(PDF 2.0; Required if this file specification references an encrypted payload document as described in 7.6.7, "*Unencrypted wrapper document*")* The value of this key is an encrypted payload dictionary which identifies that the file specified in the **EF** dictionary is an encrypted payload.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
AFRelationship	name	(*Optional; PDF 2.0*) A name value that represents the relationship between the component of this PDF document that refers to this file specification and the associated file denoted by this file specification dictionary. See 14.13, "Associated files" for more details. These values represent the following relationships: *Source* — shall be used if this file specification is the original source material for the associated content. *Data* — shall be used if this file specification represents information used to derive a visual presentation – such as for a table or a graph. *Alternative* — shall be used if this file specification is an alternative representation of content, for example audio. *Supplement* — shall be used if this file specification represents a supplemental representation of the original source or data that may be more easily consumable (e.g., A MathML version of an equation). *EncryptedPayload* — shall be used if this file specification is an encrypted payload document that should be displayed to the user if the PDF processor has the cryptographic filter needed to decrypt the document. *FormData* — shall be used if this file specification is the data associated with the AcroForm (see 12.7.3, "Interactive form dictionary") of this PDF. *Schema* — shall be used if this file specification is a schema definition for the associated object (e.g. an XML schema associated with a metadata stream). *Unspecified* — shall be used when the relationship is not known or cannot be described using one of the other values. NOTE 2 *Unspecified* is to be used only when no other value correctly reflects the relationship. Second-class names (see Annex E, "Extending PDF") should be used to represent other types of relationships. Default: *Unspecified* NOTE 3 The value of **AFRelationship** does not explicitly provide any processing instructions for a PDF processor. It is provided for information and semantic purposes for those processors that are able to use such additional information.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 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 PDF processor.)

© ISO 2020 – All rights reserved

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 file 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 in the PDF file'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 43 — Entries in a file specification dictionary") should be used to provide a textual description of the embedded file, which can be displayed in the user interface of an interactive PDF processor. 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 ECMAScript name tree associates name strings with document-level ECMAScript actions (see 12.6.4.17, "ECMAScript actions").

The stream dictionary describing an embedded file shall contain the standard entries for any stream, such as **Length** and **Filter** (see "Table 5 — Entries common to all stream dictionaries"), as well as the additional entries shown in "Table 44 — Additional entries in an embedded file stream dictionary".

Table 44 — 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, required in the case of an embedded file stream used as an associated file*) The subtype of the embedded file. The value of this entry shall conform to the MIME media type names defined in *Internet RFC 2046*, with the provision that characters not permitted in names shall use the 2-character hexadecimal code format described in 7.3.5, "Name objects". NOTE The media type for PDF is defined by *Internet RFC 8118*.
Params	dictionary	(*Optional, required in the case of an embedded file stream used as an associated file*) An embedded file parameter dictionary that shall contain additional file-specific information (see "Table 45 — Entries in an embedded file parameter dictionary").

Table 45 — Entries in an embedded file parameter dictionary

Key	Type	Value
Size	integer	(*Optional*) The size of the uncompressed embedded file, in bytes. See "Table 155 — Entries in a collection field dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
CreationDate	date	(*Optional*) The date and time when the embedded file was created. See "Table 155 — Entries in a collection field dictionary".
ModDate	date	(*Optional, required in the case of an embedded file stream used as an associated file*) The date and time when the embedded file was last modified. See "Table 155 — Entries in a collection field dictionary".
Mac	dictionary	(*Optional; deprecated in PDF 2.0*) A subdictionary containing additional information specific to Mac OS files.
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 (defined in *Internet RFC 1321*) to the bytes of the embedded file stream. NOTE This is strictly a checksum, and is not used for security purposes.

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

[$string_1$ $stream_1$
$string_2$ $stream_2$
...
$string_n$ $stream_n$
]

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, object 31 is an embedded file stream containing Encapsulated Postscript File (EPSF) file Sunset.eps. The file specification dictionary's **RF** entry specifies an array, object 30, identifying a set of embedded files, forming a DCS 1.0 set.

```
10 0 obj                              %File specification dictionary
      <</Type /Filespec
         /F (Sunset.eps)
         /UF (Sunset.eps)
         /EF <</F 21 0 R
               /UF 41 0 R
             >>
         /RF <</UF 30 0 R>>           %Related files array

endobj


30 0 obj                              %Related files array
         [(Sunset.eps) 31 0 R         %Embedded file stream for the EPSF Sunset.eps
         (Sunset.C) 32 0 R
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            (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 3986*.

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 *Internet RFC 3986*. Because 7-bit U.S. ASCII is a strict subset of PDFDocEncoding, this value shall also be considered to be in PDFDocEncoding.

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 46 — Entries in a collection item dictionary" describes the entries in a collection item dictionary.

Table 46 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 47 — Entries in a collection subitem dictionary"). The type of each entry shall match the type of data identified by the collection field dictionary (see "Table 155 — Entries in a collection field dictionary") referenced by the same key in the collection schema dictionary (see "Table 154 — Entries in a collection schema dictionary"). 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 47 — Entries in a collection subitem dictionary" describes the entries in a collection subitem dictionary.

Table 47 — 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 155 — Entries in a collection field dictionary"). 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 an interactive PDF processor sorts the items in the collection. Default: *none*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

7.12 Extensions dictionary

7.12.1 General

An optional key, **Extensions**, in the document's catalog dictionary shall have as its value an Extensions dictionary (see "Table 48 — Entries in an extensions dictionary") containing one or more entries identifying developer-defined extensions to this document. The keys allowed in an extensions dictionary, are shown in "Table 48 — Entries in an extensions dictionary", and may optionally contain a **Type** entry whose value is the name *Extensions*. The remaining keys shall be names consisting of registered prefix names of the developer extensions used, as described in Annex E, "Extending PDF". The values of those prefix name keys shall be developer extensions dictionaries (see "Table 49 — Entries in a developer extensions dictionary") specifying developer-defined version information. The extensions dictionary, all developer extensions dictionaries, as well as their entries, shall be direct objects (i.e., this information shall be nested directly within the catalog dictionary with no indirect

© ISO 2020 – All rights reserved

objects used).

7.12.2 Extensions dictionary

"Table 48 — Entries in an extensions dictionary" describes the entries in an extensions dictionary.

Table 48 — Entries in an extensions dictionary

Key	Type	Value
Type	name	(*Optional; , shall be a direct objectif present*) The type of PDF object that this dictionary describes; if present, shall be *Extensions*.
Prefix name	array or dictionary	(*Required; PDF 1.7; shall be a direct object*) A developer extensions dictionary, or an array of developer extension dictionaries (*PDF 2.0*), denoting extensions used (See "Table 49 — Entries in a developer extensions dictionary"). The key shall be a registered prefix name as described in Annex E, "Extending PDF". This entry is repeated for each different developer's extensions used. Multiple distinct registered prefix names for different developers may be used for each different developer's extensions used. See Example 3 below. NOTE Arrays of developer extension dictionaries were introduced in this document (2020).

7.12.3 Developer extensions dictionary

"Table 49 — Entries in a developer extensions dictionary" describes the entries in a developer extensions dictionary.

Table 49 — Entries in a developer extensions dictionary

Key	Type	Value
Type	name	(*Optional; shall be a direct object if present*) The type of PDF object that this dictionary describes; if present, shall be *DeveloperExtensions*.
BaseVersion	name	(*Required; shall be a direct object*) 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 dictionary").
ExtensionLevel	integer	(*Required; shall be a direct object*) 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 should increase over time.
URL	string	(*Optional; PDF 2.0; shall be a direct object if present*) A URL that refers to the documentation for this extension (see Annex E, "Extending PDF").
ExtensionRevision	text string	(*Optional; PDF 2.0; shall be a direct object if present*) An optional text string that provides additional revision information on the extension level being used. NOTE The **ExtensionRevision** key was introduced in this document (2020).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

7.12.4 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 dictionary"). 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 dictionary **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 dictionary. This is because it reflects the version of the standard that has been extended and not the version of this particular file.

NOTE The value of **BaseVersion** is not to be interpreted as a real number but as two integers with a PERIOD (2Eh) between them.

7.12.5 ExtensionLevel

The value of the **ExtensionLevel** 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 **ExtensionLevel** numbers shall be a monotonically increasing sequence over time.

7.12.6 URL

The value of the optional **URL** entry shall be a string containing a URL to the documentation that is provided by the developer of this extension and as registered in the PDF Registry (see Annex E, "Extending PDF").

EXAMPLE 1

```
%PDF-2.0
<< /Type /Catalog
   /Extensions
         << /Type /Extensions
            /ADBE << /BaseVersion /1.7 /ExtensionLevel 3
                     /URL (http://www.adobe.com/documentation/ext3.pdf)
                  >>
         >>
>>
```

EXAMPLE 2 This is a minimal example.

```
%PDF-2.0
<< /Type /Catalog
   /Extensions << /GLGR << /BaseVersion /1.7 /ExtensionLevel 1002 >> >>
>>
```

EXAMPLE 3 This example shows the use of both arrays (*PDF 2.0*) and dictionaries with different developer prefixes.

```
%PDF-2.0
<< /Type /Catalog
   /Extensions <<
      /Type /Extensions
      /ISO_ [ % Multiple developer extensions under the registered prefix "ISO_"
               << /Type /DeveloperExtensions
                  /BaseVersion /2.0
                  /ExtensionLevel 24064
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
                  /URL (https://www.iso.org/standard/77686.html)
               >>
               << /Type /DeveloperExtensions
            /BaseVersion /2.0
            /ExtensionLevel 24654
            /URL (https://www.iso.org/standard/79150.html)
         >>
      ]
      /GLGR << /Type /DeveloperExtensions
            /BaseVersion /1.7
            /ExtensionLevel 1002
            /URL (http://www.generallanguage.com/ISO32000/extension1002.pdf)
            >>
   >>
>>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

8 Graphics

8.1 General

The graphics operators used in PDF content streams describe the appearance of pages that are to be reproduced on an 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 specialised 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. Specialised 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").
- 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 form 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 — Graphics objects" shows the ordering rules for the operations that define graphics objects. Only those operators that are listed in "Figure 9 — Graphics objects" for each type of graphics object or in the intervals between graphics objects (called the *content stream level* in the figure) shall be used in that context. Every content stream begins at the content stream level, where changes may be made to the graphics state, such as colours and text attributes, as discussed in the following subclauses.

In "Figure 9 — Graphics objects", arrows indicate the operators that mark the beginning or end of each

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

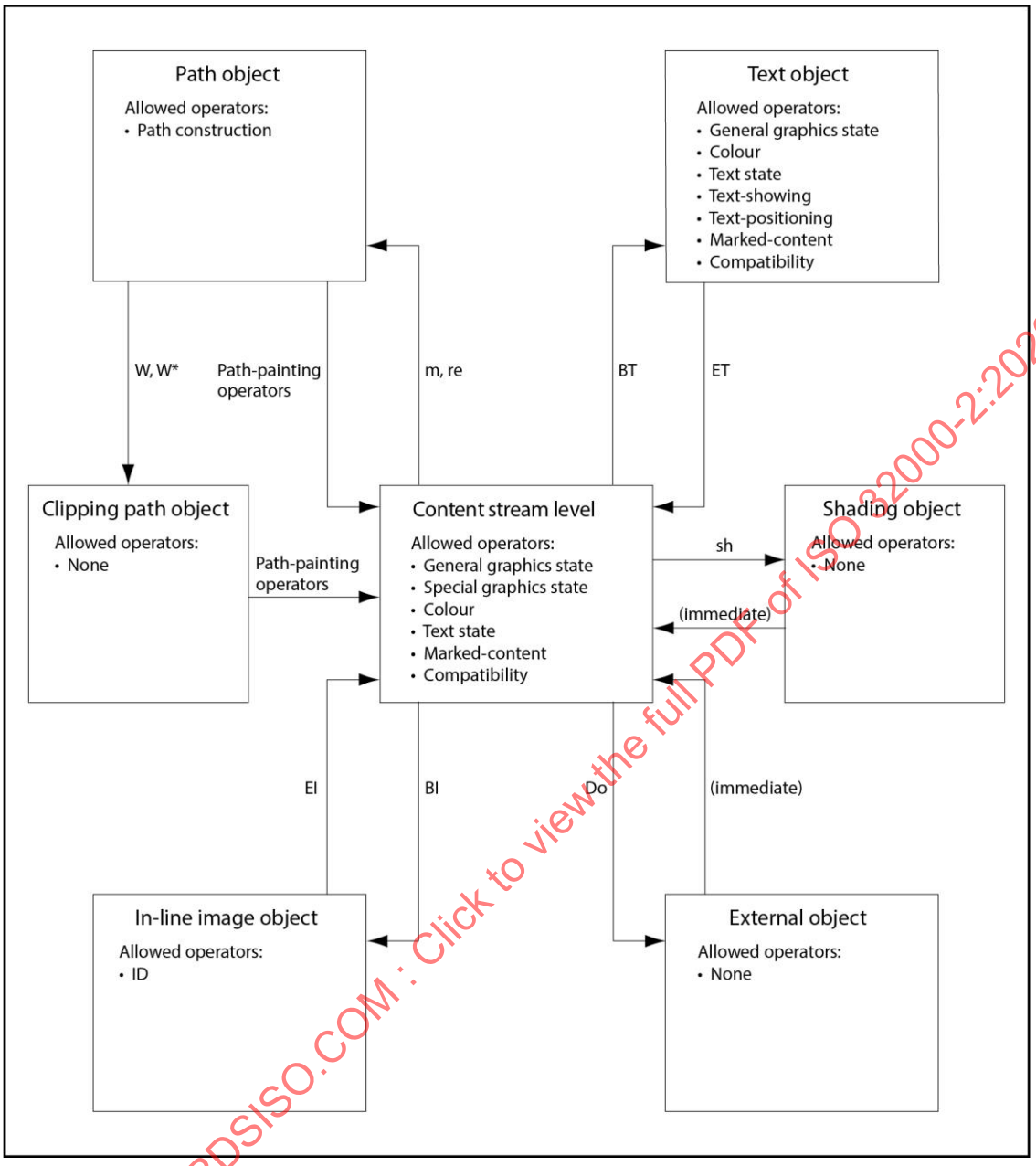
type of graphics object. Some operators are identified individually, others by general category. "Table 50 — Operator categories" summarises these categories for all PDF operators.

Table 50 — Operator categories

Category	Operators	Location
General graphics state	**w, J, j, M, d, ri, i, gs, q, Q**	"Table 56 — Graphics state operators"
Special graphics state	**cm**	"Table 56 — Graphics state operators"
Path construction	**m, l, c, v, y, h, re**	"Table 58 — Path construction operators"
Path painting	**S, s, f, F, f*, B, B*, b, b*, n**	"Table 59 — Path-painting operators"
Clipping paths	**W, W***	"Table 60 — Clipping path operators"
Text objects	**BT, ET**	"Table 105 — Text object operators"
Text state	**Tc, Tw, Tz, TL, Tf, Tr, Ts**	"Table 103 — Text state operators"
Text positioning	**Td, TD, Tm, T***	"Table 106 — Text-positioning operators"
Text showing	**Tj, TJ, ', "**	"Table 107 — Text-showing operators"
Type 3 fonts	**d0, d1**	"Table 111 — Type 3 font operators"
Colour	**CS, cs, SC, SCN, sc, scn, G, g, RG, rg, K, k**	"Table 73 — Colour operators"
Shading patterns	**Sh**	"Table 76 — Shading operator"
Inline images	**BI, ID, EI**	"Table 90 — Inline image operators"
XObjects	**Do**	"Table 86 — XObject operator"
Marked-content	**MP, DP, BMC, BDC, EMC**	"Table 351 — Entries in a data dictionary"
Compatibility	**BX, EX**	"Table 33 — Compatibility operators"

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 content stream level.

NOTE 1 "Table 50 — Operator categories" and "Figure 9 — Graphics objects" were updated in this document (2020).

NOTE 2 A PDF reader can process a content stream whose operations violate these rules for describing graphics objects and can produce unpredictable behaviour, even though it can display and print the stream correctly. PDF processors that attempt to extract graphics objects for editing or other purposes often 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

© ISO 2020 – All rights reserved

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 content stream 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. When processing a PDF content stream a PDF processor 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. PDF processors 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 subclause 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 subclause 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 subclauses 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"

© ISO 2020 – All rights reserved

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 — Device space" shows how the same graphics object, specified in device space, can appear drastically different when rendered on different output devices.

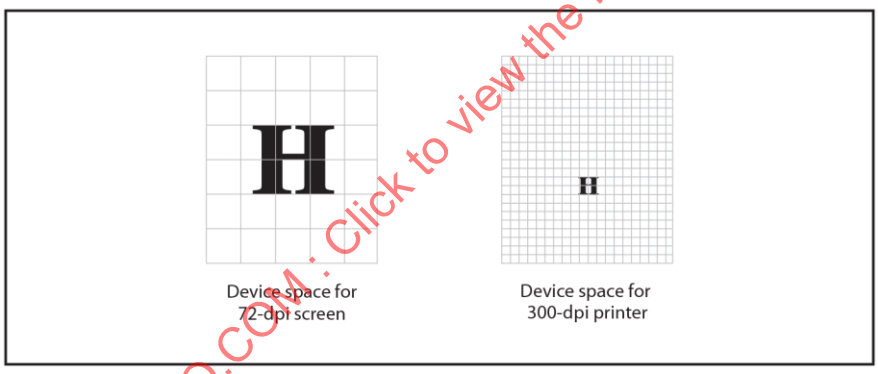


Figure 10 — Device space

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 initialised 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 31 — Entries in a page object"). 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

NOTE 1 In the PostScript language, 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 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 PDF reader may adjust the CTM for the native resolution of a particular output device, maintaining the device-independence of the PDF page description. "Figure 11 — User space" shows how this allows an object specified in user space to appear the same regardless of the device on which it is rendered.

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 might 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 can 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.

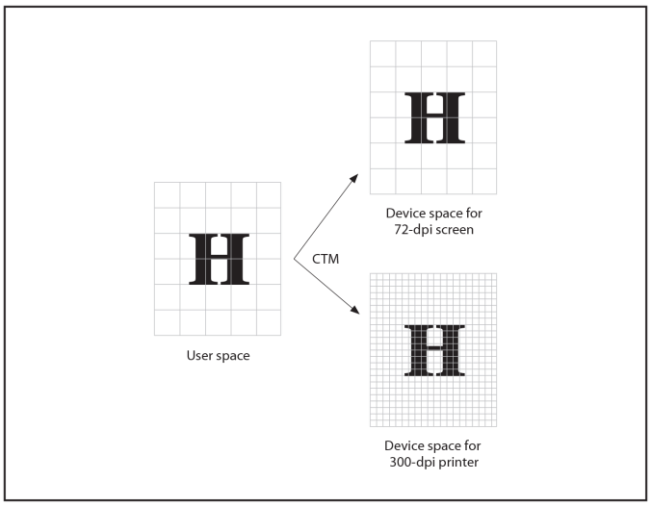


Figure 11 — User space

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

8.3.2.4 Other coordinate spaces

In addition to device space and user space, PDF uses a variety of other coordinate spaces for specialised 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 9.6.4, "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 — Relationships among coordinate systems" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

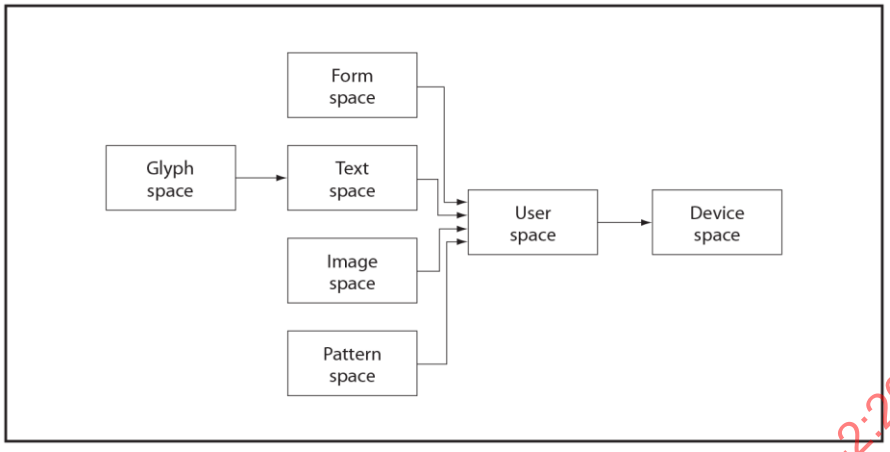


Figure 12 — Relationships among coordinate systems

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.

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 subclause 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 [r_c r_s -r_s r_c 0 0], where $r_c = \cos(q)$ and $r_s = \sin(q)$ which has the effect of rotating the coordinate system axes by an angle q counter clockwise.
- Skew shall be specified by [1 w_x w_y 1 0 0], where $w_x = \tan(a)$ and $w_y = \tan(b)$ which skews the x axis by an angle a and the y axis by an angle b.

"Figure 13 — Effects of coordinate transformations" shows examples of each transformation. The directions of translation, rotation, and skew shown in the figure correspond to positive values of the array elements.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

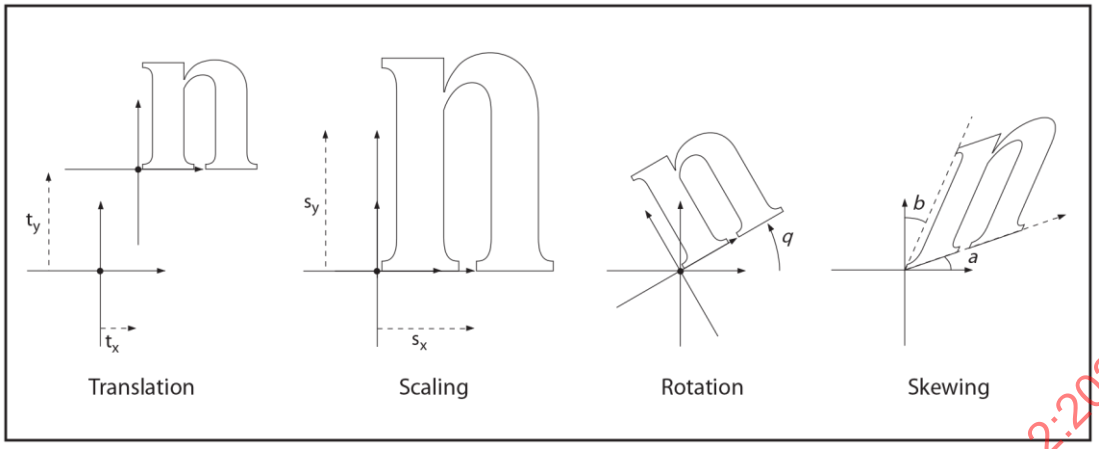


Figure 13 — Effects of coordinate transformations

NOTE 1 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 need to be done in the following order: Translate, Rotate, Scale or skew.

"Figure 14 — Effect of transformation order" 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.

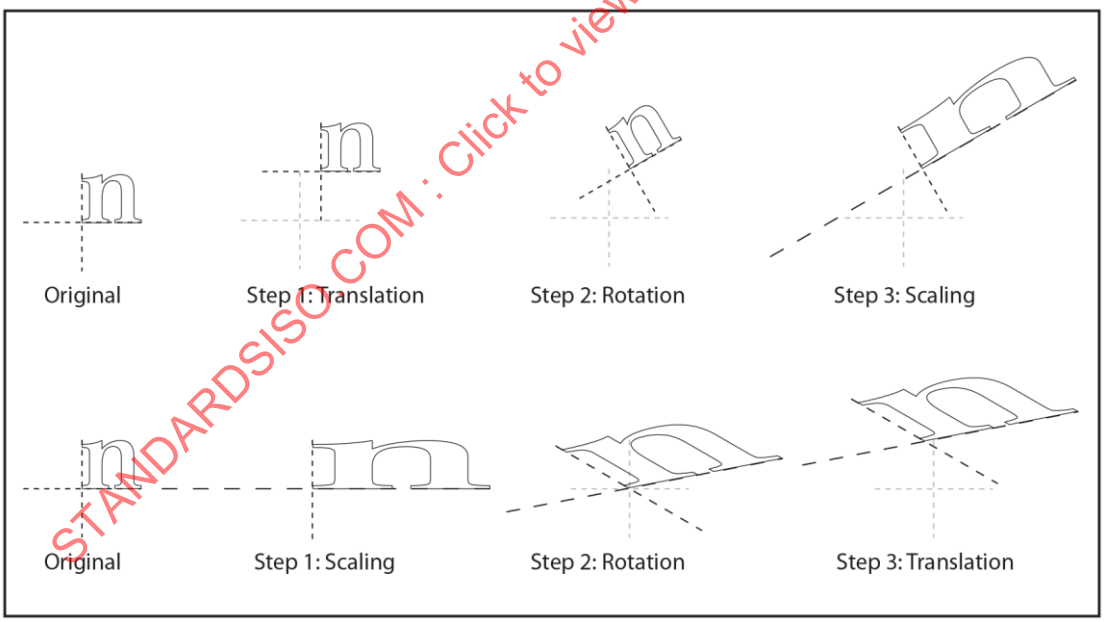


Figure 14 — Effect of transformation order

NOTE 2 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

the *x* and *y* axes no longer perpendicular; the recommended translate-rotate-scale ordering results in no distortion.

8.3.4 Transformation matrices

This subclause 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:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF 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 PDF processor 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 PDF processor shall initialise the graphics state at the beginning of each page with the values specified in "Table 51 — Device-independent graphics state parameters" and "Table 52 — Device-dependent graphics state parameters". "Table 51 — Device-independent graphics state parameters" lists those graphics state parameters that are device-independent and are appropriate to specify in page descriptions. The parameters listed in "Table 52 — Device-dependent graphics state parameters" control details of the rendering (scan conversion) process and are device-dependent; a page

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

description that is intended to be device-independent should not be written to modify these parameters.

Table 51 — 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 size of the MediaBox.
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* that shall 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*.
line cap	integer	A code specifying the shape of the start and endcaps for an open stroked path or the caps at both ends of dashes in a stroked path (see 8.4.3.3, "Line cap style"). Initial value: *0*, for 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 miter limit imposes a maximum on the ratio of the miter length to the line width. When the limit is exceeded, the join is converted from a miter to a bevel (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 that shall be used when paths are stroked (see 8.4.3.6, "Line dash pattern"). Initial value: *[] 0*, a solid line.
rendering intent	name	The rendering intent that shall be used when converting CIE-based colours to device colours (see 8.6.5.8, "Rendering intents"). Initial value: *RelativeColorimetric*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Parameter	Type	Value
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 10.7.5, "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 (*array is deprecated in PDF 2.0*)	(*PDF 1.4, array is deprecated in PDF 2.0*) The current blend mode that shall be used in the transparent imaging model (see 11.3.5, "Blend mode"). A PDF 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"). The value shall be either a name object, designating one of the standard blend modes listed in "Table 134 — Standard separable blend modes" and "Table 135 — Standard non-separable blend modes" in 11.3.5, "Blend mode", or an array of such names. In the latter case, the PDF reader shall use the first blend mode in the array that it recognises (or **Normal** if it recognises none of them). Initial value: ***Normal***.
soft mask	dictionary or name	(*PDF 1.4*) A soft-mask dictionary (see 11.6.5.1, "Soft-mask dictionaries") 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"), or the name *None* if no such mask is specified. A PDF 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: *None*.
alpha constant	number	(*PDF 1.4*) The constant shape or constant opacity value that shall 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 PDF 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*.
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*.
black point compensation	name	*(PDF 2.0)* The black point compensation algorithm that shall be used when converting CIE-based colours (see 8.6.5.9, "Use of black point compensation"). Initial value: *Default*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 52 — 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 colourants should cause the corresponding areas of other colourants to be erased (*false*) or left unchanged (*true*); see 8.6.7, "Overprint control". PDF 1.3, introduced 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 10.4.2.4, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: a PDF reader shall initialise 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 10.4.2.4, "Conversion from DeviceRGB to DeviceCMYK"). Initial value: a PDF reader shall initialise this to a suitable device dependent value.
transfer	function, name, or array	(*PDF 1.2, deprecated in PDF 2.0*) A function that adjusts device gray or colour component levels to compensate for nonlinear response in a particular output device (see 10.5, "Transfer functions"). Initial value: a PDF reader shall initialise 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 10.6, "Halftones"). Initial value: a PDF reader shall initialise this to a suitable device dependent value.
flatness	number	The precision with which curves shall be rendered on the output device (see 10.7.2, "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*.
smoothness	number	(*PDF 1.3*) The precision with which colour gradients are to be rendered on the output device (see 10.7.3, "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 PDF reader shall initialise this to a suitable device dependent value.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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

© ISO 2020 – All rights reserved

halftone is set only with a graphics state parameter dictionary.

In general, a PDF processor, 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 PDF processor 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 clipped 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 51 — Device-independent graphics state parameters" and "Table 52 — Device-dependent graphics state parameters", some of the parameters — colour space, colour, 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 is to 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 subclause gives details of several of the device-independent graphics state parameters listed in "Table 51 — Device-independent graphics state parameters".

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 10.7.5, "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 both ends of open subpaths (and dashes 8.4.3.6, "Line dash pattern") when they are stroked. "Table 53 — Line cap styles" shows the allowed values.

Table 53 — 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 54 — Line join styles" shows the allowed 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 54 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 pie-slice-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.

A zero length dash occurring at a zero length subpath segment does not have a determinable direction and thus, if the line caps are non-round is rendered in an implementation-dependent manner.

In a closed subpath that is dashed, if the first segment starts with an on-dash and the last segment ends within an on-dash, then they shall be joined.

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 — Miter length"). 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:

$$\frac{miterLength}{lineWidth} = \frac{1}{\sin\frac{j}{2}}$$

When the line width is zero, the miter length is zero.

NOTE Very large miter lengths are allowed.

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*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

less than 60 degrees, and a limit of 10.0 converts them for *j* less than approximately 11.5 degrees.

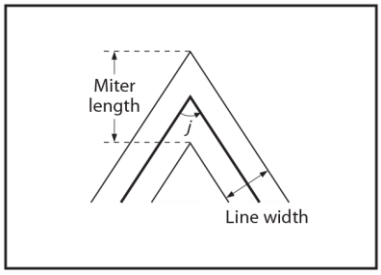


Figure 15 — Miter length

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 be a number that specifies the distance into the dash pattern at which to start the dash. If the dash phase is negative, it shall be incremented by twice the sum of all lengths in the dash array until it is positive. 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 55 — Examples of line dash patterns" shows examples of line dash patterns. If the dash array is empty, the dash phase shall be zero and the path shall be stroked with a solid, unbroken line.

Table 55 — 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, ...
[2 1 3] 0		2 on, 1 off, 3 on, 2 off, 1 on, 3 off, 2 on, ...
[2 1 3] -2		2 off, 2 on, 1 off, 3 on, 2 off, 1 on, 3 off, ...

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. The treatment of overlapping line caps shall follow the rules given in 11.6.2, "Specifying source and backdrop colours". 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. If the end of a dashed segment coincides exactly with a join point, then the end cap is painted before the corner.

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.

NOTE As noted in 8.5.3.2, "Stroking" and in "Table 58 — Path construction operators", closed paths have no end caps, but the individual dash segments of a path stroked using a non-empty line dash pattern are individually open paths and therefore receive end cap processing as specified in the graphics state. If any dash segment includes a corner then that corner is painted using the current join style in the graphics state. If a corner is not contained within any dashed segment the corner is not painted.

8.4.4 Graphics state operators

"Table 56 — Graphics state operators" shows the operators that set the values of parameters in the graphics state. (See also the colour operators listed in "Table 73 — Colour operators" and the text state operators in "Table 103 — Text state operators".)

Table 56 — 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Operands	Operator	Description
flatness	**i**	Set the flatness tolerance in the graphics state (see 10.7.2, "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 subclause).

8.4.5 Graphics state parameter dictionaries

While some parameters in the graphics state may be set with individual operators, as shown in "Table 56 — Graphics state operators", 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 57 — Entries in a graphics state parameter dictionary". 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 56 — Graphics state operators" and "Table 57 — Entries in a graphics state parameter dictionary"; 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 57 — Entries in a graphics state parameter dictionary

Key	Type	Value
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 a number (see 8.4.3.6, "Line dash pattern").

© ISO 2020 – All rights reserved

Key	Type	Value
RI	name	(*Optional; PDF 1.3*) The name of the rendering intent (see 8.6.5.8, "Rendering intents").
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, two separate overprint parameters were defined: 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 10.4.2.4, "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 undercolour-removal function, which maps the interval [0.0 1.0] to the interval [−1.0 1.0] (see 10.4.2.4, "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 undercolour-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, name, or array	(*Optional, deprecated in PDF 2.0*) The transfer function, which maps the interval [0.0 1.0] to the interval [0.0 1.0] (see 10.5, "Transfer functions"). The value shall be either a single function (which applies to all process colourants) or an array of four functions (which apply to the process colourants individually). The name *Identity* may be used to represent the *Identity* function.
TR2	function, name, or array	(*Optional; PDF 1.3, deprecated in PDF 2.0*) 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 10.6, "Halftones") or the name **Default**, denoting the halftone that was in effect at the start of the page.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
FL	number	(*Optional; PDF 1.3*) The flatness tolerance (see 10.7.2, "Flatness tolerance").
SM	number	(*Optional; PDF 1.3*) The smoothness tolerance (see 10.7.3, "Smoothness tolerance").
SA	boolean	(*Optional*) A flag specifying whether to apply automatic stroke adjustment (see 10.7.5, "Automatic stroke adjustment").
BM	name or array (*array is deprecated in PDF 2.0*)	(*Optional; PDF 1.4; array is deprecated in PDF 2.0*) The current blend mode that shall be used in the transparent imaging model (see 11.3.5, "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*). This flag also governs the interpretation of the **SMask** entry, if any, in an image dictionary (see 8.9.5, "Image dictionaries").
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"). This flag controls the behavior of glyphs obtained from any font type, including Type 3.
UseBlackPtComp	name	(*Optional; PDF 2.0*) This graphics state parameter controls whether black point compensation is performed while doing CIE-based colour conversions. It shall be set to either *OFF*, *ON* or *Default*. The semantics of *Default* are up to the PDF processor. See 8.6.5.9, "Use of black point compensation". The default value is: *Default*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
HTO	array	(*Optional; PDF 2.0*) Halftone origin, specified as an array of two numbers specifying the X and Y location of the halftone origin in the current coordinate system. Although the numbers are specified in the current coordinate system, changes to the current coordinate system (for example as a result of invocation of a form XObject) do not move the halftone origin relative to the underlying device coordinate system. NOTE: The **HTO** key is very similar to the **HTP** key defined in PDF versions up to PDF 1.3 (1st Edition), but differs in the coordinate system used.

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 (*deprecated in PDF 2.0*) that inverts its value, $f(x) = 1 - x$. In the second, overprint is turned off, and the dictionary includes a parabolic transfer function (*deprecated in PDF 2.0*), $f(x) = (2x - 1)^2$, 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
      <</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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 initialised to the size of the **MediaBox**.

A path may contain any combination of zero or more line segments, which may be straight or curved. Paths 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 can be concave or convex, can contain multiple subpaths representing disjoint areas, and can 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

8.5.2 Path construction operators

8.5.2.1 General

A path description is built up through the invocation of one or more path construction operators that 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 58 — Path construction operators" shows the path construction operators. All operands shall be numbers denoting coordinates in user space.

Table 58 — 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).
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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Operands	Operator	Description
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 — Cubic Bézier curve generated by the c operator". (The starting point, P_0, is defined implicitly by the current point.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

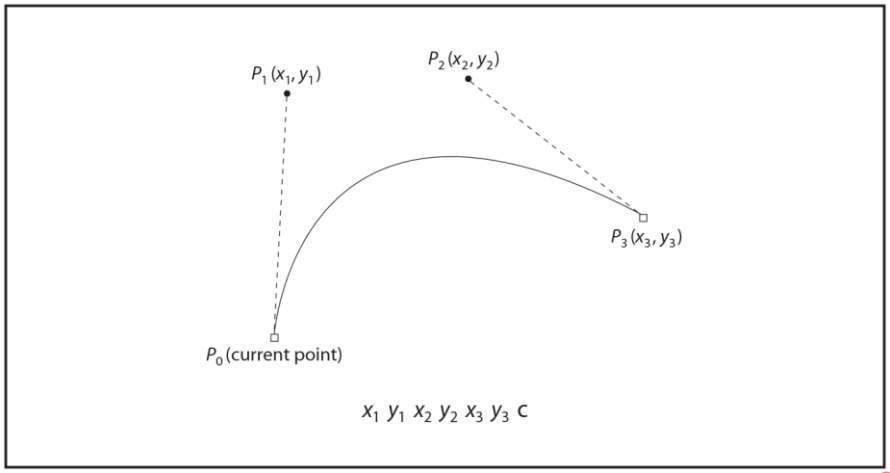


Figure 16 — Cubic Bézier curve generated by the c operator

Two more operators, **v** and **y**, each specify one of the two control points implicitly (see "Figure 17 — Cubic Bézier curves generated by the v and y operators"). 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.

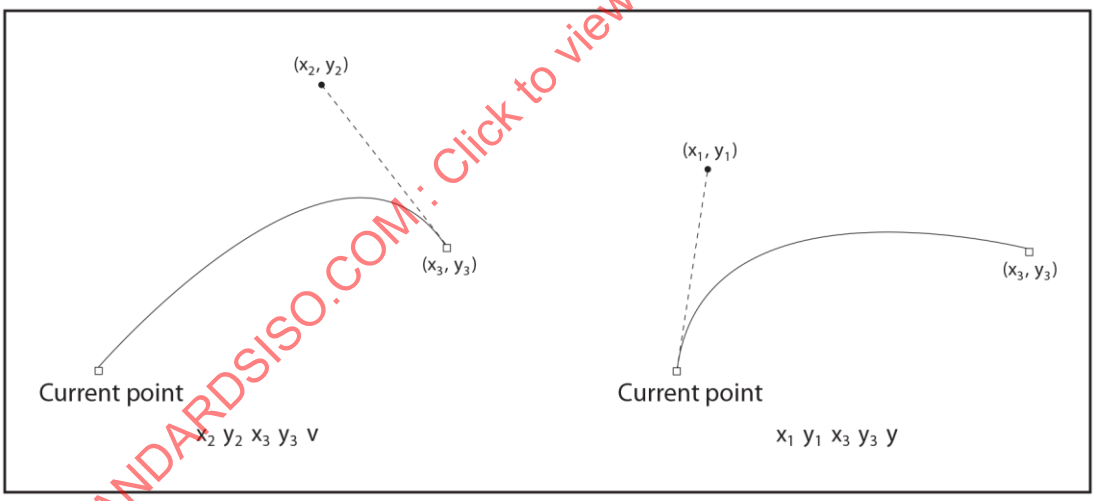


Figure 17 — Cubic Bézier curves generated by the v and y operators

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. Attempting to execute a painting operator when the current path is undefined (at the beginning of a new page or immediately after a painting operator has been executed) shall generate an error. "Table 59 — Path-painting operators" lists all the path-

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

painting operators.

Table 59 — 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 non-zero winding number rule to determine the region to fill (see 8.5.3.3.2, "Non-zero winding number rule"). Any subpaths that are open shall be implicitly closed before being filled.
—	**F**	Equivalent to **f**; deprecated in PDF 2.0 and included only for compatibility. Although PDF readers shall be able to accept this operator, PDF writers 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 non-zero 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 non-zero 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 is defined by the current line width parameter (8.4.3.2, "Line width").

© ISO 2020 – All rights reserved

- The colour or pattern of the line is defined 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 (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 10.7.5, "Automatic stroke adjustment").
- For transparency compositing purposes a path shall be treated as a single graphics object as described in 11.6.2, "Specifying source and backdrop colours".

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 centred 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 of a non-degenerate subpath. In the latter case, the line caps shall always be painted, since their orientation is determined by the direction of the underlying path except in the case of a degenerate subpath. 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, except that if the last subpath in the path is a single-point open subpath (specified by a trailing **m** operator), it shall be disregarded and not considered to be part of the path. If a subpath is degenerate (consists entirely of one or more points at the same coordinates), the subpath shall be considered to enclose the single device pixel lying under that point; the result is device-dependent and not generally useful.

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 10.7.4, "Scan conversion rules".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 non-zero winding number rule and the even-odd rule, both discussed in detail below. The non-zero 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 Non-zero winding number rule

The *non-zero 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 non-zero 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 — Non-zero winding number rule". 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

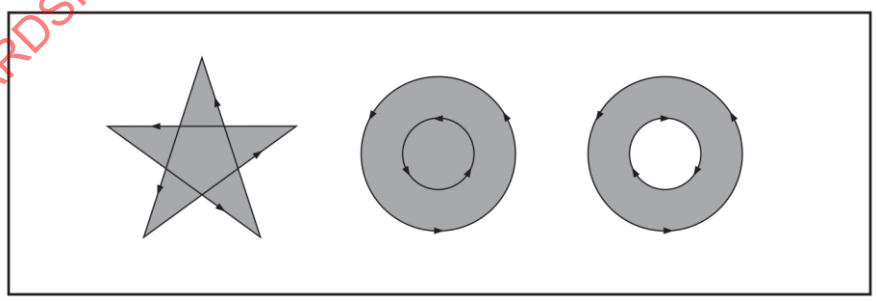


Figure 18 — Non-zero winding number rule

8.5.3.3.3 Even-odd rule

An alternative to the non-zero 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

© ISO 2020 – All rights reserved

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 non-zero winding number rule for paths with simple shapes, but produces different results for more complex shapes.

"Figure 19 — Even-odd rule" 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.

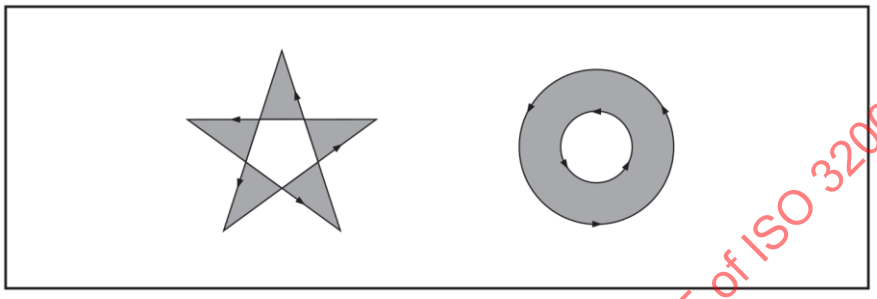


Figure 19 — Even-odd rule

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. Subclause 8.5.3.3, "Filling" defines what is inside a path as well as stating rules for closing paths and for degenerate paths. For a given path definition, the same area that would be filled by the **f** operator is the area that would be used for a clip.

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 60 — Clipping path operators") 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 60 — Clipping path operators

Operands	Operator	Description
—	**W**	Modify the current clipping path by intersecting it with the current path, using the non-zero 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 can also be used for clipping; see 9.3.6, "Text rendering mode".

The **n** operator (see "Table 59 — Path-painting operators") 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. The colour facilities are divided into two parts:

- *Colour specification*. A PDF file 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 PDF processor 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.

When the device is a subtractive colour device, "rendering for separations" may be implemented (see 10.8.2, "Separations"). In addition, a PDF reader may optionally support "separation simulation" for any device (see 10.8.3, "Separation simulation"). In both cases overprinting (see 8.6.7, "Overprint control") may be enabled. "Figure 20 — Colour specification" and "Figure 21 — Colour rendering" illustrate the division between PDF's (device-independent) colour specification and (device-dependent) colour-rendering facilities. This subclause describes the colour specification features, covering everything that PDF documents need to specify colours. The facilities for controlling colour

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

rendering are described in clause 10, "Rendering"; a PDF processor 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 73 — Colour operators" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

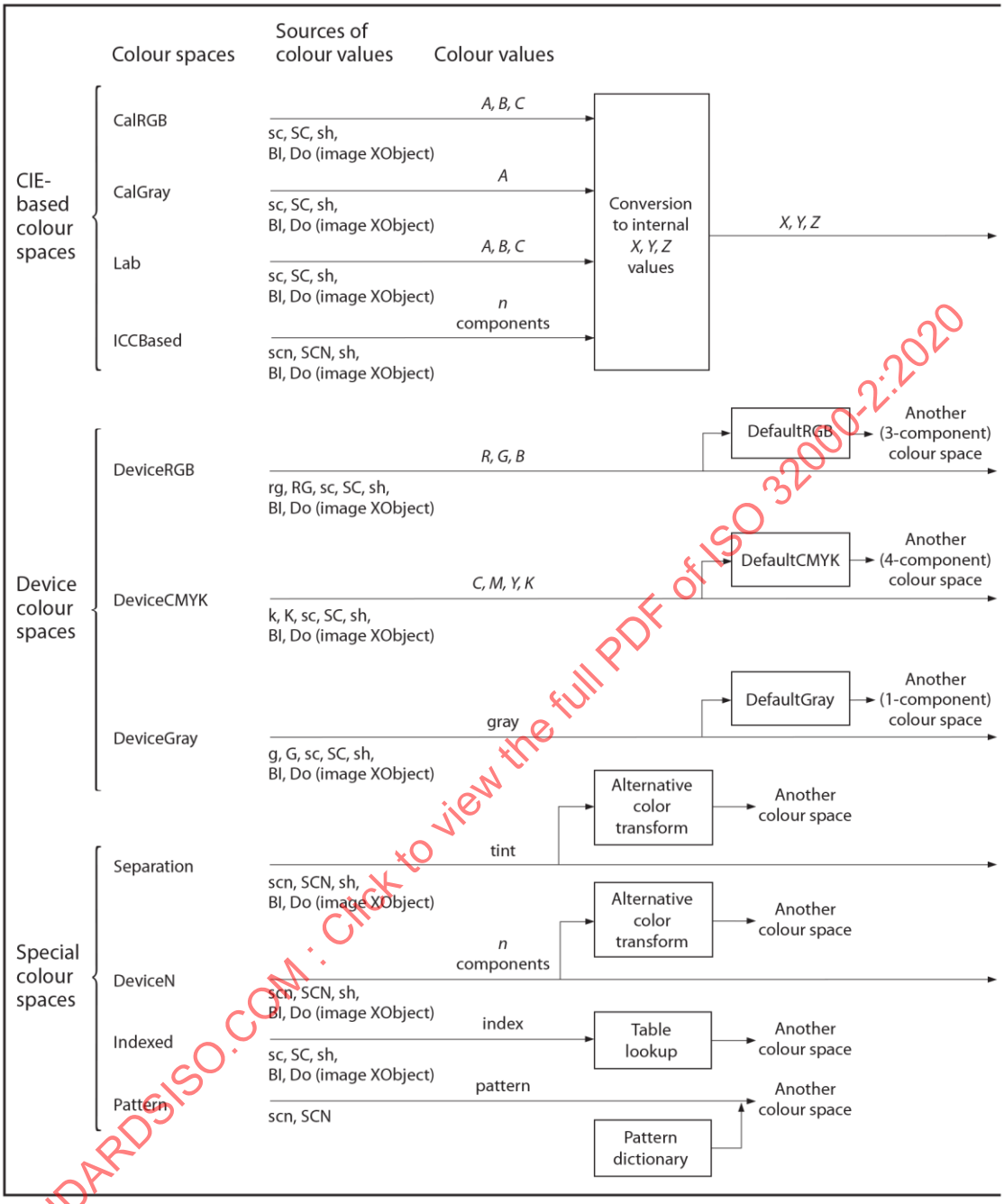


Figure 20 — Colour specification

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

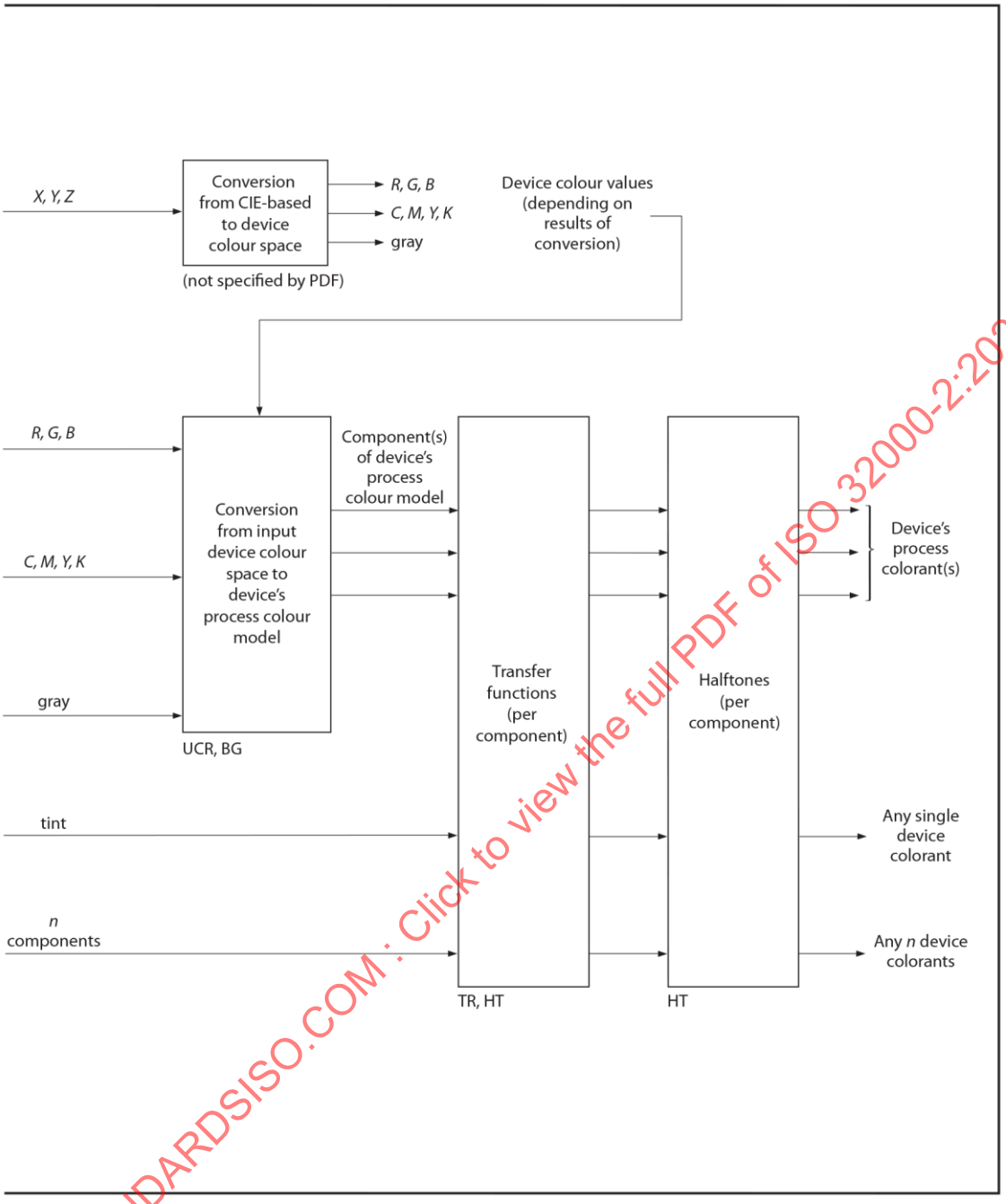


Figure 21 — Colour rendering

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

"Table 61 — Colour space families" summarises the colour space families in PDF.

Table 61 — 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*)

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 characterise 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 colourants, such as quantities of ink or intensities of display

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

phosphors. This enables a PDF 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 22 — Additive and subtractive colour"). 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".

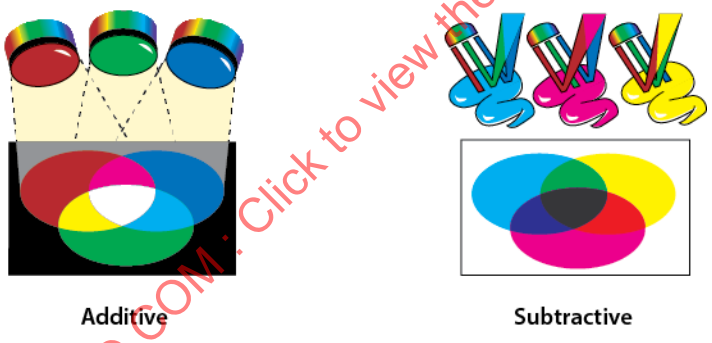


Figure 22 — Additive and subtractive colour

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 colour 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 initialise 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 colourants 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
1 0 0 RG                         %Set a pure red colour for stroking operations
```

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 initialise 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 colourants. 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 colourant 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

initialise 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 23 — Uncalibrated colour" 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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Figure 23 — Uncalibrated colour

A PDF reader shall ignore **CalCMYK** colour space attributes and render colours specified in this family as if they had been specified using **DeviceCMYK**.

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 completely removed. 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").

NOTE 2 The details of the CIE colorimetric 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

© ISO 2020 – All rights reserved

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 digitised 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 24 — Component transformations in a CIE-based ABC colour space").

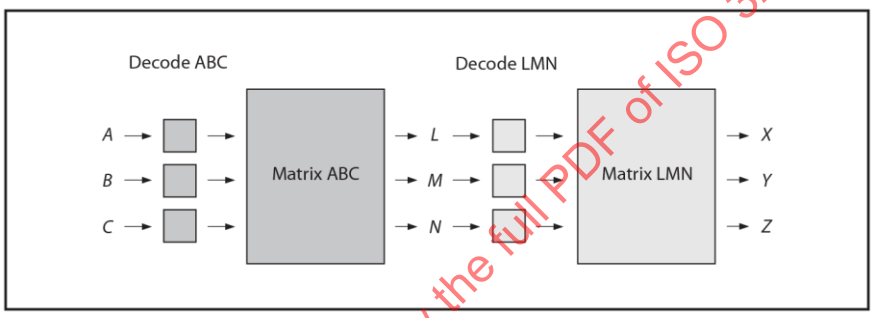


Figure 24 — Component transformations in a CIE-based ABC colour space

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 characterise 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 subclauses on specific colour space families for details.

Setting the current stroking or nonstroking colour space to any CIE-based colour space shall initialise 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 25 — Component transformations in a CIE-based A colour space" illustrates the transformations of the *A* component to *X*, *Y*, and *Z* components of the CIE 1931

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

XYZ space.

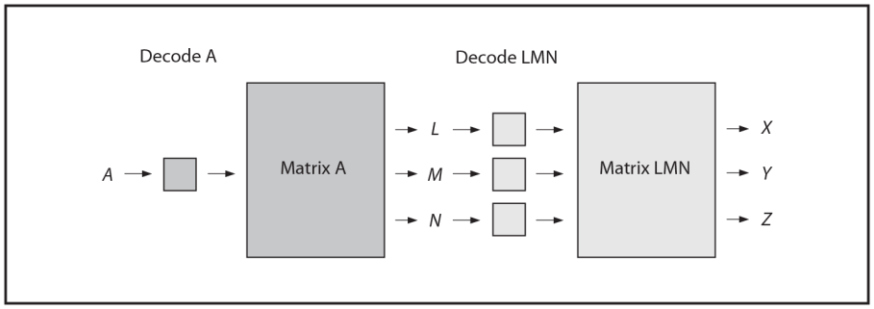


Figure 25 — Component transformations in a CIE-based A colour space

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; component values falling outside that range shall be adjusted to the nearest valid value without error indication. The decoding function (denoted by "Decode *A*" in "Figure 25 — Component transformations in a CIE-based A colour space") is a gamma function whose coefficient shall be specified by the **Gamma** entry in the colour space dictionary (see "Table 62 — Entries in a CalGray colour space dictionary"). 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 62 — 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

$$X = L = X_W \times A^G$$
$$Y = M = Y_W \times A^G$$
$$Z = N = Z_W \times A^G$$

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

the *X, Y*, and *Z* components of the final representation.

EXAMPLE 1 The examples in this subclause 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 1.00 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 1.00 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 24 — Component transformations in a CIE-based ABC colour space") are gamma functions whose coefficients shall be specified by the **Gamma** entry in the colour space dictionary (see "Table 63 — Entries in a CalRGB colour space dictionary"). The transformation matrix denoted by "Matrix *ABC*" in "Figure 24 — Component transformations in a CIE-based ABC colour space" 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 63 — 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_K Y_K Z_K] 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 subclause 10.3, "CIE-Based colour to device

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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

$$X = L = X_A \times A^{G_R} + X_B \times B^{G_G} + X_C \times C^{G_B}$$
$$Y = M = Y_A \times A^{G_R} + Y_B \times B^{G_G} + Y_C \times C^{G_B}$$
$$Z = N = Z_A \times A^{G_R} + Z_B \times B^{G_G} + Z_C \times C^{G_B}$$

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 1.00 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:

$$z = y_W \times \left((x_G - x_B) \times y_R - (x_R - x_B) \times y_G + (x_R - x_G) \times y_B\right)$$

$$Y_A = \frac{y_R}{R} \times \frac{(x_G - x_B) \times y_W - (x_W - x_B) \times y_G + (x_W - x_G) \times y_B}{z}$$

$$X_A = Y_A \times \frac{x_R}{y_R} \qquad Z_A = Y_A \times \left(\frac{1 - x_R}{y_R} - 1\right)$$

$$Y_B = \frac{y_G}{G} \times \frac{(x_R - x_B) \times y_W - (x_W - x_B) \times y_R + (x_W - x_R) \times y_B}{z}$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$$X_B = Y_B \times \frac{x_G}{y_G} \qquad Z_B = Y_B \times \left(\frac{1-x_G}{y_G} - 1\right)$$

$$Y_C = \frac{y_B}{B} \times \frac{(x_R - x_G) \times y_W - (x_W - x_G) \times y_R + (x_W - x_R) \times y_G}{z}$$

$$X_C = Y_C \times \frac{x_B}{y_B} \qquad Z_C = Y_C \times \left(\frac{1-x_B}{y_B} - 1\right)$$

$$X_W = X_A \times R + X_B \times G + X_C \times B$$
$$Y_W = Y_A \times R + Y_B \times G + Y_C \times B$$
$$Z_W = Z_A \times R + Z_B \times G + Z_C \times B$$

8.6.5.4 Lab colour spaces

A Lab colour space is a CIE-based *ABC* colour space with two transformation stages (see "Figure 24 — Component transformations in a CIE-based ABC colour space"). 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 64 — Entries in a Lab colour space dictionary"). Component values falling outside the specified range shall be adjusted to the nearest valid value without error indication.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 64 — 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]*

"Figure 26 — Lab colour space" illustrates the coordinates of a typical **Lab** colour space; "Figure 27 —

© ISO 2020 – All rights reserved

Colour gamuts" compares the gamuts (ranges of representable colours) for *L*a*b**, *RGB*, and *CMYK* spaces.

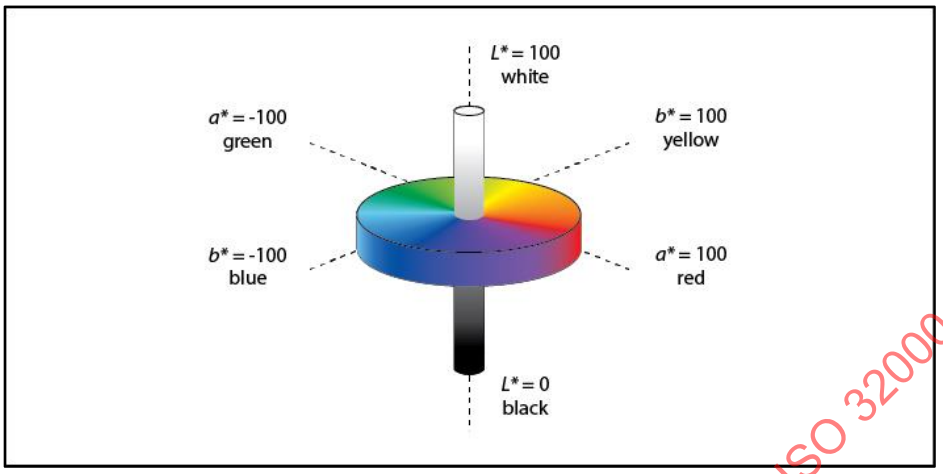


Figure 26 — Lab colour space

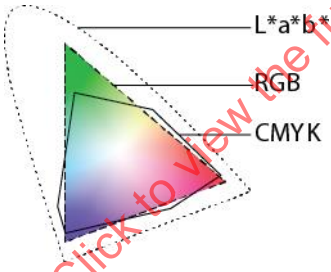


Figure 27 — Colour gamuts

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 24 — Component transformations in a CIE-based ABC colour space"). 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

$$g(x) = x^3 \qquad \text{if } x \geq \frac{6}{29}$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$$g(x) = \frac{108}{841} \times \left(x - \frac{4}{29}\right) \qquad \text{otherwise}$$

EXAMPLE The following defines the CIE 1976 L*a*b* space with the CCIR XA/11–recommended D65 white point (see ITU Recommendation BT.709). The a* and b* components, although theoretically unbounded, are defined to lie in the useful range -128 to +127.

```
[/Lab
        <</WhitePoint [0.9505 1.00 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). Unlike the **CalGray**, **CalRGB**, and **Lab** colour spaces, which are characterised by entries in the colour space dictionary, an **ICCBased** colour space shall be characterised by a sequence of bytes in a standard format. Details of the profile format can be found in the ICC specification.

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 — Entries common to all stream dictionaries"), the profile stream shall have the additional entries listed in "Table 65 — Additional entries specific to an ICC profile stream dictionary".

Table 65 — 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. Valid values for **N**: *1*, *3*, or *4*.
Alternate	name or array	(*Optional*) An alternate colour space that shall be used in case the one specified in the stream data is not supported. PDF readers should not 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 PDF 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 and after constraining to the ICCBased range, the nearest values within the range of the alternate space shall be substituted without error indication.
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").

"Table 66 — ICC Specification versions supported by ICC based colour spaces" 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.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 66 — 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.1:2010 (ISO 15076-1:2010)
2.0	ICC.1:2010 (ISO 15076-1:2010)

PDF processors shall follow these guidelines for writing and rendering ICC based color spaces:

- A PDF reader shall support ICC.1:2010 as required by PDF 2.0, which will enable it to properly render all embedded ICC profiles regardless of the PDF version.
- A PDF reader shall always process an embedded ICC profile according to the corresponding version of the PDF being processed as shown in "Table 66 — ICC Specification versions supported by ICC based colour spaces" above; it shall not substitute the alternate colour space in these cases.
- A PDF writer should use ICC 1:2010 profiles. It may embed profiles conforming to an earlier or later ICC version.
- A PDF processor shall substitute the alternate colour space for embedded profiles conforming to later ICC versions, if the PDF processor is not capable of properly processing the embedded ICC profile.
- PDF writers shall only use the profile types shown in "Table 67 — ICC profile types" for specifying calibrated colour spaces for colouring graphics 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.
- Profiles shall conform to the specification version indicated by the Profile version number in its header.

NOTE 1 *XYZ* and 16-bit L*a*b* profiles are not listed.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 67 — 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 ')

© ISO 2020 – All rights reserved

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 68 — Ranges for typical ICC colour spaces".

Table 68 — 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]

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 PDF 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; 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 IEC 61966-2-1 ed1.0 (1999-10) Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB (with Amendment 1 IEC 61966-2-1-am1 ed1.0 (2003-01)). 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** can be used.

```
10 0 obj                                        %Colour space
      [/ICCBased 15 0 R]
endobj

15 0 obj                                        %ICC profile stream
```

© ISO 2020 – All rights reserved

```
          <</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 PDF 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 older PDF readers that do not support CIE-based colours.
- 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.

NOTE (2020) This remapping means that the current colour space is defined by the default colour space rather than **DeviceGray**, **DeviceRGB** or **DeviceCMYK**. Provisions in this standard that apply specifically to device colour spaces are then not applicable to graphic objects painted when the default colour space is not one of **DeviceGray**, **DeviceRGB** or **DeviceCMYK**.

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 PDF processor 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 24 — Component transformations in a CIE-based ABC colour space".

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 components and back is unnecessary, and results in a loss of fidelity in the black component. In such cases, a PDF processor could 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, can suggest such a calibration; "see 14.11.5, "Output intents"). The conversion is completely hidden by the PDF processor 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 non-zero 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 need 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 PDF writer to set priorities regarding which of these properties to preserve and which to sacrifice.

EXAMPLE The PDF 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 colorimetric 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 69 — Rendering intents" lists the standard rendering intents that shall be recognised.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

Table 69 — 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. Logos and solid colours are typical cases requiring 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. Vector graphics are a typical use case.
Saturation	Colours shall be represented in a manner that preserves or emphasizes saturation. Reproduction of in-gamut colours may or may not be colorimetrically accurate. NOTE 3 Business graphics are a typical use case 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 colorimetric values. NOTE 4 Scanned images are a typical use case.

"Figure 28 — Rendering intents" illustrates the effects of the standard rendering intents. These intents have been chosen to correspond to those defined by the International Color Consortium (ICC), an industry organisation that has developed standards for device-independent colour. If a PDF processor does not recognise the specified name, it shall use the **RelativeColorimetric** intent by default.

NOTE The exact set of rendering intents supported can vary from one output device to another; a particular device does not have to support all PDF rendering intents and can support additional ones beyond those listed in the table above.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Figure 28 — Rendering intents

See 11.7.5, "Rendering parameters and transparency", and in particular 11.7.5.3, "Rendering intent, black point compensation and colour conversions", for further discussion of the role of rendering intents in the transparent imaging model.

8.6.5.9 Use of black point compensation

Black point compensation applies to CIE-based colour conversion and extends the concept of the use of rendering intents for colour conversion based upon the ICC architecture.

The use of black point compensation can be controlled through the **UseBlackPtComp** entry in the **ExtGState** dictionary. If the value for **UseBlackPtComp** is *ON*, colour conversion shall be carried out according to the provisions in ISO 18619. If it is set to *OFF* no black point compensation shall be carried out. If the value is not given or set to *Default*, then the behaviour is left to the PDF processor to determine. If the current render intent of an object is **AbsColorimetric** then the value of **UseBlackPtComp** shall be treated as *OFF*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 a *colour map* or *colour table* of arbitrary colours in some other space. A PDF 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 initialise 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 $m \times (hival + 1)$ 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 2 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 will be 256 colour values and the string will be 768 bytes long. Having established this colour space, the PDF file 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 3 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 (0.5 values shall be rounded up); 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 colourants 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 colourant or colour components only.

NOTE 1 Colour output devices produce full colour by combining *primary* or *process colourants* in varying amounts. On an additive colour device such as a display, the primary colourants 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 colourants, often called *spot colourants*, to produce effects that cannot be achieved with the standard process colourants 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 colourants (and spot colourants, if any) are combined. However, some devices, such as imagesetters, produce a separate, monochromatic rendition of the page, called a *separation*, for each colourant. When the separations are later combined — on a printing press, for example — and the proper inks or other colourants 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 colourant. In the context of this discussion, a printing system that produces separations generates a separate piece of physical medium (generally film) for each colourant. It is these pieces of physical

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

medium that are correctly referred to as separations. A particular colourant properly constitutes a separation only if the device is generating physical separations, one of which corresponds to the given colourant. The **Separation** colour space is so named for historical reasons, but it has evolved to the broader purpose of controlling the application of individual colourants in general, regardless of whether they are actually realised 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 colour space does not have to correspond to a true, physical separation or to an actual colourant. For example, a **Separation** colour space could be used to control the application of a single process colourant (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 colourant 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 colourant 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 colourant, 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 colourant that this **Separation** colour space is intended to represent (or one of the special names **All** or **None**; see below). With the exception of the names **Cyan**, **Magenta**, **Yellow** and **Black** which are reserved to name the process colourants of a CMYK device, such colourant names are arbitrary, and there may be any number of them, subject to implementation limits.

The special colourant name **All** shall refer collectively to all colourants available on an output device, including those for the standard process colourants. When a **Separation** space with this colourant name is the current colour space, painting operators shall apply tint values to all available colourants at once. When outputting to an additive device, such as a computer monitor, the subtractive tint values of the **All** colourant shall be complemented by subtracting from 1 before applying to all available colourants.

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 colourant name **None** shall not produce any visible output. Painting operations in a **Separation** space with this colourant name shall have no effect on the current page.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

A PDF processor shall support **Separation** colour spaces with the colourant 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 colourant names PDF processors 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 PDF reader shall determine whether the device has an available colourant corresponding to the name of the requested space. If so, the PDF processor shall ignore the *alternateSpace* and *tintTransform* parameters; subsequent painting operations within the space shall apply the designated colourant 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 colourant directly; it always reverts to the alternate colour space as described below. This is because the model of applying process colourants independently does not work as intended on an additive device.

EXAMPLE 1 In an R, G, B colour space, painting tints of the ***Red*** component on a white background (1,1,1) produces a result that varies from white to cyan (0,1,1) which is not as might be otherwise expected for a red component.

This exception applies only to colourants for additive devices, not to any specific names, e.g., **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 colourant.

If the colourant name associated with a **Separation** colour space does not correspond to a colourant available on the device, the PDF processor 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 using the usual primary or process colourants:

- 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 PDF processor 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 In some cases where colourants are unavailable on the output device, painting in the alternate colour space can produce a good approximation of the intended colour when only opaque objects are painted. However, it does not necessarily reflect the interactions between an object and its backdrop when overprinting (see 8.6.7, "Overprint control") is enabled. Separation simulation (see 10.8.3) can be used as an alternative method to yield better results when overprinting is involved. When transparency is involved, the use of the alternate space can produce incorrect output regardless of what method is used.

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 colourant corresponding to this colour, **DeviceCMYK** is used as the alternate colour space, and the tint transformation function (object 12) maps tint values

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 available 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 colourants to produce an extended gamut, or range of colours. A popular example is the PANTONE Hexachrome system, which uses six colourants: 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 can be painted with the exact sample data from the single-component image; the spot colour component can be generated as a nonlinear function of the image data in a manner that emphasizes the shadows. "Figure 29 — Duotone image" shows an example that uses black and magenta colour components. In "Figure 30 — Quadtone image" a single-component grayscale image is used to generate a quadtone result that uses four colourants: black and three PANTONE spot colours. See Example 4 in this subclause for the code used to generate this image.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

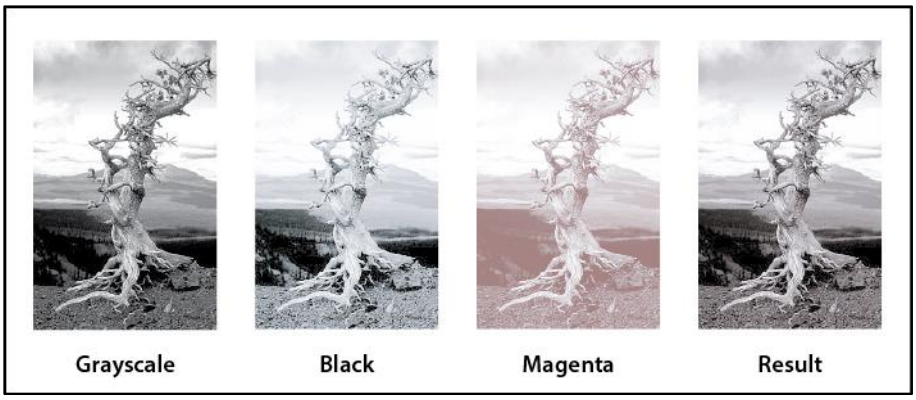


Figure 29 — Duotone image

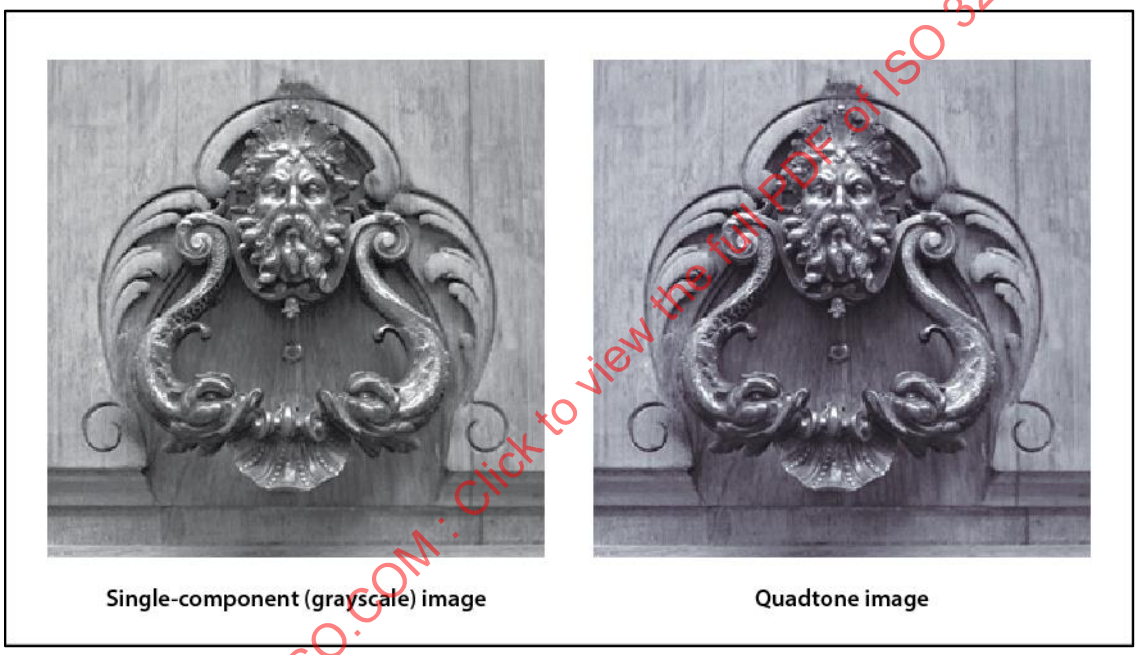


Figure 30 — Quadtone image

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

DeviceN shall be used to represent colour spaces containing multiple components that correspond to colourants of some target device. As with **Separation** colour spaces, PDF processors shall be able to approximate the colourants 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 colourants (the process components shall be from a single process colour space). They provide information about each component that allows PDF processors 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 70 — Entries in a DeviceN colour space attributes dictionary"). A value of **DeviceN** for the **Subtype** entry, or no value, shall mean that only the previous features shall be supported. PDF processors that do not support PDF 1.6 shall

© ISO 2020 – All rights reserved

treat these colour spaces as normal **DeviceN** colour spaces and shall use the tint transformation function as appropriate. PDF writers using the **NChannel** features should follow certain guidelines, as noted throughout this subclause, to achieve good backward compatibility.

NOTE 3 PDF processors can use their own blending algorithms for on-screen viewing and composite printing, rather than being required to use a specified tint transformation function. See also clause 10.8, "Rendering for separations".

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 maximum number of entries in the names array in the computer on which the PDF processor is running may be subject to implementation limits; see Annex C, "Advice on maximising portability".

The component names shall all be different from one another, except for the name **None**, which may be repeated as described later in this subclause. The special name **All**, used by **Separation** colour spaces, shall not be used. The names **Cyan**, **Magenta**, **Yellow** and **Black** are reserved to name the subtractive process colourants of a CMYK device.

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 colourant; 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 colourant, and 1.0 the darkest.

NOTE 4 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 colourant available on the device, the PDF processor should perform subsequent painting operations in the alternate colour space specified by this parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 5 In some cases PDF processors have more information about colourants and their interaction than is provided through the *alternateSpace* parameter, and are free to use such information instead of the *alternateSpace* parameter. In addition, where a **DeviceN** space contains an attributes dictionary, PDF processors are free to use the information provided in the attributes dictionary instead of the *alternateSpace* 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 of that component.

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 6 Painting in the alternate colour space can 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** colourant. When a **DeviceN** colour space is painting the named device colourants directly, colour components corresponding to **None** colourants 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 colourant 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 70 — Entries in a DeviceN colour space attributes dictionary") containing additional information about the components of this colour space that PDF processors may use. PDF processors 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 PDF processors that want to use them or do not support PDF 1.6.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 70 — 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*.

© ISO 2020 – All rights reserved

Key	Type	Value
Colorants	dictionary	(*Required if **Subtype** is **NChannel** and the colour space includes spot colourants; otherwise optional; PDF 1.6*) A dictionary describing the individual colourants used in the **DeviceN** colour space. For each entry in this dictionary, the key shall be a colourant name and the value shall be an array defining a **Separation** colour space for that colourant (see 8.6.6.4, "Separation colour spaces"). The key shall match the colourant name given in that colour space. This dictionary provides information about the individual colourants that may be useful to some PDF processors. In particular, the alternate colour space and tint transformation function of a **Separation** colour space describe the appearance of that colourant alone, whereas those of a **DeviceN** colour space describe only the appearance of its colourants in combination.
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 71 — Entries in a DeviceN process dictionary") that describes the process colour space whose components are included in this colour space.
MixingHints	dictionary	(*Optional; PDF 1.6*) A dictionary (see "Table 72 — Entries in a DeviceN mixing hints dictionary") that specifies optional attributes of the inks that shall be used in blending calculations when used as an alternative to the tint transformation function.

This dictionary provides information about the individual colourants that may be useful to some PDF processors. In particular, the alternate colour space and tint transformation function of a **Separation** colour space describe the appearance of that colourant alone, whereas those of a **DeviceN** colour space describe only the appearance of its colourants in combination.

If **Subtype** is **NChannel**, the **Colorants** dictionary shall have entries for all spot colourants in this colour space. The **Colorants** dictionary may also include additional colourants not used by this colour space.

A value of **NChannel** for the **Subtype** entry indicates that some of the other entries in the **Colorants** dictionary are required rather than optional. The **Colorants** entry specifies a **Colorants** dictionary that contains entries for all the spot colourants in the colour space; they shall be defined using individual **Separation** colour spaces. The **Process** entry specifies a process dictionary (see "Table 71 — Entries in a DeviceN process dictionary") 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 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

(see "Table 71 — Entries in a DeviceN process dictionary" and discussion below), which shall be present.

- Definitions for process colourants should not appear in the **Colorants** dictionary. Any such definition shall be ignored if the colourant is also present in the process dictionary. Any component not specified in the process dictionary shall be considered to be a spot colourant.
- 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 colourants 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 72 — Entries in a DeviceN mixing hints dictionary") that provides information about the characteristics of colourants that may be used in blending calculations when the actual colourants are not available on the target device. PDF processors need not use this information.

Table 71 — 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 except **Lab**. 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 72 — 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 colourant name, and the value shall be a number between 0.0 and 1.0. This dictionary need not contain entries for all colourants used in this colour space; it may also include additional colourants 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 colourants shall be 0.0; interactive PDF processors 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 colourant 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 colourants unused by this specific **DeviceN** instance. NOTE (2020) **PrintingOrder** precisely matches the optional ICC profile *colorantOrderTag* (ISO 15076-1, 9.2.17), which specifies physical colourant laydown relative to the substrate. It does not define viewing direction.
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 colourant 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 colourants unused by this specific **DeviceN** instance and need not list all colourants. An entry with a key of **Default** shall specify a function that shall be used by all colourants for which a dot gain function is not explicitly specified. PDF processors 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 colourants.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Each entry in the mixing hints dictionary refers to colourant names, which include spot colourants 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 7 The mixing hints subdictionaries (as well as the **Colorants** dictionary) can specify colourants 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, the following guidelines apply:

- The PDF processor should apply either the specified tint transformation function or invoke the same alternative blending algorithm for all **DeviceN** instances in the document.

© ISO 2020 – All rights reserved

NOTE 8 When the tint transformation function is used, the burden is on the PDF 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 2 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). *tintTransform*1, whose definition is not shown, maps three colour components (tints of the colourants **Orange**, **Green**, and **None**) to four colour components in the alternate colour space, **DeviceCMYK**. *tintTransform*2 maps a single colour component (an orange tint) to four components in **DeviceCMYK**. Likewise, *tintTransform*3 maps a green tint to **DeviceCMYK**, and *tintTransform*4 maps a tint of PANTONE 131 to **DeviceCMYK**.

```
30 0 obj                                                %Colour space
      [/DeviceN
            [/Orange /Green /None]
            /DeviceCMYK
            tintTransform1
             <</Colorants 45 0 R>>
      ]
endobj

45 0 obj                                                %Colorants dictionary
      <</Orange [/Separation
                 /Orange
                 /DeviceCMYK
                 tintTransform2
                 ]
         /Green [/Separation
                 /Green
                 /DeviceCMYK
                 tintTransform3
                ]
         /PANTONE#20131 [/Separation
                         /PANTONE#20131
                         /DeviceCMYK
                         tintTransform4
                         ]
      >>
endobj
```

NOTE 9 Example 3, Example 4 and Example 5 show the use of **NChannel** colour spaces.

EXAMPLE 3 This example shows the use of calibrated CMYK 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

EXAMPLE 4 This example shows the recommended convention for dealing with situations where a spot colourant and a process colour component have the same name. Since the *names* array cannot have duplicate names, the process colours will need to 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 colourant; **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]>>
         >>
      ]
endobj
```

EXAMPLE 5 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]
               >>
         >>
         ]
endobj
```

See 11.7.3, "Spot colours and transparency", for further discussion of the role of **DeviceN** colour spaces in the transparent imaging model.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause 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 colourants. 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 ob                                          %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 colourants (where gold is a spot colourant) 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 available 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 subclause, each entry in the lookup table has five components: two for the black and gold colourants and three more (specified as **None**) for the equivalent **CalRGB** colour components. If the black and gold colourants 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 subclause).

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 30 — Quadtone image").

```
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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 colourants (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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 colourants: 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 colourants 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

© ISO 2020 – All rights reserved

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 subclause 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 colourants, depending on the colour space in which the painting takes place. In a **Separation** or **DeviceN** colour space, the colourants 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 colourants 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 colourants 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 colourants is left undisturbed. Consequently, the colour at a given position on the page may be a combined result of several painting operations in different colourants. The effect produced by such overprinting is device-dependent and is not defined here.

NOTE 1 Not all devices support overprinting. Furthermore, many PostScript language compatible 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. Determination of whether a tint value is zero or non-zero shall be made on the tint value defined within the PDF file, before quantisation into a device tint value for the output device.

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 colourant directly. Ordinarily, each source colour component value replaces the value previously painted for the corresponding device colourant, 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 *non-zero 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 non-zero.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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
```

Non-zero 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, however, apply to the painting of images or shadings (8.7.4, "Shading patterns"). It also shall not apply if the native colour space of the output device does not include CMYK device colourants; 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 8.6.6.5, "DeviceN 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 73 — Colour operators" lists the PDF operators that control colour spaces and colour values. Also colour-related is the graphics state operator **ri**, listed in "Table 56 — Graphics state operators" 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 — Graphics objects").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 73 — 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 colourants. 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.
$c_1 \ldots c_n$	**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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Operands	Operator	Description
*c*1... *c*n *c*1... *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 9.6.4, "Type 3 fonts") and to all other content streams invoked from within the same glyph description.
- In the content stream of an uncoloured tiling pattern (see 8.7.3.3, "Uncoloured tiling patterns") and to all other content streams invoked from within the uncoloured tiling pattern stream.

In these circumstances:

- All of the following operators shall be ignored:

CS	**scn**	**K**
cs	**G**	**k**
SC	**g**	**ri**
SCN	**RG**	**sh**
sc	**rg**	

- All of the following entries, if present in the graphics state parameter dictionary of a **gs** operator shall be ignored:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

TR	**BG**	**UCR**
TR2	**BG2**	**UCR2**
HT	**UseBlackPtComp**	

- Unless painting an image mask, all image painting operators shall be ignored.

NOTE 1 Painting an image mask (see 8.9.6.2, "Stencil masking") is permitted because it does not specify colours; instead, it designates places where the current colour is painted.

NOTE 2 (2020) Prior PDF specifications at this location stated that these circumstances would cause an error, but elsewhere also stated that these same circumstances would be ignored. In PDF 2.0, the requirement stated here has changed to ignore to be consistent throughout this document.

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, "paint" can be applied 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 are not recommended. Another technique, defining patterns as halftone screens, is also not recommended 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 subclause 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 subclause 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclauses 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 73 — Colour operators").

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.

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 can be used within another pattern; the inner pattern's matrix defines its relationship to the pattern space of the outer pattern.

NOTE The PostScript language allows a pattern to be defined in one context but used in another. For example, a pattern can be defined on a page (that is, its pattern matrix maps the pattern coordinate space to the user space of the page) but is 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.

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 PDF processor 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 — Entries common to all stream dictionaries"), this stream's dictionary may contain the additional entries listed in "Table 74 — Additional entries specific to a Type 1 pattern dictionary".

Table 74 — 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.
PaintType	integer	(*Required*) A code that determines how the colour of the pattern cell shall be specified: 1 *Coloured tiling pattern.* The pattern's content stream shall specify the colours used to paint the pattern cell. The current colours in use when the PDF processor begins processing the content stream are the ones 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". 2 *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 nonstroking colour shall be poured. The content stream shall not invoke operators that specify colours or other colour-related parameters in the graphics state; otherwise, the operator is ignored and processing of the stream continues without error (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: 1 *Constant spacing*. Pattern cells shall be spaced consistently — that is, by a multiple of a device pixel. To achieve this, the PDF processor 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. 2 *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. 3 *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. NOTE 1 A BBox of zero height or width will still paint one pixel (see 10.7.4, "Scan conversion rules").
XStep	number	(*Required*) The desired horizontal spacing between pattern cells, measured in the pattern coordinate system.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
YStep	number	(*Required*) The desired vertical spacing between pattern cells, measured in the pattern coordinate system. NOTE 2 **XStep** and **YStep** can 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 PDF 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 PDF processor 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)

The pattern's content stream shall not set any of the device-dependent parameters in the graphics state (see "Table 52 — Device-dependent graphics state parameters") because it can result in incorrect output.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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

© ISO 2020 – All rights reserved

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 31 — Coloured tiling pattern" shows the pattern cell on the left side and the patterned shapes on the right side.

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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
         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 87.90 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 subclause are noteworthy:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.

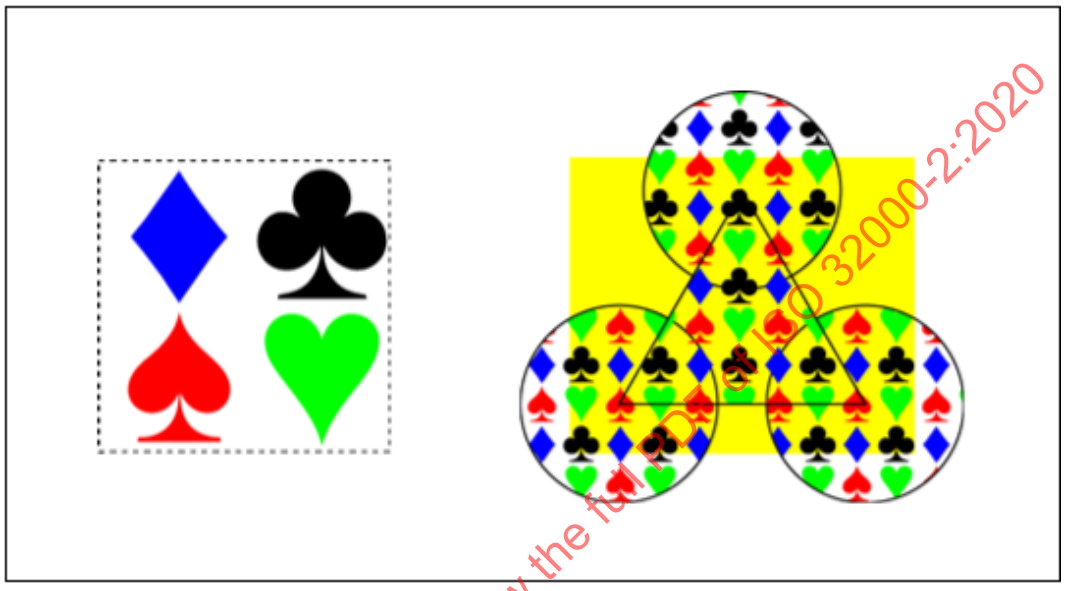


Figure 31 — Coloured tiling pattern

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 (see 8.6.8, "Colour operators"); 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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, "Coloured tiling patterns", except that it uses an uncoloured tiling pattern to paint the three circles and the triangle, each in a different colour (see "Figure 32 — Uncoloured tiling pattern". 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
      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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

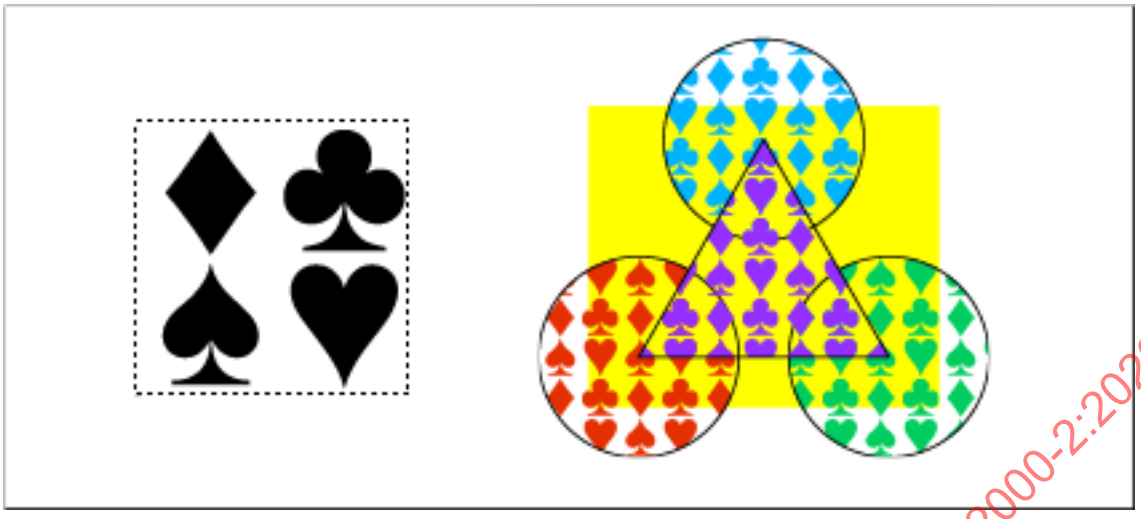


Figure 32 — Uncoloured tiling pattern

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 75 — Entries in a Type 2 pattern dictionary" shows the contents of this type of dictionary.

Table 75 — 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 77 — Entries common to all shading dictionaries" and those in one of Table 78 to Table 83.
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 not so specified shall be inherited from the graphics state that was in effect at the beginning of the pattern's parent content stream, and as modified by clause 11.6.7, "Patterns and transparency".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause 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 76 — Shading operator" 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 76 — 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 77 — Entries common to all shading dictionaries"). 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 77 — Entries common to all shading dictionaries" shows the entries that all shading dictionaries share in common; entries specific to particular shading types are described in the relevant subclause.

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 77 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 1 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. The background colour shall be applied only when the shading is used as part of a shading pattern, not when 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. NOTE 2 A BBox of zero height or width will still paint one pixel (see 10.7.4, "Scan conversion rules").
AntiAlias	boolean	(*Optional*) A flag indicating whether to filter the shading function to prevent aliasing artifacts. NOTE 3 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. Default value: *false*.

Shading types 4 to 7 shall be defined by a stream containing descriptive data characterising 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 — Entries common to all stream dictionaries"). In particular, the stream dictionary 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, can exhibit aliasing effects when painted at low effective resolutions.

8.7.4.4 Colour space: special considerations

Conceptually, a shading determines a colour value for each individual point within the area to be painted. In practice, however, PDF processors may actually 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. PDF processors 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 10.7.3, "Smoothness tolerance"). The **ColorSpace** entry

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause. 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, can 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 10.4**,** "Conversions among device colour spaces". To optimise 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.

NOTE 3: This does not apply to shadings having a **Function** entry in their shading dictionary because those shading 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 colourants 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 an optimal 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 quantised 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 77 — Entries common to all shading dictionaries", all shading dictionaries have entries specific to the type of shading they represent, as indicated by the value of

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

their **ShadingType** entry. The following subclauses 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 78 — Additional entries specific to a Type 1 shading dictionary" shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries").

This type of shading shall not be used with an **Indexed** colour space.

Table 78 — 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 1 0 1]*.
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 1 0 1] 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 or array	(*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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

or both endpoints by continuing the boundary colours indefinitely. "Table 79 — Additional entries specific to a Type 2 shading dictionary" shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries").

This type of shading shall not be used with an **Indexed** colour space.

Table 79 — 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. If the starting and ending coordinates are coincident (x_0=x_1 and y_0=y_1) 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 points of the axis. The variable *t* becomes the input argument to the colour function(s). Default value: [0.0 1.0].
Function	function or array	(*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*].

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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:

$$x' = \frac{(x_1 - x_0) \times (x - x_0) + (y_1 - y_0) \times (y - y_0)}{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

For $0 \le x' \le 1, t = t_0 + (t_1 - t_0) \times x'$.

The value of the parametric variable t is then determined from x' as follows:

- For $0 \le x' \le 1, t = t_0 + (t_1 - t_0) \times x'$.
- 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.

© ISO 2020 – All rights reserved

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 33 — Axial shading" 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 not 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.

Figure 33 — Axial shading

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 80 — Additional entries specific to a Type 3 shading dictionary", as well as those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries").

This type of shading shall not be used with an **Indexed** colour space.

© ISO 2020 – All rights reserved

Table 80 — 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 1]*.
Function	function or array	(*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*].

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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. The appearance of the shading shall be as if an infinite number of such circles are painted in turn, each with an infinitely narrow stroke.

$$s = \frac{t - t_0}{t_1 - t_0}$$

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:

$$x_c(s) = x_0 + s \times (x_1 - x_0)$$
$$y_c(s) = y_0 + s \times (y_1 - y_0)$$
$$r(s) = r_0 + s \times (r_1 - r_0)$$

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 paint 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 unless radii r_0 and r_1 in the **Coords** array are both zero.

© ISO 2020 – All rights reserved

NOTE 1 Either of the starting and ending circles can 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 34 — Radial shadings depicting a cone" depict the results of extending the shading at the smaller and larger ends, respectively.

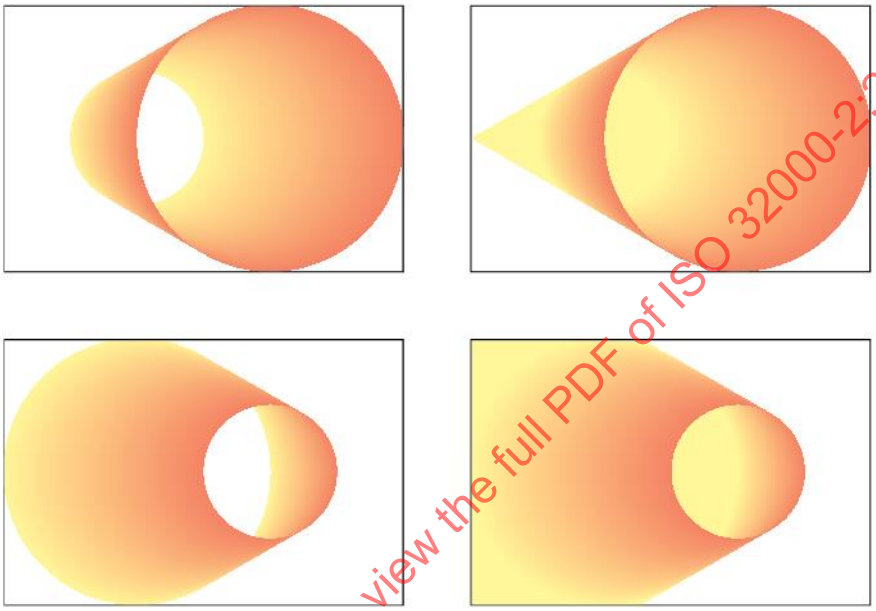


Figure 34 — Radial shadings depicting a cone

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 on 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 35 — Radial shadings depicting a sphere" and "Figure 36 — Radial shadings with extension". In "Figure 35 — Radial shadings depicting a sphere", 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 36 — Radial shadings with extension", the inner circle in both figures has a non-zero 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

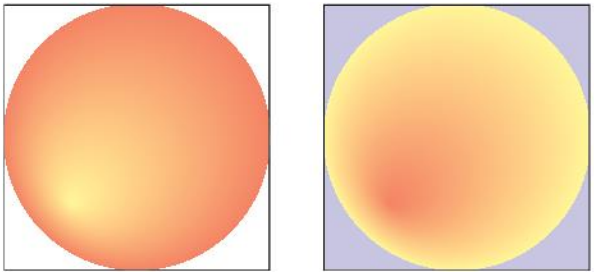


Figure 35 — Radial shadings depicting a sphere

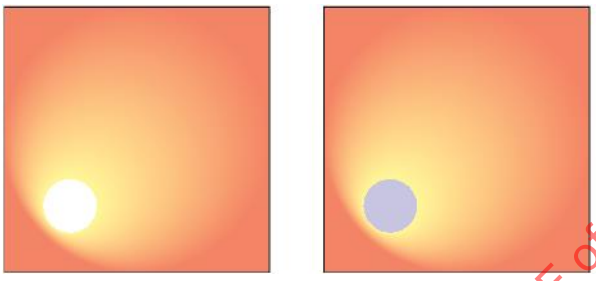


Figure 36 — Radial shadings with extension

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 34 — Radial shadings depicting a cone").

EXAMPLE 1 This example shows the shading used for the objects in the leaf-covered branch in "Figure 37 — Radial shading effect" (8.7.4.5.4, "Type 3 (radial) shadings"). 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       1.0]         %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 1.0 0.298]
         /C1 [0.631 0.278 1.0 0.027]
         /N 1.048
      >>
endobj

12 0 obj                                                   %Second subfunction
       <</FunctionType 2
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            /Domain [0.0 1.0]
            /C0 [0.929 0.357 1.0 0.298]
            /C1 [0.941 0.400 1.0 0.102]
            /N 1.374
         >>
endobj
```

EXAMPLE 2 This example shows how each leaf shown in "Figure 37 — Radial shading effect" 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 0.00 0.00 -27.7843 310.2461 121.1521 cm            %Set matrix
/Sh1 sh                                                    %Paint shading
Q                                                          %Restore graphics state
```

Figure 37 — Radial shading effect

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

8.7.4.5.5 Type 4 (free-form Gouraud-shaded triangle mesh) shadings

Type 4 (free-form Gouraud-shaded triangle mesh) shadings 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. "Table 81 — Additional entries specific to a Type 4 shading dictionary" shows the entries specific to this type of shading dictionary, in addition to those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries") and stream dictionaries (see "Table 5 — Entries common to all stream dictionaries").

Table 81 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
BitsPerFlag	integer	(*Required*) The number of bits used to represent the edge flag for each vertex (see below). The value 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} $c_{1,min}$ $c_{1,max}$... $c_{n,min}$ $c_{n,max}$] Only one pair of *c* values shall be specified if a **Function** entry is present.
Function	function or array	(*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:

$f\ x\ y\ c_1 \ldots\ c_n$

where

f is the vertex's edge flag (discussed below)

x and *y* are its horizontal and vertical coordinates

$c_1 \ldots\ c_n$ 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 $c_1 \ldots\ c_n$.

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 38 — Starting a new triangle in a free-form Gouraud-shaded triangle mesh".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

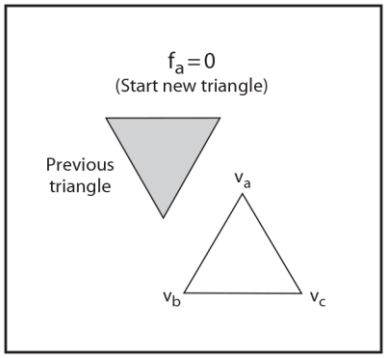


Figure 38 — Starting a new triangle in a free-form Gouraud-shaded triangle mesh

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 39 — Connecting triangles in a free-form Gouraud-shaded triangle mesh". (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.

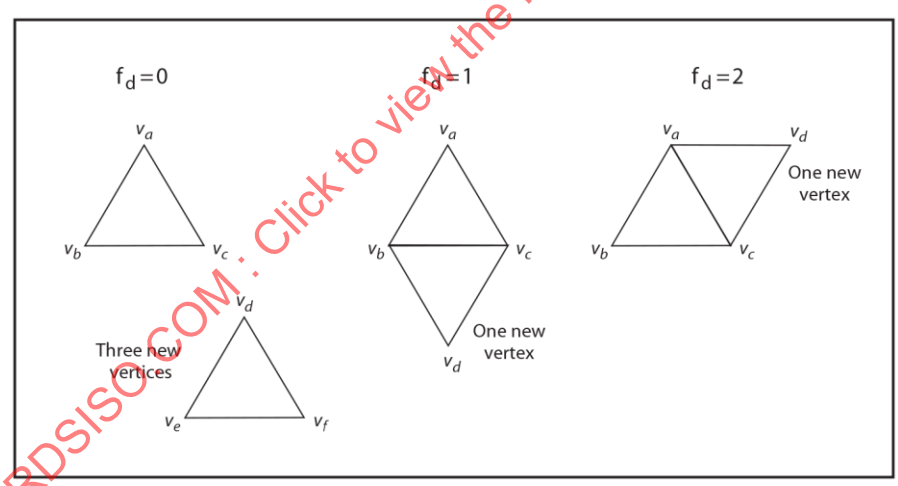


Figure 39 — Connecting triangles in a free-form Gouraud-shaded triangle mesh

Complex shapes can be created by using the edge flags to control the edge on which subsequent triangles are formed.

EXAMPLE "Figure 40 — Varying the value of the edge flag to create different shapes" shows two simple examples. Mesh1 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 ($f_m = 1$)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

6 ($f_h = 1$)

Mesh 2 again begins with triangle 1 and uses the following edge flags:

1 ($f_a = f_b = f_c = 0$)
2 ($f_d = 1$)
3 ($f_e = 2$)
4 ($f_f = 2$)
5 ($f_g = 2$)
6 ($f_h = 2$)

The stream shall provide vertex data for a whole number of triangles with appropriate edge flags; otherwise, an error occurs.

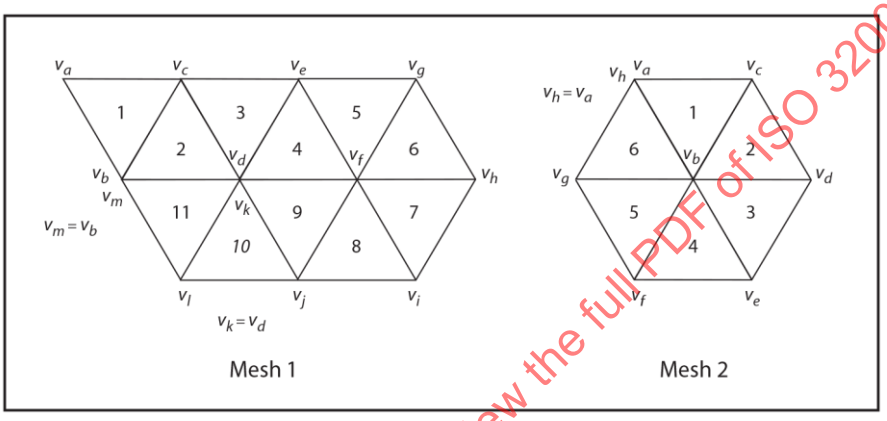


Figure 40 — Varying the value of the edge flag to create different shapes

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 (lattice-form Gouraud-shaded triangle mesh) shadings

Type 5 (lattice-form Gouraud-shaded triangle mesh) shadings 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 organised into rows, which need not be

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

geometrically linear (see "Figure 41 — Lattice-form triangle meshes").

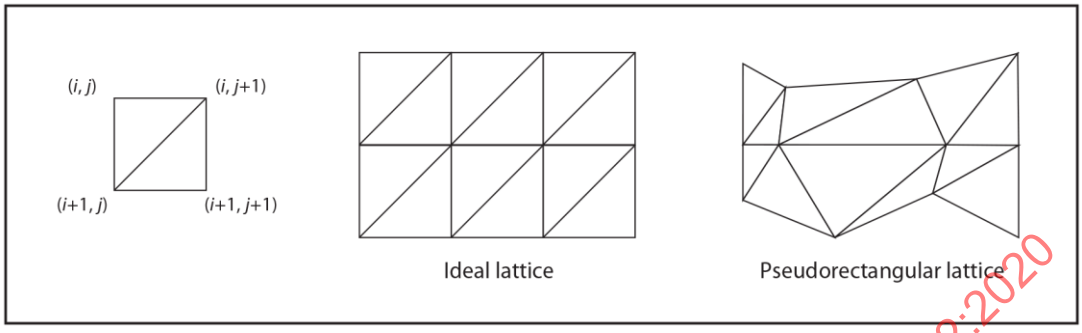


Figure 41 — Lattice-form triangle meshes

"Table 82 — Additional entries specific to a Type 5 shading dictionary" shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries") and stream dictionaries (see "Table 5 — Entries common to all stream dictionaries").

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 82 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Function	function or array	(*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 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 $c_1 \ldots c_n$.

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 41 — Lattice-form triangle meshes":

$$(V_{i,j},\ V_{i,j+1}, V_{i+1,j}) \qquad \text{for } 0 \le i \le m-2, 0 \le j \le k-2$$
$$(V_{i,j+1},\ V_{i+1,j}, V_{i+1,j+1})$$

See 8.7.4.5.5, "Type 4 (free-form Gouraud-shaded triangle mesh) shadings" for further details on the format of the vertex data.

8.7.4.5.7 Type 6 (Coons patch mesh) shadings

Type 6 (Coons patch mesh) shadings 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 42 — Coons patch mesh").
- Coordinates are mapped from the unit square into a four-sided patch whose sides are not necessarily linear (see the lower figure in "Figure 42 — Coons patch mesh". 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 43 — Coordinate mapping from a unit square to a four-sided Coons Patch".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

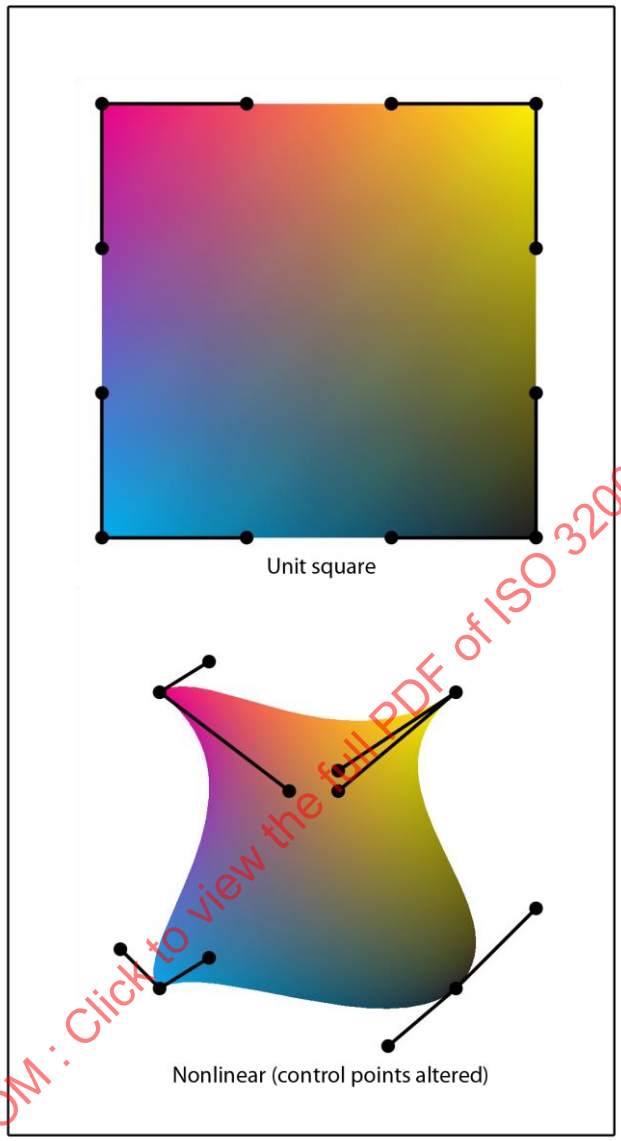


Figure 42 — Coons patch mesh

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:

$$C_1(0) = D_1(0)$$
$$C_1(1) = D_2(0)$$
$$C_2(0) = D_1(1)$$
$$C_2(1) = D_2(1)$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

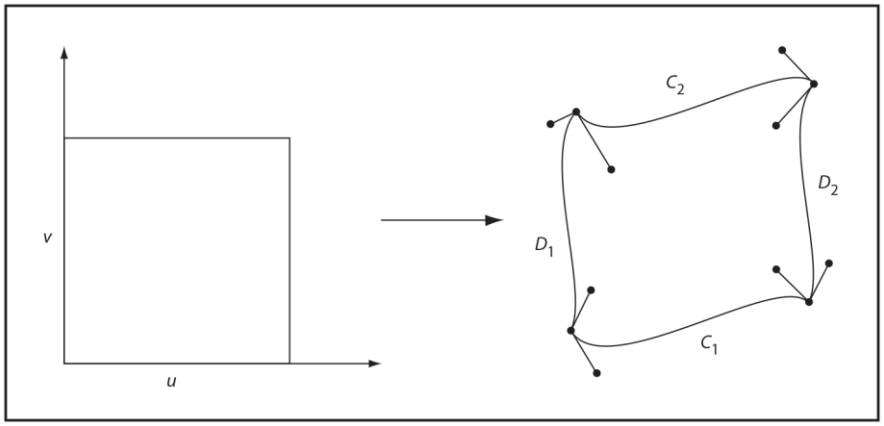


Figure 43 — Coordinate mapping from a unit square to a four-sided Coons Patch

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:

$$S_B(u,v) = (1-v) \times [(1-u) \times C_1(0) + u \times C_1(1)] + v \times [(1-u) \times C_2(0) + u \times C_2(1)]$$

The coordinate mapping for the shading is 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) need not be the same as the patch boundary if, for example, the patch folds over on itself, as shown in "Figure 44 — Painted area and boundary of a Coons Patch".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

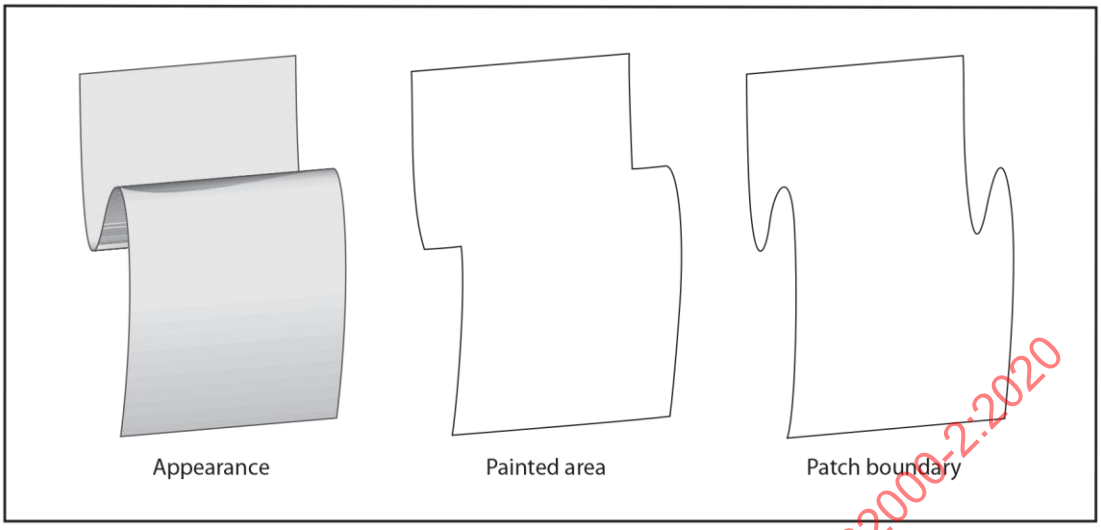


Figure 44 — Painted area and boundary of a Coons Patch

"Table 83 — Additional entries specific to a Type 6 shading dictionary" shows the shading dictionary entries specific to this type of shading, in addition to those common to all shading dictionaries (see "Table 77 — Entries common to all shading dictionaries") and stream dictionaries (see "Table 5 — Entries common to all stream dictionaries").

Table 83 — 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: $[x_{min}\ x_{max}\ y_{min}\ y_{max}\ c_{1,min}\ c_{1,max} \ldots\ c_{n,min}\ c_{n,max}]$ Only one pair of *c* values shall be specified if a **Function** entry is present.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Function	function or array	(*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 the corner points of each patch 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 shall be 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 shall be expressed in the shading's target coordinate space. See 8.7.4.5.5, "Type 4 (free-form Gouraud-shaded triangle mesh) shadings" 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, $x_1 y_1 x_2 y_2 ... x_{12} y_{12}$, which specify the Bézier control points that define the four boundary curves. "Figure 45 — Colour values and edge flags in Coons Patch meshes" shows how these control points correspond to the cubic Bézier curves C_1, C_2, D_1, *and* D_2 identified in "Figure 43 — Coordinate mapping from a unit square to a four-sided Coons Patch". 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.

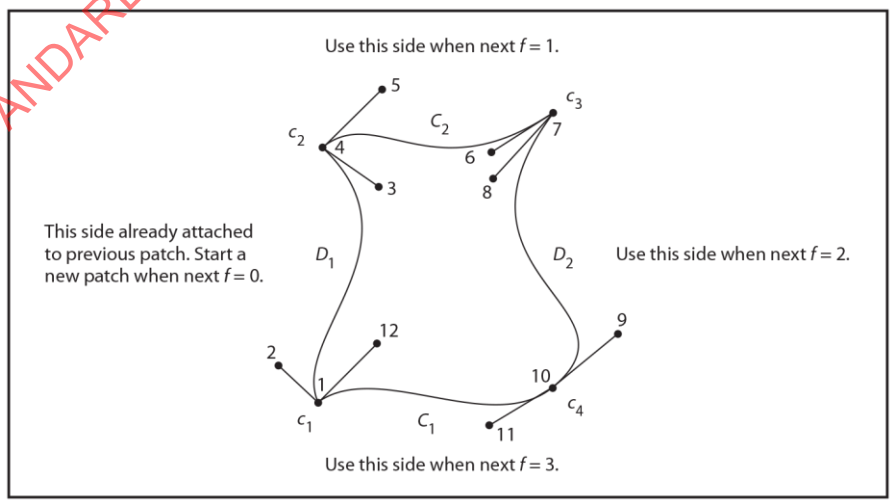


Figure 45 — Colour values and edge flags in Coons Patch meshes

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

"Figure 45 — Colour values and edge flags in Coons Patch meshes" also shows how non-zero 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 46 — Edge connections in a Coons Patch Mesh"), and therefore shall not be explicitly repeated in the data stream. "Table 84 — Data Values in a Coons Patch Mesh" summarises the required data values for various values of the edge flag.

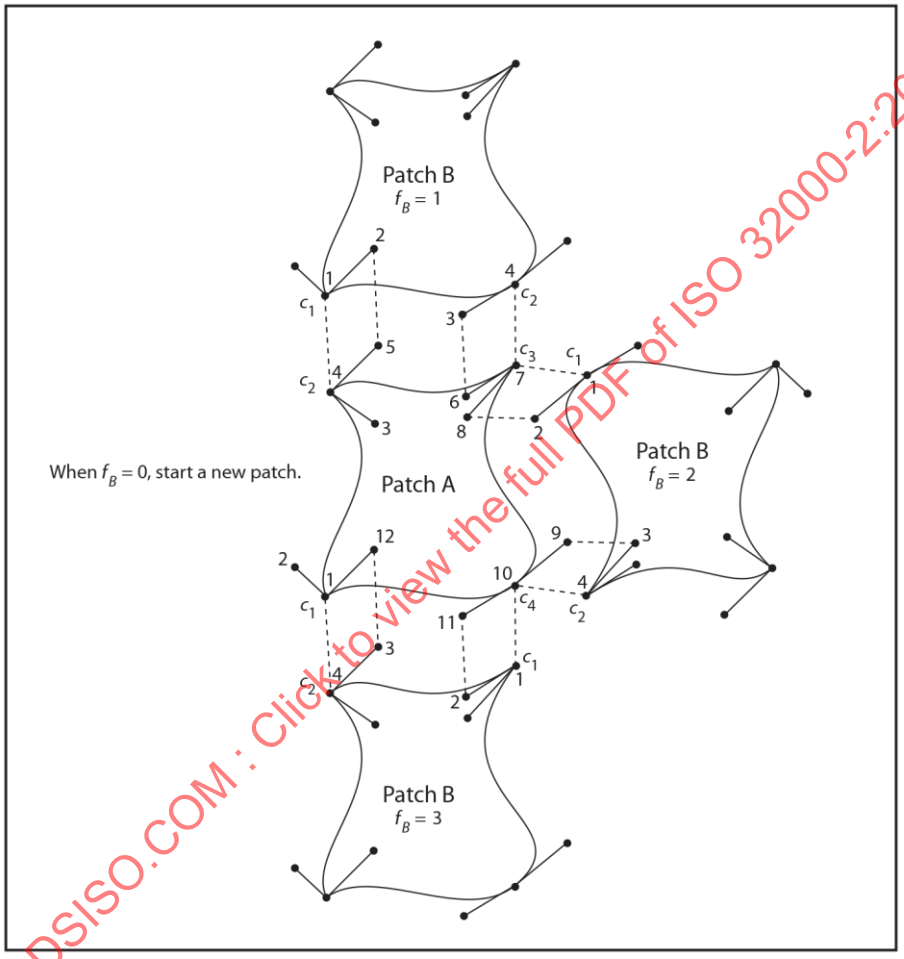


Figure 46 — Edge connections in a Coons Patch Mesh

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 84 — 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

© ISO 2020 – All rights reserved

Edge Flag	Next Set of Data 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 $\quad c_1 = c_2$ previous $(x_2, y_2) = (x_5, y_5)$ previous $\quad c_2 = c_3$ 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 $\quad c_1 = c_3$ previous $(x_2, y_2) = (x_8, y_8)$ previous $\quad c_2 = c_4$ 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 $\quad c_1 = c_4$ previous $(x_2, y_2) = (x_{11}, y_{11})$ previous $\quad c_2 = c_1$ 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 (tensor-product patch mesh) shadings

Type 7 (tensor-product patch mesh) shadings 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 47 — Control points in a tensor-product patch"):

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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}

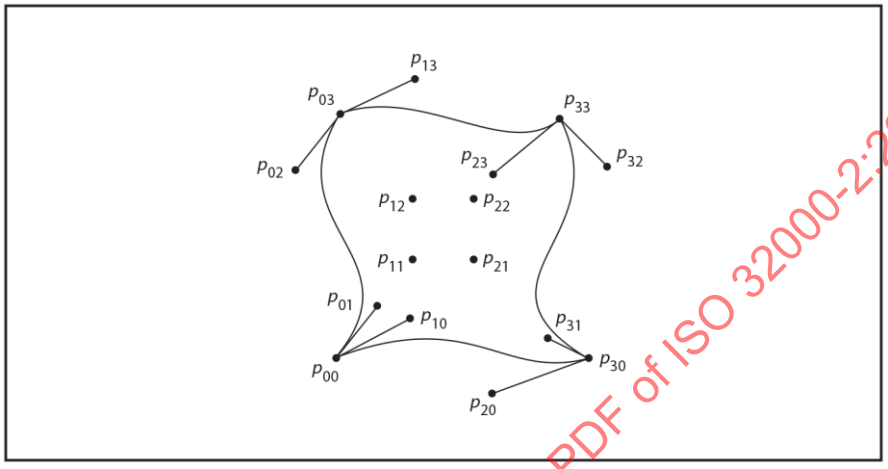


Figure 47 — Control points in a tensor-product patch

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 over 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}...p_{30}) and top (p_{03}...p_{33}) boundary curves,

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 $p_{00}...p_{03}$, the trajectory of the ending point is defined by points $p_{30}...p_{33}$, and those of the two intermediate control points by $p_{10}...p_{13}$ and $p_{20}...p_{23}$. 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

$$p_{11} = \frac{1}{9}(-4 \times p_{00} + 6 \times (p_{01} + p_{10}) - 2 \times (p_{03} + p_{30}) + 3 \times (p_{31} + p_{13}) - 1 \times p_{33})$$

$$p_{12} = \frac{1}{9}(-4 \times p_{03} + 6 \times (p_{02} + p_{13}) - 2 \times (p_{00} + p_{33}) + 3 \times (p_{32} + p_{10}) - 1 \times p_{30})$$

$$p_{21} = \frac{1}{9}(-4 \times p_{30} + 6 \times (p_{31} + p_{20}) - 2 \times (p_{33} + p_{00}) + 3 \times (p_{01} + p_{23}) - 1 \times p_{03})$$

$$p_{22} = \frac{1}{9}(-4 \times p_{33} + 6 \times (p_{32} + p_{23}) - 2 \times (p_{30} + p_{03}) + 3 \times (p_{02} + p_{20}) - 1 \times p_{00})$$

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 85 — Data values in a tensor-product patch mesh" summarises the data values for various values of the edge flag *f*, expressed in terms of the row and column indices used in "Figure 47 — Control points in a tensor-product patch". See 8.7.4.5.5, "Type 4 (free-form Gouraud-shaded triangle mesh) shadings" for further details on the format of the data.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 85 — 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 $\quad c_{00} = c_{03}$ previous $(x_{01}, y_{01}) = (x_{13}, y_{13})$ previous $\quad c_{03} = c_{33}$ 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 $\quad c_{00} = c_{33}$ previous $(x_{01}, y_{01}) = (x_{32}, y_{32})$ previous $\quad c_{03} = c_{30}$ 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 $\quad c_{00} = c_{30}$ previous $(x_{01}, y_{01}) = (x_{20}, y_{20})$ previous $\quad c_{03} = c_{00}$ previous $(x_{02}, y_{02}) = (x_{10}, y_{10})$ previous $(x_{03}, y_{03}) = (x_{00}, y_{00})$ previous

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 two 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.

Two further categories of external objects, *group XObjects* and *reference XObjects* (both PDF 1.4), are

© ISO 2020 – All rights reserved

actually specialised 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 86 — XObject operator"). This operator applies to any type of XObject — image or form. The syntax is the same in all cases, although details of the operator's behaviour differ depending on the type.

Table 86 — 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") or **Form** (see 8.10, "Form XObjects").

Annex J, "XObject comparison", contains one method by which XObjects can be compared for equivalency.

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 48 — Typical sampled image" shows a typical sampled image.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

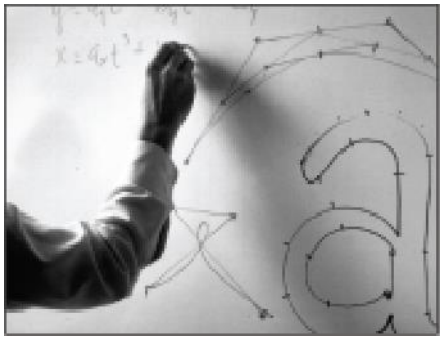


Figure 48 — Typical sampled image

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,

© ISO 2020 – All rights reserved

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".

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 PDF processor 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 subclauses 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 can 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 is not specified in the image dictionary. The colour space does not have to be specified in the image 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 PDF processor 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 are ordered by row, with the horizontal coordinate varying most rapidly. This is shown in "Figure 49 — Source image coordinate system", 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 50 — Mapping the source image"). 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

be described by the matrix [1/*w* 0 0 -1/*h* 0 1].

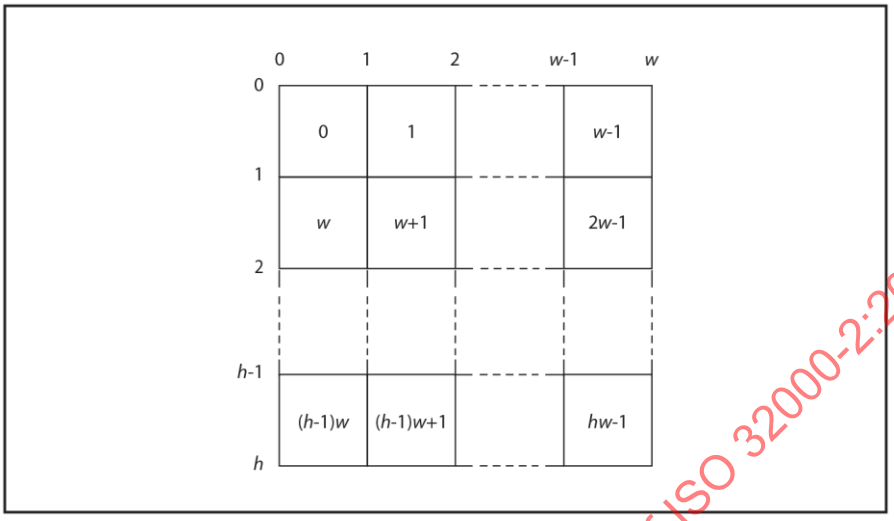


Figure 49 — Source image coordinate system

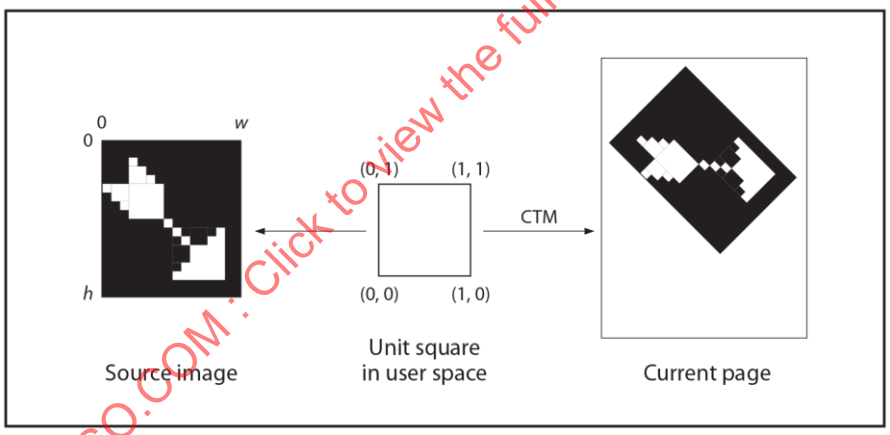


Figure 50 — Mapping the source image

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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 87 — Additional entries specific to an image dictionary" in addition to the usual entries common to all streams (see "Table 5 — Entries common to all stream dictionaries"). 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.

NOTE Some of the entries are not defined 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").

Table 87 — 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	(*Optional when used only as a thumbnail image, required otherwise*) The type of XObject that this dictionary describes; shall be *Image* for an image XObject. NOTE The conditions for when the **Subtype** key is required were clarified in this document (2020).
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 permitted 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 JPEG 2000 data shall be ignored. • If **ColorSpace** is absent, the colour space specifications in the JPEG 2000 data shall be used. The **Decode** array shall also be ignored unless **ImageMask** is *true*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 (from 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 PDF processor in the process of decoding the JPEG 2000 image.
Intent	name	(*Optional; PDF 1.1*) The name of a colour rendering intent that shall be used in rendering any image that is not an image mask (see 8.6.5.8, "Rendering intents"). This value is ignored if **ImageMask** is *true*. 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**, if present, 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; shall not be present 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 if **ColorSpace** is absent, the **Decode** array shall be ignored unless **ImageMask** is *true*. Default value: see "Table 88 — Default decode arrays".
Interpolate	boolean	(*Optional*) A flag indicating whether image interpolation should be performed by a PDF processor (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"). This entry shall not be present in an image XObject that is itself an alternate image.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
SMask	stream	(*Optional; PDF 1.4*) A subsidiary image XObject defining a soft-mask image (see 11.6.5.2, "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 and **SMaskInData** has value 0, the image shall have no associated soft mask (although the current soft mask in the graphics state may still apply). NOTE 1 Interactions between **SMask**, **SMaskInData** and the current soft mask in the graphics state are set out in clause 11.6.4.3, "Mask shape and opacity".
SMaskInData	integer	(*Optional for images that use the **JPXDecode** filter; meaningless otherwise; PDF 1.5*) A code specifying how soft-mask information (see 11.6.5.2, "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 PDF processor shall 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 premultiplied with an opacity channel; the image data also includes the opacity channel. A PDF processor shall create a soft-mask image 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 non-zero value, **SMask** shall not be specified. See also 7.4.9, "JPXDecode filter". NOTE 2 Interactions between **SMask**, **SMaskInData** and the current soft mask in the graphics state are set out in clause 11.6.4.3, "Mask shape and opacity". Default value: *0*.
Name	name	(*Required in PDF 1.0; optional otherwise; deprecated in PDF 2.0*) The name by which this image XObject is referenced in the XObject subdictionary of the current resource dictionary (see 7.8.3, "Resource dictionaries").
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 14.7.5.4, "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; deprecated in PDF 2.0*) 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 PDF processor, 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.
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this image XObject. See 14.13, "Associated files" and 14.13.7, "Associated files linked to XObjects" for more details.
Measure	dictionary	(*Optional; PDF 2.0*) A measure dictionary (see "Table 266 — Entries in a measure dictionary") that specifies the scale and units which shall apply to the image.
PtData	dictionary	(*Optional; PDF 2.0*) A point data dictionary (see "Table 272 — Entries in a point data dictionary") that specifies the extended geospatial data that shall apply to the image.

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
       <</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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        /Im1 Do                                    %Paint image
      Q                                            %Restore graphics state
endstream
endobj
```

8.9.5.2 Decode arrays

Each image's colour component data is initially decomposed into integers in the domain 0 to 2^n-1, where *n* is the bit depth of the colour component. This bit depth is specified as the value of the image dictionary's **BitsPerComponent** entry or, when the image uses the **JPXDecode** filter, is defined in the JPEG 2000 data and can have different values per colour component. An 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 PDF processor shall be a linear transformation; that is, it shall use the following formula for linear interpolation:

$$y = \text{Interpolate}(x, x_{\min}, x_{\max}, y_{\min}, y_{\max}) = y_{\min} + \left((x - x_{\min}) \times \frac{y_{\max} - y_{\min}}{x_{\max} - x_{\min}} \right)$$

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 $x = x_{\min}$ for all $x \le x_{\min}$, and $x = x_{\max}$ for all $x \ge x_{\max}$.

For a **Decode** array of the form $[D_{\min}\ D_{\max}]$, this can be written as

$$y = \text{Interpolate}(x, 0, 2^n - 1, D_{\min}, D_{\max}) = D_{\min} + \left(x \times \frac{D_{\max} - D_{\min}}{2^n - 1} \right)$$

where

n shall be the bit depth of the corresponding colour component

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 88 — Default decode arrays" lists the default **Decode** arrays which shall be used with the various colour spaces by a PDF processor.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 88 — 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
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 PDF supports mappings that invert 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 can 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 ÷ 63).

If an output value is not permitted for a component, it shall be adjusted to the nearest allowed value.

8.9.5.3 Image interpolation

Image interpolation is an attempt to produce a smooth transition between adjacent sample values when rendering an image whose resolution is significantly lower than that of the output device. Setting the value of the **Interpolate** entry in an image dictionary to *true*, is a way for a PDF to declare to a PDF processor that a specific image might render better if interpolation is used for this particular image. However, this is only a hint, and a PDF processor may ignore it.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 89 — Entries in an alternate image dictionary" shows the contents of an alternate image dictionary.

Table 89 — 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 is the default version to be used for printing according to the algorithm described below. At most one alternate for a given base image shall be so designated. Default value: *false*.
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.

In PDF 1.5, optional content (see 8.11, "Optional content") may be used to facilitate selection between alternate images. The following algorithm shall be used to determine which image, if any, shall be rendered:

NOTE (2020) The following algorithm was changed in this document to reflect that **OC** processing has precedence over **DefaultForPrinting** functionality, and situations where no image is to be rendered.

a) If the base image contains an **OC** key then **DefaultForPrinting** shall be ignored on all **Alternates** entries.

b) If the base image contains an **OC** entry that specifies that the base image is visible, then the base image shall be rendered.

c) If the base image contains an **OC** entry that specifies that the base image is not visible, then the list of alternate image dictionaries specified by the base image **Alternates** entry shall be examined in order, and the first entry not containing an **OC** key, or containing an **OC** entry specifying that the alternate image should be visible, shall be selected. Further, if this selected alternate image has an **OC** entry, then that **OC** entry shall also be processed to determine if the alternate image shall be rendered or not. If none of the alternate image dictionaries have an **OC** key, or none of the alternate image dictionaries with an **OC** entry specify that that alternate image is visible, then nothing shall be shown. **DefaultForPrinting** shall be ignored on all **Alternates** entries.

d) If the base image does not contain an **OC** key and the PDF is being printed then the first entry in the **Alternates** array of the base image that has **DefaultForPrinting** set to *true* shall be selected. Further, if

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

this selected alternate image has an **OC** entry, then that **OC** entry shall also be processed to determine if the alternate image shall be printed or not. If no alternate image dictionary in the **Alternates** array has **DefaultForPrinting** set to *true*, then the base image shall be printed.

NOTE Alternate images cannot also have an **Alternates** key as described in "Table 87 — Additional entries specific to an image dictionary".

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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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:

© ISO 2020 – All rights reserved

- 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 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 interactive PDF processors 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 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 9.6.5.3, "Encodings for Type 3 fonts", an image mask, rather than an image, need 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 minimise 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 subclause 8.9.6.2, "Stencil masking", which serves as an explicit mask for the primary (base) image. The base

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 $2 \times n$ integers, $[\min_1 \max_1 \ldots \min_n \max_n]$, 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 $min_i \le ci \le max_i$ for all $1 \le i \le n$).

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 although **JPXDecode** has a lossy filter option. The use of a lossy filter means that the output is only an approximation of the original input data. Therefore, the use of this filter can 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 PDF processor less flexibility in managing the image data, it should be used only for small images (4096 bytes 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 summarised in "Table 90 — Inline image operators". **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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 90 — 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 (as listed in "Table 91 — Entries in an inline image object") are analogous to their respective key-value pairs in an image XObject dictionary (see "Table 87 — Additional entries specific to an image dictionary") or a stream dictionary (see "Table 5 — Entries common to all stream dictionaries"). For convenience, the abbreviations shown in "Table 91 — Entries in an inline image object" and "Table 92 — Additional abbreviations in an inline image object" may be used in place of the full names. Entries other than those listed shall be ignored.

The value of the **Length** (or **L**) key, which shall be present on all inline images, is the length of the data between the **ID** and **EI** operators excluding the white-space delimiting those operators. The value of the **Length** key should not exceed 4096 bytes.

NOTE 1 Because the **Length** (or **L**) key is new to PDF 2.0, PDF processors will not encounter this key in older versions of PDF.

NOTE 2 The **L** key permits PDF processors to efficiently skip inline images if they do not need to display them. To skip an image a processor can advance beyond the single white-space character following the **ID** operator, then if the final or only filter is **ASCIIHexDecode** or **ASCII85Decode** skip any further white-space. The number of characters expressed by the **L** key is then skipped, and the **EI** operator is expected following optional white-space.

"Table 92 — Additional abbreviations in an inline image object" 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**, **Crypt** and **JPXDecode** are not listed in "Table 92 — Additional abbreviations in an inline image object", because those filters shall not be used with inline images.

Table 91 — Entries in an inline image object

Full Name	Abbreviation
BitsPerComponent	**BPC**
ColorSpace	**CS**
Decode	**D**

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Full Name	Abbreviation
DecodeParms	**DP**
Filter	**F**
Height	**H**
ImageMask	**IM**
Intent (*PDF 1.1*)	No abbreviation
Interpolate	**I** (uppercase I)
Length (*PDF 2.0*)	**L**
Width	**W**

Table 92 — Additional abbreviations in an inline image object

Full Name	Abbreviation
DeviceGray	**G**
DeviceRGB	**RGB**
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**

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The colour space specified by the **ColorSpace** (or **CS**) entry shall be one of the standard device colour spaces (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**) and shall be present unless **ImageMask** (**IM**) is present and has the value of *true*. 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.

© ISO 2020 – All rights reserved

NOTE 3 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**, **Crypt** and **JBIG2Decode**. The bytes between the **ID** operator and a white-space token, but before the **EI** operator 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 4 This is an exception to the usual rule that the data in a content stream is interpreted according to the standard PDF syntax for objects. Accordingly, this does not permit comments (see 7.2.4, "Comments") within the image data.

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
   /L 763
   /F [/A85 /LZW]                                   %Filters

ID                                                  %Begin image data
J1/gKA>.]AN&J?]-<HW]aRVcg*bb.\eKAdVV%/PcZ
… Image data representing 289 samples …
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 PDF processor can optimise 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, "Forms". Whereas the form XObjects described in this subclause 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:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- 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 specialised 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 11.6.5.1, "Soft-mask dictionaries" and 11.6.6, "Transparency group XObjects").
- Another specialised 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 PDF 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 PDF processor 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 93 — Additional entries specific to a Type 1 form dictionary", in addition to the usual entries common to all streams (see "Table 5 — Entries common to all stream dictionaries").

Table 93 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 14.7.5.4, "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 14.7.5.4, "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; deprecated in PDF 2.0*) 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; deprecated in PDF 2.0*) The name by which this form XObject is referenced in the **XObject** subdictionary of the current resource dictionary (see 7.8.3, "Resource dictionaries").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this form XObject. See 14.13, "Associated files" and 14.13.7, "Associated files linked to XObjects" for more details.
Measure	dictionary	(*Optional; PDF 2.0*) A measure dictionary (see "Table 266 — Entries in a measure dictionary") that specifies the scale and units which shall apply to the form.
PtData	dictionary	(*Optional; PDF 2.0*) A point data dictionary (see "Table 272 — Entries in a point data dictionary") that specifies the extended geospatial data that shall apply to the form.

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]
         /Length 58
      >>
stream
      0  0 m
      0  1000 l
      1000 1000 l
      1000 0 l f
endstream
endobj
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 94 — Entries common to all group attributes dictionaries", 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 94 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 should be processed by a PDF processor when the referenced content is not available.

NOTE The proxy can 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.

PDF processors that do not recognise the **Ref** entry shall simply display or print the proxy as an ordinary form XObject. Those PDF processors that do implement reference XObjects shall use the proxy in place of the imported content if the latter is unavailable. An interactive PDF processor 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 95 — Entries in a reference dictionary" shows the contents of the reference dictionary.

Table 95 — Entries in a reference dictionary

Key	Type	Value
F	file specification	(*Required*) The PDF 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
ID	array	(*Optional*) An array of two byte strings constituting a PDF file identifier (14.4, "File identifiers") for the PDF file containing the target document. The use of this entry improves a PDF processor's chances of finding the intended PDF file and allows it to warn the user if the PDF 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, a PDF processor may emit any of the following items, depending on its capabilities, 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

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 that 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.

8.11 Optional content

8.11.1 General

Optional content (*PDF 1.5*) refers to sections 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

drawings, layered artwork, maps, and multi-language documents.

The following subclauses 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 the document.
- 8.11.3, "Making graphical content optional", describes how individual pieces of content in a document can 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 PDF processors. 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 96 — Entries in an optional content group dictionary" shows the entries in an optional content group dictionary.

Table 96 — 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 an interactive PDF processor's user interface.
Intent	name or array	(*Optional*) A single name or an array of names that represent the intended use of the graphics in the group. The values *View* and *Design*, or any second-class name may be used (see Annex E, "Extending PDF"). A PDF processor 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 have an **Intent** entry that describes its intended use (see 8.11.2.3, "Intent"), and a **Usage** entry that describes 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 is affected by a given optional content group belongs to that group.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 interactive PDF processor's user interface to change the visibility of content. When a document is first opened by a PDF processor, the groups' states shall be initialised based on the document's default configuration dictionary (see 8.11.4.3, "Optional content configuration dictionaries").

Content belonging to a single group shall be visible when the group is **ON** and invisible when it is **OFF**. Content may, however, belong to multiple groups, when its group is nested inside of another (parent) group. In such a case, the content shall only be visible if this group and all its parent groups indicate visibility. In other words, if the visibility state of an outer level indicates that the content needs to be hidden, all inner levels shall be hidden regardless of their individual visibility states.

8.11.2.2 Optional content membership dictionaries

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 97 — Entries in an optional content membership dictionary".

NOTE 1 8.11.3, "Making graphical content optional" describes how content declares its membership in a group or membership dictionary.

Table 97 — 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 **P** entry 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:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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.

If the **VE** key is present it shall be used in preference to the **OCGs** and **P** keys. For compatibility purposes PDF writers should provide **OCGs** and **P** entries where possible, and especially in cases where the use of **VE** is necessary to express the intended behaviour.

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 2 It is valid 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 subclause. Example 2 in this subclause 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 subclause

```
<</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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        /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 specific intents: *Design*, which may be used to represent a document designer's structural organisation of artwork, and *View*, which may be used for interactive PDF processors.

NOTE The **Intent** entry in "Table 96 — Entries in an optional content group dictionary" 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 can be different (at a finer granularity, for example) than would be useful to consumers of the document. Therefore, a single document can specify different intents for different optional content groups. A PDF processor can 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 it is determined that a piece of optional content in a PDF file is to be hidden, the following shall occur:

- 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.

NOTE 1 Hiding a section of optional content does not change the colour of objects that do not belong to the same optional content group.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- 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 interactive PDF processors, such as searching and editing, may be affected by the ability to selectively show or hide content. An interactive PDF processor 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 2 Tools to select and move annotations need to honour the current on-screen visibility of annotations when performing cursor tracking and mouse-click processing. A full text search engine, however, is likely to need to process all content in a document, regardless of its current visibility on-screen. Export filters could choose the current on-screen visibility, the full content, or present the user with a selection of optional content groups to control visibility.

NOTE 3 A PDF processor 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 content stream, a form or pattern's content stream, glyph descriptions of 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 subclause.

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 is optional content only if the tag is *OC* and the dictionary operand is a valid optional content group that is included in the **OCGs** array of the optional content properties dictionary (see "Table 98 — Entries in the optional content properties dictionary"), or a valid optional content membership dictionary.

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.

NOTE 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 is visible; when the group is **OFF**, the text is 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 **OCG**s 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 is 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
      /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 XObject 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 content 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. The following subclauses describe the PDF structures that are used to specify this information.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 dictionary (see 7.7.2, "Document catalog dictionary") 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 PDF file contains any optional content; if it is missing, a PDF processor shall ignore any optional content structures in the document. This dictionary contains the following entries:

Table 98 — 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.
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 98 — Entries in the optional content properties dictionary" are *configuration dictionaries*, which represent different presentations of a document's optional content groups for use by PDF processors. The **D** configuration dictionary shall be used to specify the initial state of the optional content groups when a document is opened. **Configs** lists other configurations that may be used under particular circumstances. The entries in a configuration dictionary are shown in "Table 99 — Entries in an optional content configuration dictionary".

Table 99 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
BaseState	name	(*Optional*) Used to initialise 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. Any OCG group included in the **ON** array shall not also be included in the **OFF** array. If the **BaseState** entry is *OFF*, this entry is redundant.
Intent	name or array	(*Optional*) A single name or an array of names used by a PDF processor 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"). Since this value may contain any name that could appear as the value of the **Intent** key in an optional content group dictionary, a special name, *All*, is used to indicate the set of all intents. Default value: *View*. (If **Intent** is present in the document's default configuration dictionary, its value shall be *View*.)
AS	array	(*Optional*) An array of usage application dictionaries (see "Table 101 — Entries in a usage application dictionary") specifying which usage dictionary categories (see "Table 100 — Entries in an optional content usage dictionary") shall be consulted by PDF processors, when automatically setting 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Order	array	(*Optional*) An array specifying the order for presentation of optional content groups in an interactive PDF processor'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 interactive PDF processor. • Arrays of optional content groups which may be displayed by an interactive PDF processor 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 an interactive PDF processor'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.
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 an interactive PDF processor. PDF 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. An interactive PDF processor may allow the states of optional content groups to be changed by means other than the user interface, such as ECMAScript or items in the **AS** entry of a configuration dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE Example 1 and Example 2 in this subclause 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]]>>
```

An interactive PDF processor needs to 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]]>>
```

An interactive PDF processor needs to display the optional content groups 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 interactive PDF processors shall, and non-interactive PDF processors should, automatically set the state of optional content groups based on external factors, as discussed in 8.11.4.4, “Usage and usage application dictionaries”.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

8.11.4.4 Usage and usage application dictionaries

Optional content groups are typically constructed to control the visibility of graphics 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 96 — Entries in an optional content group dictionary") 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 100 — Entries in an optional content usage dictionary".

Table 100 — 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. If an Optional Content Group Dictionary (see "Table 96 — Entries in an optional content group dictionary") **Intent** entry contains *Design* then a **CreatorInfo** entry should be included.
Language	dictionary	(*Optional*) A dictionary specifying the language of the content controlled by this optional content group. It shall contain the following entry: **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. Additionally, it may contain the following entry: **Preferred** (*optional*) A name whose values shall be either *ON* or *OFF*. Default value: *OFF*. It shall be used by PDF processors 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 indicates the recommended state for content in this group when the document (or part of it) is saved by a PDF processor to a format that does not support optional content (for example, a raster image format).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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** A number representing the minimum recommended magnification factor at which the group shall be **ON**. Default value: *0*. **max** A number representing the magnification factor below which the group shall be **ON**. Default value: infinity.
Print	dictionary	(*Optional*) A dictionary specifying that the content to 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* or *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.
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 the state of the group when the document is first opened by a PDF processor.
User	dictionary	(*Optional*) A dictionary specifying one or more users for whom this optional content group is primarily intended. This dictionary shall have two required entries: **Type** A name object that shall be either *Ind* (individual), *Ttl* (title or position), or *Org* (organisation). **Name** A text string or array of text strings representing the name(s) of the individual, position or organisation.
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 graphics), *BG* (background image or graphics), or *L* (logo).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

While the data in the usage dictionary serves as information for a document user to examine, it may also be used by PDF processors 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 99 — Entries in an optional content configuration dictionary"). If no **AS** entry is present, states shall not be automatically adjusted based on usage information.

A *usage application dictionary* specifies the rules by which usage entries shall be used by interactive PDF processors, and should be used by non-interactive PDF processors, to automatically manipulate the state of optional content groups, which groups shall be affected, and under which circumstances. "Table 101 — Entries in a usage application dictionary" shows the entries in a usage application dictionary.

© ISO 2020 – All rights reserved

Table 101 — 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 100 — Entries in an optional content usage dictionary"). 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 100 — Entries in an optional content usage dictionary") 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 organisation). 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 have a state of **ON**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 dictionary
      <</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 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 is not affected by zoom level changes. Object 3 receives an **OFF** recommendation when exporting. When printing or exporting, object 4 receives an **ON** recommendation.

8.11.4.5 Determining the state of optional content groups

This subclause summarises the rules by which PDF processors 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 PDF processors:

- Viewer applications which allow users to interact with the document in various ways.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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 can 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 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 initial state used by all PDF processors.

NOTE 2 Viewer applications can 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 in an aggregating application or a stand-alone printing system.

The remaining discussion in this subclause applies only to interactive PDF processors. 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 set-OCG-state actions (see 12.6.4.13, "Set-OCG-state actions") by, for example, clicking links or document outline items. 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 an interactive PDF processor, 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 can be applied.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 subclause is a general description of how glyphs from fonts are painted on the page. Subsequent subclauses 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 PDF processor 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 PDF processor, or obtained from an external font program.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

9.2 Organisation 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 language operators), but advances in this area have made the distinction more meaningful. Consequently, this document distinguishes between characters and glyphs, though with some residual names which are inconsistent.

Glyphs are organised 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 PDF processor 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 specialised 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.

© ISO 2020 – All rights reserved

For most font types, the font program shall be defined in a separate *font program*, 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.

NOTE 2 It is important to carefully distinguish among: 1) PDF objects that compose the PDF structure flowing from a PDF font dictionary, 2) actual font files that can be external to the PDF or embedded in the PDF as a PDF stream object, and 3) terminology used to distinguish varying font and glyph technologies. For example, TrueType is a valid value of the **Subtype** key in a font dictionary, but is also the name used for a glyph technology and the name used for font programs employing that technology. It is often the case that select material from a font file is extracted and used to populate various PDF data structures and the terminology will shift from what is used for font file formats to what is used for the PDF objects.

PDF supports 3 different graphic representations for the shapes of glyphs which have come to be called: Type 1, TrueType and CFF (a compact representation derived from Type 1). Further complicating this, TrueType fonts contain TrueType glyph descriptions, Type 1 fonts contain Type 1 glyph descriptions and OpenType fonts can contain either TrueType glyph descriptions or CFF glyph descriptions. So the term TrueType can be used to distinguish a font file format, the glyph descriptions found in those files, the glyph descriptions found in some OpenType format fonts and the value of the **Subtype** key in a font dictionary. "Table 108 — Font types" provides a good summary of this information.

In the descriptions below the discussion is about the PDF objects used to represent the font information and the glyph technology used, so the terminology is primarily referring to PDF objects. For example, the use of TrueType (as a glyph technology) is much more prevalent than the use of OpenType (as a file format).

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 PDF processor executes the glyph description.

NOTE 3 Programmers who have experience with scan conversion of general shapes need not 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 EXAMPLE 1 perform these steps:

a) Begin a text object.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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.

This subclause explains 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 PDF processor 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 subclause).

```
/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 31 — Entries in a page object". 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 subclause 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 subclause).

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 is 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 centimetre, painting glyphs from the same 12-unit font generates results that are 12 centimetres 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 51 — Glyphs painted in 50% gray".

```
BT
      /F13 48 Tf
      20 40 Td
      0 Tr
      0.5 g
      (ABC) Tj
ET
```

Figure 51 — 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 applies 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 units thick (see "Figure 52 — Glyph outlines treated

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

as a stroked path").

```
BT
      /F13 48 Tf
      20 38 Td
      1 Tr
      2 w
      (ABC) Tj
ET
```

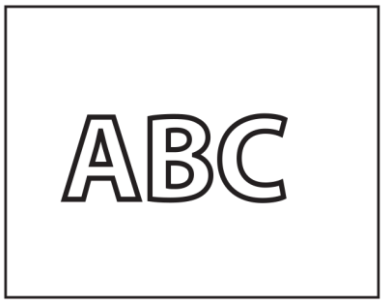


Figure 52 — Glyph outlines treated as a stroked path

EXAMPLE 3 This example illustrates how the glyphs' outlines can 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 53 — Graphics clipped by a glyph path". 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 …
```

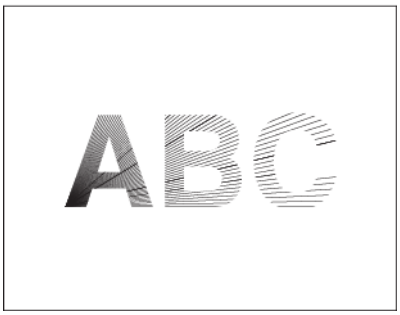


Figure 53 — Graphics clipped by a glyph path

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-

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.

These widths shall be consistent with the actual widths given in the font program.

NOTE 2 Storing this information in the font dictionary, although redundant, enables a PDF processor to determine glyph positioning without having to look inside the font program.

NOTE 3 Due to differences in the way that TrueType and the **Widths** array store width information, there can be cases where widths are not identical between the two. TrueType stores widths in units of 1024 or 2048 to an Em (a unit in the field of typography), equal to the currently specified point size, however the **Widths** array stores widths in units of 1000 to an Em. Using real numbers instead of integers can mitigate rounding errors caused by this difference, and is recommended when precise character positioning is required.

NOTE 4 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 9.6.4, "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 54 — Glyph metrics" shows a typical glyph outline and its metrics.

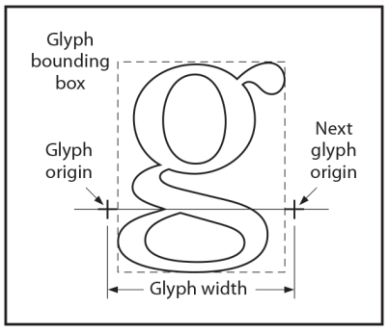


Figure 54 — Glyph metrics

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

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 5 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 non-zero 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 6 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 55 — Metrics for horizontal and vertical writing modes" 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 centre 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

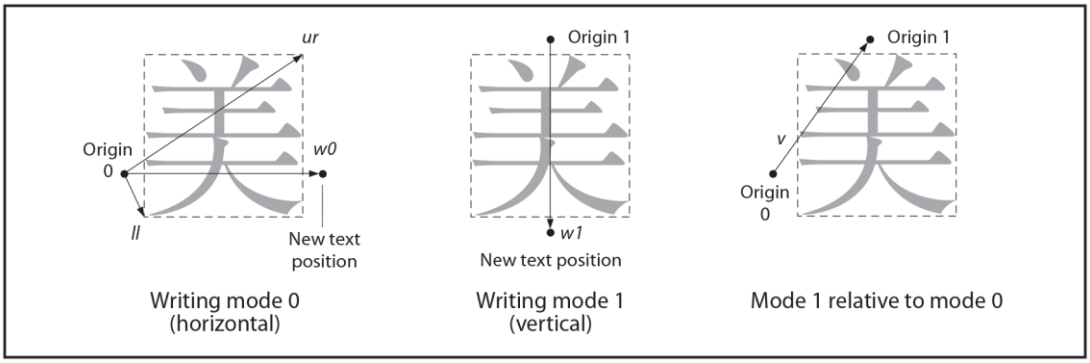


Figure 55 — Metrics for horizontal and vertical writing modes

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 102 — Text state parameters").

Table 102 — 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Except for the previously described T_f and T_{fs}, these parameters are discussed further in subsequent subclauses. (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

© ISO 2020 – All rights reserved

103 — Text state operators". All text state parameters shall apply to Type 3 fonts unless otherwise stated.

NOTE Negative text font size is permitted.

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 initialised to their default values at the beginning of each page.

Table 103 — 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, **Tw**, 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).
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, **Tf**, 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. Zero sized text shall not mark or clip any pixels (depending on text render mode).
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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 56 — Character spacing in horizontal writing"), whereas for vertical writing, a negative value of T_c has this effect.

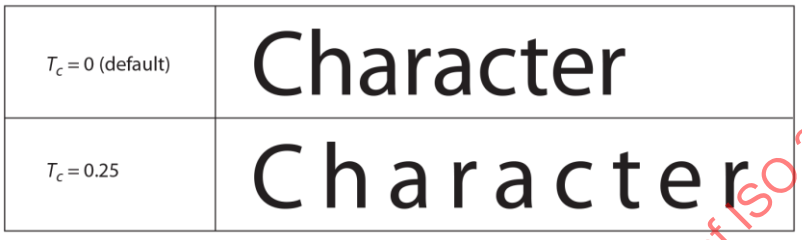


Figure 56 — Character spacing in horizontal writing

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 57 — Word spacing in horizontal writing" illustrates the effect of word spacing in horizontal writing.

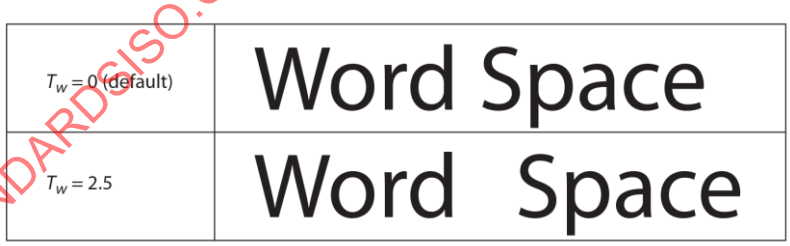


Figure 57 — Word spacing in horizontal writing

Word spacing shall be applied to every occurrence of the single-byte character code 32 in a string when using a simple font (including Type 3) 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,

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 affect the spacing parameters T_c and T_w, as well as any positioning adjustments performed by the **TJ** operator. "Figure 58 — Horizontal scaling" shows the effect of horizontal scaling.

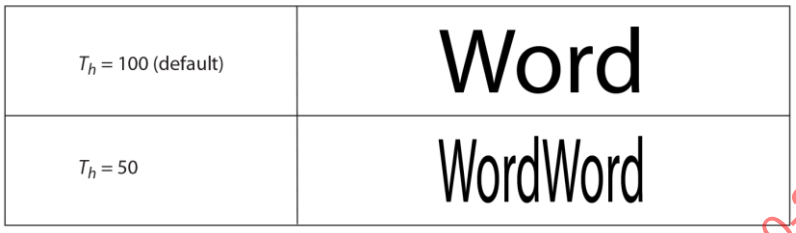


Figure 58 — Horizontal scaling

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 59 — Leading".

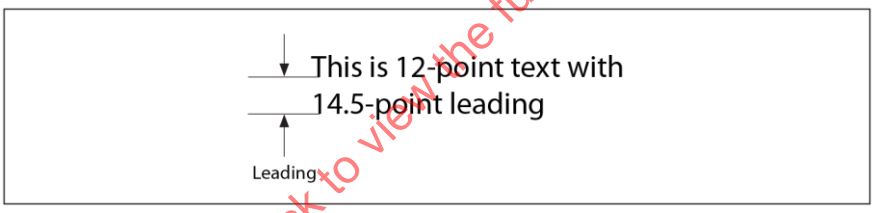


Figure 59 — Leading

The leading parameter shall be used by the **TD**, **T***, **'**, and **"** operators; see "Table 106 — Text-positioning operators" 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 1 The text rendering modes are shown in "Table 104 — Text rendering modes". 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.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 104 — 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.
5		Stroke text and add to path for clipping.
6		Fill, then stroke text and add to path for clipping.
7		Add text to path for clipping.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

At the end of the text object identified by the **ET** operator 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 non-zero winding number rule (see 8.5.3.3.2, "Non-zero 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.

NOTE 2 Due to the use of non-zero winding number rule, the direction of the paths comprising each glyph can cause different output for overlapping glyphs.

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.

© ISO 2020 – All rights reserved

NOTE 3 Certain degenerate glyph sub-paths that are not visible when filled can become apparent when stroking, e.g., a zero line with round end caps will paint a circle according to the current stroke width.

The *e* and *f* components of **Tm** shall be updated for each glyph drawn when using text rendering mode 3 or 7 in exactly the same way as would be done for other text rendering modes.

Where text is drawn using a Type 3 font:

- if text rendering mode is set to a value of 3 or 7, the text shall not be rendered.
- if text rendering mode is set to a value other than 3 or 7, the text shall be rendered using the glyph descriptions in the Type 3 font.
- If text rendering mode is set to a value of 4, 5, 6 or 7, nothing shall be added to the clipping path.

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 60 — Text rise" 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*.

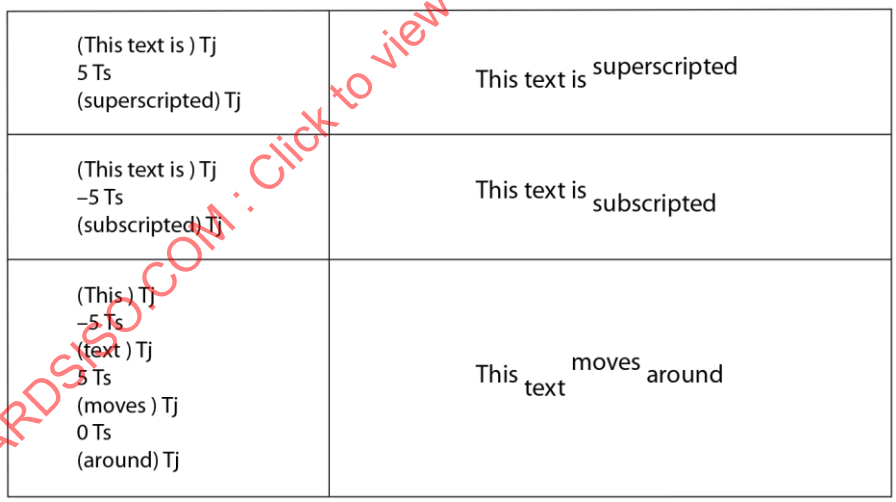


Figure 60 — Text rise

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. Any **TK** value in a graphics state parameter dictionary installed using the **gs** operator shall be ignored between the **BT** and **ET** operators delimiting a text object. The text knockout parameter controls the behaviour of glyphs obtained from any font type, including Type 3. Its initial

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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*, the behaviour shall be equivalent to treating the entire text object as if it were a non-isolated knockout transparency group; see 11.4.6, "Knockout groups" where each glyph is an individual element in that group's transparency stack. When glyphs overlap, later glyphs shall overwrite ("knock out") earlier ones in the area of overlap. The following additional rules shall apply:

- 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.

NOTE With the above listed exceptions, the compositing described in 11.4.6, "Knockout groups" applies when the text knockout graphics parameter is *true* including the handling of masks.

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 105 — Text object operators".

NOTE Although text objects cannot be statically nested, text can be shown using a Type 3 font whose glyph descriptions include any graphics objects, including another text object. Likewise, the current colour can be a tiling pattern whose pattern cell includes a text object.

EXAMPLE

```
BT
    …Zero or more text operators or other allowed operators…
ET
```

Table 105 — 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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Operands	Operator	Description
—	**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 subclauses 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 — Graphics objects".

When the graphics state stack operators **q** and **Q** (see 8.4.2, "Graphics state stack") are combined with the text object operators **BT** and **ET**, each pair of matching operators (**q** ... **Q**) shall be properly (separately) nested. Therefore, the sequences

```
q
   BT
   …
   ET
Q
```

and

```
BT
   q
   …
   Q
ET
```

are valid, but

```
q
   BT
   …
   Q
ET
```

and

```
BT
   q
   …
   ET
Q
```

are not valid.

NOTE The above paragraph and example sequences were added in this document (2020).

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The text-positioning operators shall only appear within text objects.

Table 106 — Text-positioning operators

Operands	Operator	Description
tx ty	**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: $T_m = T_{lm} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \times T_{lm}$
tx ty	**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
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 T_m 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 106 — Text-positioning operators" alter T_m and thereby control the placement of glyphs that are subsequently painted. Also, the *text-showing operators*, described in "Table 107 — Text-showing operators", 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.

© ISO 2020 – All rights reserved

Additionally, within a text object, a PDF processor 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 "Table 106 — Text-positioning operators" and "Table 107 — Text-showing operators".

NOTE This can be used to compactly represent evenly spaced lines of text.

9.4.3 Text-showing operators

The text-showing operators ("Table 107 — Text-showing operators") 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 107 — Text-showing operators".

The text-showing operators shall only appear within text objects.

Table 107 — 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`
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 zero 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 61 — Operation of the TJ operator in horizontal writing" shows an example of the effect of passing offsets to **TJ**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

[(AWAY again)] TJ	AWAY again
[(A) 120 (W) 120 (A) 95 (Y again)] TJ	AWAY again

Figure 61 — 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 9.6.5, "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 an empty string or 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 1 Use of longer text strings is generally more efficient.

NOTE 2 In some cases, the text that is extracted can vary depending on the grouping of glyphs into strings. See, for example, 14.8.2.5.3, "Reverse-order show strings".

NOTE 3 Empty strings are valid.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 2 Conceptually, the entire transformation from text space to device space can 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 then shall be 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 language name, its encoding, and information that can be used to provide a substitute when the font program is not available. Optionally, the font program (more commonly known as font files) may be embedded as a stream object in the PDF file.

NOTE 1 See 9.2, "Organisation and use of fonts" for more introductory information.

"Table 108 — Font types" lists the types of fonts that are supported in ISO 32000 along with the value that shall be used for the **Subtype** key in the font dictionary that represents the font. No values for **Subtype** other than those listed in the table are valid. 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 108 — Font types" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 108 — 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 9.6.4, "Type 3 fonts")
TrueType	*TrueType*	A font based on the TrueType font format (see 9.6.3, "TrueType fonts") and with glyph descriptions based on TrueType glyph technology.
CIDFont	*CIDFontType0*	(*PDF 1.2*) A CIDFont whose glyph descriptions are based on CFF font technology (see 9.7.4, "CIDFonts")
	CIDFontType2	(*PDF 1.2*) A CIDFont whose glyph descriptions are based on TrueType glyph technology (see 9.7.4, "CIDFonts")

NOTE 2 Embedding of OpenType font programs (files) in PDF is described in "Table 124 — Embedded font organisation for various font types" when the value of the **FontFile3** key is *OpenType*. The font dictionary that refers to it can have a **Subtype** of *TrueType*, *Type1*, *CIDFontType0*, or *CIDFontType2*, depending upon the glyph technology used in the OpenType font and other details provided in that table.

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 3 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 specialised font interpreter when necessary; its contents never materialize as PDF objects.

NOTE 4 Most font programs (and related programs, such as CIDFonts and CMaps) conform to external specifications, such as the *Adobe Type 1 Font Format*. This document does not include those specifications. See 2, "Normative references" for more information about the specifications mentioned in this clause.

NOTE 5 The most predictable and dependable results are produced when all font programs used to show text are embedded in the PDF file. See 9.9, "Embedded font programs" for the precise description of 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 PDF processor's environment. See 9.8, "Font descriptors" for some conventions for referring to external font programs. However, some details of font naming, font substitution, and glyph selection are implementation-dependent and can vary among different PDF processors and operating system environments.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 (see 9.4.3, "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 9.6.5, "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, but strongly recommended, 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 stylised PostScript language 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 language compatible interpreter; a specialised Type 1 font interpreter suffices.

A Type 1 font dictionary may contain the entries listed in "Table 109 — Entries in a Type 1 font dictionary".

Table 109 — 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 in PDF 1.1 through 1.7, deprecated in PDF 2.0*) The name by which this font is referenced in the **Font** subdictionary of the current resource dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
BaseFont	name	(*Required*) The PostScript language 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 language name of the font may be used to find the font program in the PDF processor or its environment. It is also the name that is used when printing to a PostScript language compatible output device.
FirstChar	integer	(*Required; optional in PDF 1.0-1.7 for the standard 14 fonts*) The first character code defined in the font's **Widths** array.
LastChar	integer	(*Required; optional in PDF 1.0-1.7 for the standard 14 fonts*) The last character code defined in the font's **Widths** array.
Widths	array	(*Required; optional in PDF 1.0-1.7 for the standard 14 fonts; indirect reference preferred*) An array of (**LastChar** - **FirstChar** + 1) numbers, 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".
FontDescriptor	dictionary	(*Required; optional in PDF 1.0-1.7 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) (PDF 1.0-1.7)".
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, "Character sets and encodings") or an encoding dictionary that shall specify differences from the font's built-in encoding or from a specified predefined encoding (see 9.6.5, "Character encoding").
ToUnicode	stream	(*Optional; PDF 1.2*) A stream containing a CMap file that maps character codes to Unicode values (see 9.10.3, "ToUnicode CMaps").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

PDF versions 1.0 to 1.7 did not require Type 1 font dictionaries to include **FirstChar**, **LastChar**, **Widths** and **FontDescriptor** entries as described in 9.6.2.2, "Standard Type 1 fonts (standard 14 fonts) (PDF 1.0-1.7)". For compatibility reasons PDF processors shall provide glyph widths and font descriptor data for those standard fonts for use in processing PDF files when the entries are absent.

EXAMPLE This example shows the font dictionary for the Adobe Garamond® Semibold font. The font has an encoding dictionary (object 25), however neither the encoding dictionary nor the font descriptor (object 7) is shown.

```
14 0 obj
      <</Type /Font
         /Subtype /Type1
         /BaseFont /AGaramond-Semibold
```


© ISO 2020 – All rights reserved

```
            /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) (PDF 1.0-1.7)

The PostScript language 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.

In PDF 1.0 to PDF 1.7, the **FirstChar**, **LastChar**, **Widths** and **FontDescriptor** (see "Table 109 — Entries in a Type 1 font dictionary") were optional in Type 1 font dictionaries for the standard 14 fonts. PDF processors supporting PDF 1.0 to PDF 1.7 files shall have these fonts, or their font metrics and suitable substitution fonts, available.

These fonts, or their font metrics and suitable substitution fonts, shall be available to the PDF processor.

9.6.2.3 Multiple master fonts

The multiple master font format is an extension of the Type 1 font format as specified in Adobe Technical Note #5015, *Type 1 Font Format Supplement*, 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.

The font dictionary for a multiple master font instance contains the same entries as a Type 1 font

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

dictionary (see "Table 109 — Entries in a Type 1 font dictionary"), with these differences:

- The value of **Subtype** shall be *MMType1*.
- If the PostScript language 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.

NOTE This convention is commonly used for accessing multiple master font instances from an external source in the PDF processor's environment; it is documented in *Adobe Technical Note #5088, Font Naming Issues*. However, this convention is not prescribed as part of this 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

PDF TrueType fonts are those with a **Subtype** value of *TrueType* in the font dictionary. PDF fonts of this type shall support both the TrueType font format (see *Apple Computer, Inc., TrueType Reference Manual*) as well as the OpenType font format (as defined by ISO/IEC 14496-22).

A TrueType font dictionary may contain the same entries as a Type 1 font dictionary (see "Table 109 — Entries in a Type 1 font dictionary"), with these differences:

- The value of **Subtype** shall be *TrueType*.
- The value of **Encoding** is subject to limitations that are described in 9.6.5.4, "Encodings for TrueType fonts".
- The value of **BaseFont** is derived differently.

The PostScript language name for the value of **BaseFont** should be determined in one of two ways:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- If the TrueType or OpenType font program's "name" table contains a PostScript language name, it should be used.
- In the absence of a PostScript language name in the "name" table, a PostScript language name should be derived from the name by which the font is known in the host operating system.

NOTE 1 The *OpenType* font format (also known as Open Font Format, or ISO/IEC 14496-22) was developed jointly by Microsoft and Adobe. It extends the TrueType font format, developed by Apple, to include more metadata about the glyphs and text layout as well as supporting both the TrueType glyph technology and the CFF glyph technology (see Adobe Technical Note #5176, *The Compact Font Format Specification*). OpenType is available on most modern operating systems.

NOTE 2 A TrueType or an OpenType font program can be embedded directly in a PDF file as a stream object.

NOTE 3 The Type 42 font format that is defined for the PostScript language 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 can have a name that is written in Japanese using some (unidentified) Japanese encoding. Thus, TrueType font names can 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 Type 3 fonts

Type 3 fonts differ from the other fonts supported by PDF. Font dictionaries for other fonts simply contain information about the font and refer to a separate font program for the actual glyph descriptions; a Type 3 font dictionary contains the 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 can 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 110 — Entries in a Type 3 font dictionary".

Table 110 — 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 109 — Entries in a Type 1 font dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 all marks that would result if all of the glyphs of the font were placed with their origins coincident and their descriptions executed. If all four elements of the rectangle are zero, a PDF processor shall make no assumptions about glyph sizes based on the font bounding box. If any element is non-zero, the font bounding box shall be accurate. If any glyph's marks fall outside this bounding box, behaviour is implementation dependent and may not match the creator's expectations.
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. See below for more details about Type 3 glyph descriptions.
Encoding	dictionary	(*Required*) An encoding dictionary whose **Differences** array shall specify the complete character encoding for this font (see 9.6.5, "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) numbers, 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"). NOTE (2020) The conditions for when the **FontDescriptor** key is required were corrected in this document to match ISO 32000-1:2008.
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 9.10.3, "ToUnicode CMaps").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

For each character code shown by a text-showing operator that uses a Type 3 font, the PDF processor shall:

a) Look up the character code in the font's **Encoding** entry, as described in 9.6.5, "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.

d) If any glyph descriptions refer to named resources they shall be looked up in the **Resources** entry of the Type 3 font dictionary. 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.

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 graphics state at the point of invocation of the text-showing operator that caused the glyph description to be invoked. To ensure predictable results, the glyph description shall initialise any graphics state parameters on which it depends. In particular, if it invokes any operator from "Table 59 — Path-painting operators" which performs stroking, it shall explicitly set the line width, line join, line cap, and dash pattern to appropriate values. The **TK** flag (see 9.3.8, "Text knockout") of the graphics state controls the behaviour of glyphs obtained from any font type, including Type 3.

NOTE 2 Normally, it is unnecessary and undesirable to initialise the current colour parameters because the text-showing operators are designed to paint glyphs with the current colours.

The glyph description shall execute one of the operators described in "Table 111 — Type 3 font operators" 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 the PostScript language. Some of the information provided in Type 3 font dictionaries and glyph descriptions, although seemingly redundant or unnecessary, is nevertheless required for correct results when a PDF processor prints to a PostScript language compatible output device. This applies particularly to the operands of the **d0** and **d1** operators, which are the equivalent of PostScript's **setcharwidth** and **setcachedevice**. For further explanation, see clause 5.7 of the *PostScript Language Reference, Third Edition*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 111 — 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 1 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 as the first operator 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 or other colour-related parameters explicitly. NOTE 2 An image does set the colour space and colour explicitly; an image mask does not and requires explicit operators to do so for it.
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 3 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 implementation-dependent. A glyph description that begins with the **d1** operator should not execute any operators that set the colour (or other colour-related parameters including transparency) in the graphics state; any use of such operators shall be ignored and the glyph stream continues to be processed without error (see 8.6.8, "Colour operators"). 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 62 — Output from the example" shows the result of showing

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

the string (abababab) using this font.

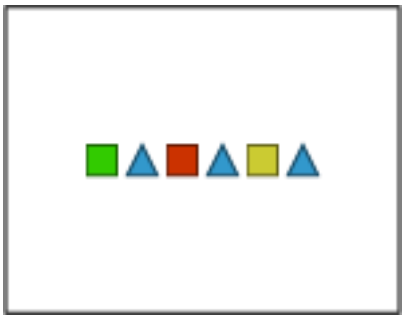


Figure 62 — Output from the example

```
%Snippet of page content stream that places several Type 3 font strings onto the page
   …
   0.2 0.8 0.0 rg
   0.1 0.4 0.0 RG
   BT
      /FT3 15 Tf
      300 400 Td
      (ab) Tj
      0.8 0.2 0.0 rg
      0.4 0.1 0.0 RG
      (ab) Tj
      0.8 0.8 0.2 rg
      0.4 0.4 0.1 RG
      (ab) Tj
   ET
   …


%Type 3 font definition encoding two glyphs, 'a' and 'b'.
4 0 obj
   <<
      /Type /Font
      /Subtype /Type3
      /FontBBox [-36 -36 786 786]
      /FontMatrix [0.001 0 0 0.001 0 0]
      /CharProcs 10 0 R
      /Encoding 9 0 R
      /FirstChar 97
      /LastChar 104
      /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

%Type 3 "square" glyph description
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
11 0 obj
      <</Length …>>
stream
   1000 0 -36 -36 786 786 d1                    %uncoloured glyph - defines only shape
   72 w
   0 0 750 750 re
   B
endstream
endobj

%Type 3 "triangle" glyph description
12 0 obj
      <</Length …>>
stream
   1000 0 d0                                    %coloured glyph - defines colour and shape
   72 w
   0.2 0.6 0.8 rg
   0.1 0.3 0.4 RG
   0 0 m
   375 750 l
   750 0 l
   b
endstream
endobj
```

9.6.5 Character encoding

9.6.5.1 General

A font's encoding is the association between character codes (obtained from text strings that are shown) and glyph descriptions. This subclause 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 PDF writer generating the text being shown.

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 PDF writers use their own special-purpose encodings.
- It permits PDF 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*. 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 112 — Entries in an encoding dictionary"). The regular encodings used for Latin-text fonts on Mac OS and Microsoft 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

sophisticated typography. Complete details of these encodings and of the characters present in typical fonts are provided in Annex D, "Character sets and encodings".

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, "Character sets and encodings".

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 112 — Entries in an encoding dictionary".

Table 112 — 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, "Character sets and encodings"). If this entry is absent, the **Differences** entry shall describe differences from a default base encoding. For a font program that is embedded in the PDF file, the default base encoding shall be the font program's built-in encoding, as described in 9.6.5, "Character encoding" and further elaborated in the subclauses 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 a default base encoding. The **Differences** array is described in subsequent subclauses.

The value of the **Differences** entry shall be an array of character codes and character names organised as follows:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$code_1$ $name_{1,1}$ $name_{1,2}$ ...

$code_2$ $name_{2,1}$ $name_{2,2}$ ...

...

$code_n$ $name_{n,1}$ $name_{n,2}$ ...

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
```

9.6.5.2 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

An **Encoding** entry in the PDF font dictionary, if present, shall override a Type 1 font's mapping from character codes to character names. The **Differences** array maps codes to the names of glyph descriptions that exist in the font program, regardless of whether those glyphs are 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.

NOTE If a font has no **.notdef** glyph definition, the results are implementation dependent.

9.6.5.3 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 for Type 3 fonts.

NOTE Type 3 fonts do not support the concept of a default glyph name.

9.6.5.4 Encodings for TrueType fonts

A TrueType or OpenType 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 subclause 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/OpenType 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 Microsoft 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/OpenType fonts, although some font programs have an optional "post" table listing glyph names for the glyphs. If the PDF processor 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/OpenType glyph selection are dependent on the PDF processor or the operating system, PDF files that use TrueType/OpenType fonts should follow certain guidelines to ensure predictable behaviour across all PDF processors:

- 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. See also D.2 "Latin character set and encodings" notes 5 and 6 for situations when a **Differences** array is required.
- 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 121 — Font flags") 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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/OpenType font programs contain incorrect encoding information. Implementations of TrueType/OpenType font interpreters have evolved heuristics for dealing with such problems; those heuristics are not described here. For maximum portability, only well-formed TrueType/OpenType font programs can be used in PDF files. Therefore, a TrueType/OpenType font program to be used in a PDF file could need modification to conform to these guidelines.

NOTE 2 Not all glyphs of a TrueType/OpenType font are always accessible with simple PDF font resources. A Type 0 PDF font resource can be used to access those glyphs.

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 121 — Font flags") is set, the PDF processor 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 initialised with the mappings described in Annex D, "Character sets and encodings".
- If the **Encoding** entry is a dictionary, the table shall be initialised with the entries from the dictionary's **BaseEncoding** entry (see "Table 112 — Entries in an encoding dictionary"). 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* and *Adobe Glyph List for New Fonts*.
- 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, "Character sets and encodings" with the addition of 15 entries and the replacement of the currency glyph with the Euro glyph, as shown in "Table 113 — Additional entries in Mac OS Roman encoding not in MacRomanEncoding".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 113 — Additional entries in Mac OS Roman encoding not in MacRomanEncoding

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 PDF processor 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 font-like object

© ISO 2020 – All rights reserved

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, the PostScript language allows a Type 0 font to have multiple descendants, which can also be Type 0 fonts. PDF supports only a single descendant, which are always 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 organisation of fonts that have complex encoding requirements.

This subclause 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 Note #5092, *CID-Keyed Font Technology Overview*, and Adobe Technical Note #5014, *Adobe CMap and CIDFont Files Specification*. This subclause 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 implementation limits (see Annex C, "Advice on maximising portability").

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 organised 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 114 — Entries in a CIDSystemInfo dictionary". In order for a CIDFont and a CMap to be compatible, their **Registry** and **Ordering** values shall be the same. Identity CMaps (*Identity-H* and *Identity-V*) are compatible with all CIDFonts.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 114 — Entries in a CIDSystemInfo dictionary

Key	Type	Value
Registry	ASCII string	(*Required*) A string identifying the issuer of the character collection. The string shall begin with the 4 or 5 characters of a registered developer prefix followed by a LOW LINE (5Fh) followed by any other identifying characters chosen by the issuer. See Annex E, "Extending PDF", for how to obtain a unique developer prefix.
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 glyph technology.

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 115 — Entries in a CIDFont dictionary" lists the entries in a CIDFont dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 115 — 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*.

© ISO 2020 – All rights reserved

Key	Type	Value
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 9.9.2, "Font subsets".
CIDSystemInfo	dictionary	(*Required*) A dictionary containing entries that define the character collection of the CIDFont. See "Table 114 — Entries in a CIDSystemInfo dictionary".
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	number	(*Optional*) The default width for glyphs in the CIDFont (see 9.7.4.3, "Glyph metrics in CIDFonts"). Default value: 1000.
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	(*Required for Type 2 CIDFonts with embedded font programs*) 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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

© ISO 2020 – All rights reserved

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 124 — Embedded font organisation for various font types") shall be either *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 (as defined by Adobe Technical Note #5177, *The Type 2 Charstring Format*).

NOTE Although in many fonts the CID value and GID value are the same, the CID and GID values can 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 (refer to Adobe Technical Note #5177, *The Type 2 Charstring Format*).

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, and the Type 2 CIDFont dictionary contains a **CIDToGIDMap** entry, the **CIDToGIDMap** entry shall be ignored, 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 PDF processor 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 subclause.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

The **W** array allows the definition of widths for individual CIDs. The elements of the **W** array shall be numbers organised in groups of two or three, where each group shall be in one of these two formats:

c [w_1 w_2... w_n]

c_{first} c_{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*, as a number, for all CIDs in the range c_{first} to c_{last}. Specifying a given CID value more than once should not be done. In the case where it is done, the first specification is the one that shall be used.

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 inclusive 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 55 — Metrics for horizontal and vertical writing modes"). 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

$v = (w_0 \div 2, 880)$
$w1 = (0, -1000)$

where w_0 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 organised 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} ...]

c_{first} c_{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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

vector *w1* (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 cfirst to clast, 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 116 — Predefined CJK CMap names".
- 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 116 — Predefined CJK CMap names" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

Table 116 — 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
ETenms-B5-V	Vertical version of ETenms-B5-H

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Description
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-CNS1-6 character collection
UniCNS-UCS2-V	Vertical version of UniCNS-UCS2-H
UniCNS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-CNS1-6 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Description
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	
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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

NOTE 2 The *Identity-H* and *Identity-V* CMaps can 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 descendant font of a Type 0 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.

© ISO 2020 – All rights reserved

Table 117 — Character collections for predefined CMaps, by PDF version

Table intentionally empty to retain table numbering in this document (2020). *Information is now located in the appropriate normative reference for each character collection.*

A PDF processor shall support Adobe-CNS1-7, Adobe-GB1-5, Adobe-Japan1-7 and Adobe-KR-9 character collections. ". Adobe-Japan2-0 and Adobe-Korea1-2 are deprecated in this document (2020). 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 PDF processor supporting the same or a later PDF version. When interpreted by a PDF processor 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".

Other supplements of these character collections may 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.

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 PDF processor 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 will need to 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 a variety of online sources.

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 — Entries common to all stream dictionaries"), the CMap stream dictionary contains the entries listed in "Table 118 — Additional entries in a CMap stream dictionary". 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*.

Table 118 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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.

NOTE While often used for mapping 2 byte character codes to CID codes, character codes with just 1 byte or more than 2 bytes can be used, as long as they can be mapped to CID codes without ambiguity.

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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
%%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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

9.7.6 Type 0 font dictionaries

9.7.6.1 General

A Type 0 font dictionary contains the entries listed in "Table 119 — Entries in a Type 0 font dictionary".

© ISO 2020 – All rights reserved

Table 119 — 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 PDF processors.
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").
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 9.9.2, "Font subsets").

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 subclause 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause.

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 subclause 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 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

For an embedded TrueType font with a **CIDtoGIDMap** stream, if a (character) code does not have a corresponding GID in the **CIDtoGIDMap** stream, the glyph for CID 0 shall be substituted.

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 PDF processor to synthesise 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 120 — Entries common to all font descriptors". Additional entries in the font descriptor for a CIDFont are described in 9.8.3, "Font descriptors for CIDFonts". All dimensional 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 120 — 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*) A byte string specifying the preferred font family name. EXAMPLE 1 For the font Times Bold Italic, the **FontFamily** is Times.
FontStretch	name	(*Optional; PDF 1.5*) 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 might appear most similar to *Normal* in another.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
FontWeight	number	(*Optional; PDF 1.5*) The weight (thickness) component of the fully-qualified font name or font specifier. If present, the value shall be one of 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 might 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. Values shall be positive. A value of 0 indicates an unknown stem thickness.
StemH	number	(*Optional*) The thickness measured vertically of the dominant horizontal stems of glyphs in the font. Values shall be positive. A value of 0 indicates an unknown stem thickness. Default value: *0*.
AvgWidth	number	(*Optional*) The average width of glyphs in the font. Default value: *0*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 124 — Embedded font organisation for various font types").
CharSet	ASCII string or byte string	(*Optional; meaningful only in Type 1 fonts; PDF 1.1; deprecated in PDF 2.0*) 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 9.9.2, "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 121 — Font flags" shows the meanings of the flags; all undefined flag bits are reserved and shall be set to 0 by PDF writers. "Figure 63 — Characteristics represented in the Flags entry of a font descriptor" shows examples of fonts with these characteristics.

Table 121 — 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 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Bit position	Name	Meaning
6	Nonsymbolic	Font uses the Standard Latin character set or a subset of it. This flag and the Symbolic flag shall not both be set or both be clear.
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.
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 subclause.

The Nonsymbolic flag (bit 6 in the **Flags** entry) indicates that the font's character set is the 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 Standard character set is considered to be symbolic. This influences the font's default base encoding and may affect a PDF processor's font substitution strategies.

The use of the two flags to represent a single binary choice is a historical accident. A PDF processor should always check the Symbolic flag to determine whether the state is Symbolic or NonSymbolic. PDF writers should still set the NonSymbolic flag to be the inverse of the Symbolic flag for backward compatibility.

NOTE 1 This classification of nonsymbolic and symbolic fonts is peculiar to PDF. A font can contain additional characters that are used in Latin writing systems but are outside the Standard Latin character set; PDF considers such a font to be symbolic.

The ForceBold flag (bit 19) shall determine whether bold glyphs shall be painted with extra pixels even at very small text sizes by a PDF processor. 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 can 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 63 — Characteristics represented in the Flags entry of a font descriptor

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 120 — Entries common to all font descriptors", the **FontDescriptor** dictionaries of CIDFonts may contain the entries listed in "Table 122 — Additional font descriptor entries for CIDFonts".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 122 — 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 a Language-Tag as defined in BCP 47. If this entry is absent, such absence provides no information as to the language of the document; in particular, an absence of this entry does not by itself mean the language of the document is 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; deprecated in PDF 2.0*) 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 9.9.2, "Font subsets"). The stream's data shall be organised 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 FamilyClass 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 2, "Normative references" 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.

© ISO 2020 – All rights reserved

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 120 — Entries common to all font descriptors".

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 123 — Glyph classes in CJK fonts" lists the valid keys for the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, Adobe-Japan2 (*deprecated in PDF 2.0 (2020)*), Adobe-Korea1 (*deprecated in PDF 2.0 (2020)*) and Adobe-KR (*added in PDF 2.0 (2020)*) character collections.

Table 123 — 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Character Collection	Class	Glyphs in Class
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 (*deprecated in PDF 2.0 (2020)*)	**Alphabetic**	Full-width Latin, Greek, and Cyrillic glyphs
	Dingbats	Special symbols
	HojoKanji	Full-width kanji glyphs
Adobe-Korea1 (*deprecated in PDF 2.0 (2020)*) Adobe-KR (*added in PDF 2.0 (2020)*)	**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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
         /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

9.9.1 General

This clause describes how a font program can be embedded in a PDF file as data contained in a PDF stream object.

For text in text rendering mode 3, font programs may be embedded. For text in other text rendering modes, font programs should be embedded.

NOTE 1 Such a stream object is also called a font file by analogy with font programs that are available from sources external to PDF processors.

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 124 — Embedded font organisation for various font types" summarises 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 124 — Embedded font organisation 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. The font program provided as the value of this key shall conform to the *Adobe Type 1 Font Format* and/or Adobe Technical Note #5015, *Type 1 Font Format Supplement*.
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. The font program provided as the value of this key shall conform to the *TrueType Reference Manual*. The font program shall include these tables: "glyf", "head", "hhea", "hmtx", "loca", and "maxp". The "cvt " (notice the trailing SPACE), "fpgm", and "prep" tables shall also be included if they are required by the font instructions.
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. The font program provided as the value of this key shall conform to Adobe Technical Note #5176.
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. The font program provided as the value of this key shall conform to Adobe Technical Note #5176.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Subtype	Description
FontFile3	OpenType	(*PDF 1.6*) OpenType font program, as described in ISO/IEC 14496-22. OpenType is an extension of TrueType that allows inclusion of font programs that use the Compact Font Format (CFF). • 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 shall include these tables: "head", "hhea", "hmtx", "loca", and "maxp". The "cvt " (notice the trailing SPACE), "fpgm", and "prep" tables shall 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 shall 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 shall include the "cmap" table. A **FontFile3** entry with an **OpenType** subtype may appear in the font descriptor for these types of font dictionaries. ISO/IEC 14496-22 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. The font program provided as the value of this key shall conform to ISO/IEC 14496-22:2019. NOTE The absence of some optional tables (such as those used for advanced line layout) can 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 — Entries common to all stream dictionaries"), plus the additional entries listed in "Table 125 — Additional entries in an embedded font stream dictionary".

Table 125 — Additional entries in an embedded font stream dictionary

Key	Type	Value
Length1	integer	(*Required for Type 1 and TrueType font programs*) 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 font programs*) 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 font programs*) 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 PDF processor.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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.

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 can be in binary or ASCII hexadecimal format, PDF supports only the binary format. However, the entire font program can 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 124 — Embedded font organisation for various font types", 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 apply 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 9.6.5.4, "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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The "vhea" and "vmtx" tables that specify vertical metrics shall never be used by a PDF processor. 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 can 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, PDF processors can use the data in OpenType fonts to do advanced line layout, automatically substitute ligatures, provide selections of alternative glyphs to users, and handle complex writing scripts.

The process of finding glyph descriptions in OpenType fonts by a PDF processor shall be the following:

- For Type 1 fonts using "CFF " tables, the process shall be as described in 9.6.5.2, "Encodings for Type 1 fonts".
- For TrueType fonts using "glyf" tables, the process shall be as described in 9.6.5.4, "Encodings for TrueType fonts". Since this process sometimes produces ambiguous results, PDF writers, instead of using a simple font, should use a Type 0 font with an *Identity-H* encoding and use the glyph indices as character codes, as described following "Table 116 — Predefined CJK CMap names".
- 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".

9.9.2 Font subsets

PDF documents may include subsets of PDF fonts whose **Subtype** is *Type1*, *TrueType* or *OpenType*. The font and font descriptor that describe a font subset are slightly different from those of ordinary fonts. These differences allow a PDF processor to recognise 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, that is, 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 (+) followed by the PostScript name of the font from which the subset was created. The tag shall consist of exactly six uppercase letters; the choice of letters is arbitrary, but different subsets of the same font in the same PDF file shall have different tags. The glyph name **.notdef** shall be defined in the font subset.

NOTE It is recommended that PDF processors treat multiple subset fonts as completely independent entities, even if they appear to have been created from the same original font.

EXAMPLE EOODIA+Poetica is the name of a subset of Poetica®, a Type 1 font.

9.10 Extraction of text content

9.10.1 General

The preceding subclauses describe all the facilities for showing text and causing glyphs to be painted on the page. In addition to rendering text, PDF processors often 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

exporting of text to other file formats.

Unicode 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.

When extracting character content, a PDF processor 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 PDF processor. 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 using the following methods:

- 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 PDF processor 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 14.8.2.6, "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 and the glyph selection algorithm (see 9.6.5, "Character encoding") uses a glyph name, that name can be looked up in the *Adobe Glyph List* and *Adobe Glyph List for New Fonts* to obtain the corresponding Unicode value.
- If the font is a composite font that uses one of the predefined CMaps listed in "Table 116 — Predefined CJK CMap names" (except *Identity–H* and *Identity–V*) or whose descendant CIDFont uses the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, Adobe-Korea1 *(deprecated in PDF 2.0 (2020))* or Adobe-KR *(added in PDF 2.0 (2020))* 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 a variety of online sources, e.g. https://github.com/adobe-type-tools/mapping-resources-pdf).
    e. Map the CID obtained in step (a) according to the CMap obtained in step (d), producing a Unicode value.

Type 0 fonts whose descendant CIDFonts use the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, Adobe-

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

Korea1 *(deprecated in PDF 2.0 (2020))* or Adobe-KR *(added in PDF 2.0 (2020))* character collection (as specified in the **CIDSystemInfo** dictionary) shall have a supplement number corresponding to the version of PDF supported by the PDF processor.

If these methods fail to produce a Unicode value, there is no way to determine what the character code represents in which case a PDF processor 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*. This CMap differs from an ordinary one in these ways:

- The only pertinent entry in the CMap stream dictionary (see "Table 118 — Additional entries in a CMap stream dictionary") 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.

For simple fonts, character codes shall be written as 1 byte in the ToUnicode CMap.

For CID keyed fonts character codes may have 1 byte, 2 bytes, or more than 2 bytes in the ToUnicode CMap.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

codes are shown.

```
16 0 obj
<<
   /Type /CMap
   /CMapName /Adobe-Identity-UCS2
   /CIDSystemInfo << /Registry (Adobe) /Ordering (UCS2) /Supplement 0 >>
   /Length 433
>>
stream
/CIDInit /ProcSet findresource begin
12 dict begin
begincmap
/CIDSystemInfo
   <</Registry (Adobe)
/Ordering (UCS2)
/Supplement 0
>> def
/CMapName /Adobe-Identity-UCS2 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 SPACE (U+0020) to TILDE (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: LATIN SMALL LETTER F (U+0066) LATIN SMALL LETTER F (U+0066) are the Unicode values for the character sequence f f, LATIN SMALL LETTER F (U+0066) LATIN SMALL LETTER I (U+0069) for f i, and LATIN SMALL LETTER F (U+0066) LATIN SMALL LETTER F (U+0066) LATIN SMALL LETTER L (U+006c) for f f l.

Finally, the character code <3A 51> is mapped to the Unicode value UNICODE HAN CHARACTER 'U+2003E' (U+2003E), which is expressed by the byte sequence <D840DC3E> in UTF-16BE encoding.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Example 2 in this subclause 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:

© ISO 2020 – All rights reserved

```
n beginbfrange
srcCode1 srcCode2 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 - (*srcCode2* - *srcCode1*). 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
srcCode1 srcCode2 [dstString1 dstString2...dstStringm]
endbfrange
```

Consecutive codes starting with *srcCode1* and ending with *srcCode2* shall be mapped to the destination strings in the array starting with *dstString1* and ending with *dstStringm* . 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.

$$m = srcCode_2 - srcCode_1 + 1$$

NOTE If number of dstString in array is different from m, the result of mapping is undefined.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

10 Rendering

10.1 General

The PDF imaging model separates the specification of graphics objects (defining shapes and colours) from the process of rendering them on a *raster output device*.

NOTE 1 "Figure 20 — Colour specification" and "Figure 21 — Colour rendering" in clause 8.6.3, "Colour space families" illustrate this division.

Clause 10 defines how the shape and colour of graphics objects are rendered on a raster output device. Depending on the current colour space and on the characteristics of the raster output device, one or more of the following steps may be necessary:

- Convert colour that is not already in the native colour space of the raster output device, into that colour space; see 10.3, "CIE-Based colour to device colour" and 10.4, "Conversions among device colour spaces" for details.
- For any object for which transfer functions are in effect, apply those transfer functions; see 10.5, "Transfer functions" for details. Note that setting a transfer function in the graphic state using the **TR** or **TR2** keys in graphics state parameter dictionaries are deprecated in PDF 2.0.
- Perform a scan conversion algorithm to mark the appropriate pixels of the raster output device; see 10.7, "Scan conversion details" for details.
- If the raster output device supports PDF-defined halftoning, apply halftoning according to 10.6, "Halftones".

For the purpose of clause 10, it is irrelevant whether a raster output device physically exists and is actually used for rendering, or is just assumed (for example to simulate the colour appearance of PDF output on one raster output device by using some other raster output device).

NOTE 2 Any raster output device – printer or monitor – can in principle be used to simulate the colour appearance produced by some other raster output device. In this case, the output colour values in the colour space of the simulated raster output device become source colour values to be rendered on the simulating raster output device. Usually the simulating raster output device will have to be more capable regarding colour reproduction than the simulated device in order to be useful.

NOTE 3 Managing colour from digital description all the way to creating a rendered reproduction on a digital display or printed output is a complex topic. The International Color Consortium (ICC) (http://www.color.org) is a good source of information. The publication "Color management: understanding and using ICC profiles" provides a good introductory overview.

A PDF document 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 PDF processor:

- The media box and 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 document or provided separately, may specify detailed instructions for placing (imposing) PDF pages onto media and for controlling special features of the output device.

NOTE 4 A widely used standard for the format of vendor neutral job tickets is the JDF (Job Definition Format) described in the CIP4 document JDF Specification.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

10.2 Raster output device native colour

Each raster output device has a *native colour space*, which often matches one of the following process colour models: gray (monochrome), RGB (red-green-blue) or CMYK (cyan-magenta-yellow-black).

Process colours are ones that are produced by combinations of one or more process colourants. Colours specified in any device or CIE-based colour space shall be rendered as process colours.

A device may also support additional spot colourants, which shall be painted only by means of **Separation** or **DeviceN** colour spaces that define such spot colourants.

10.3 CIE-Based colour to device colour

10.3.1 General

To render CIE-based colours on a raster output device, it is necessary to convert from the specified CIE-based colour space to the device's native colour space taking into account the known properties of the raster output device.

For accurately rendering a CIE-based colour on a raster output device, a CIE-based *destination colour space* that is suitable for representing the native colour space of the raster output device (for example, by means of a monitor or printer ICC profile) needs to be established. The specific method by which the CIE-based destination colour space is established is beyond the scope of this document, but may include the use of Output Intents (see 14.11.5, "Output intents").

NOTE Establishing a CIE-based destination colour space for a raster output device can happen based on a user-driven configuration, by assumptions made by the PDF processor software, by a job ticket, by settings in an output device, by built-in colour measurements, or by other mechanisms.

Conversion from a CIE-based source colour to a CIE-based destination colour shall be performed based on ISO 15076-1:2010 (ICC.1:2010).

10.3.2 Establishing CIE-based colour space definitions

A PDF processor should establish CIE-based colour specifications for device colour spaces (**DeviceGray**, **DeviceRGB**, or **DeviceCMYK**), and thus implicitly remap device colour spaces into CIE-based colour spaces, when those device colour spaces do not match that of the raster output device. If the native device colour space is CMYK, then converting colours in the **DeviceGray** colour space to that CMYK should follow the method described in 10.4.2.3, "Conversion between DeviceGray and DeviceCMYK".

NOTE Establishing a CIE-based source colour space can happen based on a user-driven configuration, by assumptions made by the PDF processor software, by analysis of the colour values and other properties, or by other mechanisms.

10.4 Conversions among device colour spaces

10.4.1 General

This subclause describes the methods of classic colour conversion between **DeviceGray** and **DeviceRGB**, **DeviceGray** and **DeviceCMYK**, and **DeviceRGB** and **DeviceCMYK**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

10.4.2 Classic colour conversion methods

10.4.2.1 General

Although ICC enabled PDF processors should always follow the provisions and recommendations provided in 10.3, "CIE-Based colour to device colour", a less-capable PDF processor may choose to use the algorithms specified in the following subclauses 10.4.2.2 through 10.4.2.5. These algorithms are, however, very simple and as perceived by a human viewer they produce only crude approximations of the original colours.

10.4.2.2 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:

$$red = grey$$
$$green = grey$$
$$blue = grey$$

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:

$$gray = 0.3 \times red + 0.59 \times green + 0.11 \times blue$$

10.4.2.3 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

$$cyan = 0.0$$
$$magenta = 0.0$$
$$yellow = 0.0$$
$$black = 1.0 - grey$$

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 10.4.2.4, “Conversion from DeviceRGB to DeviceCMYK”.

10.4.2.4 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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:

$$\text{cyan} = 1.0 - \text{red}$$
$$\text{magenta} = 1.0 - \text{green}$$
$$\text{yellow} = 1.0 - \text{blue}$$

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 could be expected to 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.
- Undercolour 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 undercolour-removal functions, respectively:

$$c = 1.0 - red$$
$$m = 1.0 - green$$
$$y = 1.0 - blue$$
$$k = min\,(c, m, y)$$
$$cyan = min\big(1.0, max\big(0.0, c - UCR(k)\big)\big)$$
$$magenta = min\big(1.0, max\big(0.0, m - UCR(k)\big)\big)$$
$$yellow = min\big(1.0, max\big(0.0, y - UCR(k)\big)\big)$$
$$black = min\big(1.0, max\big(0.0, BG(k)\big)\big)$$

The black-generation and undercolour-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 57 — Entries in a graphics state parameter dictionary"). 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 undercolour-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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 undercolour-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 colourant.

The final component values that result after applying black generation and undercolour 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 undercolour-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, black point compensation and colour conversions" for further discussion of the role of black-generation and undercolour-removal functions in the transparent imaging model.

10.4.2.5 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:

$$red = 1.0 - min(1.0, cyan + black)$$
$$green = 1.0 - min\ (1.0, magenta + black)$$
$$blue = 1.0 - min(1.0, yellow + black)$$

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.

10.5 Transfer functions

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 57 — Entries in a graphics state parameter dictionary" however this is deprecated in PDF 2.0.
- The *current halftone* parameter in the graphics state may specify transfer functions as optional entries in *halftone dictionaries* (see 10.6.5, "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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

specified in a halftone dictionary shall override the corresponding one specified by the current transfer function parameter in the graphics state.

In the sequence of steps for processing colours, the PDF processor shall apply the transfer function after performing any needed conversions between colour spaces. If the output is to be halftoned the transfer function shall be applied before halftoning. 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 can 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.

Because transfer functions produce device-dependent effects, a page description that is intended to be device-independent should not define a current transfer function in the graphics state, or define **TransferFunction** in any halftone dictionaries.

When the current colour space is **DeviceGray** and the output device's native colour space is **DeviceCMYK**, a PDF processor 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 PDF processors 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

10.6 Halftones

10.6.1 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 minimise visual artifacts.

NOTE 3 In colour printing, independent halftone screens can be specified for each of several colourants.

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.

10.6.2 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 $i = floor\ (g \times n)$.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 10.5, "Transfer functions".

10.6.3 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 is determined by a spot function, which specifies that order in an indirect way that minimises 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. 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 centre 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 centre of the cell, producing a "line screen" in which the white pixels grow away from that line.

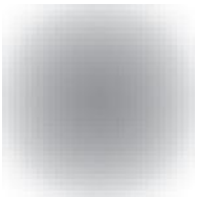
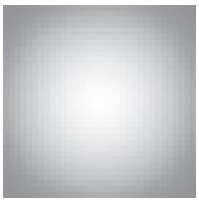
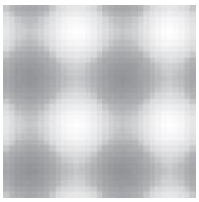
"Table 126 — Predefined spot functions" shows the predefined spot functions. The table gives the mathematical definition of each function along with the corresponding PostScript language code as it would be defined in a PostScript calculator function (see 7.10.5, "Type 4 (PostScript calculator) functions"). 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

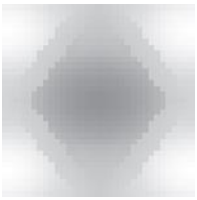
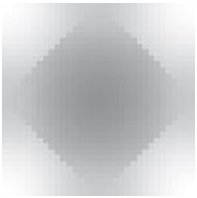
corresponding to darker points in the image are whitened later than those corresponding to lighter points.

Table 126 — 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 }
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 }

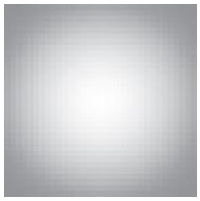
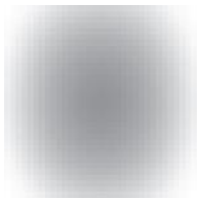
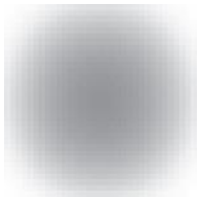
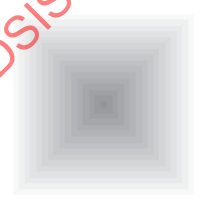
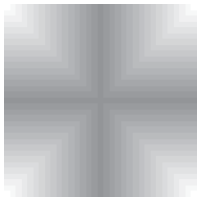
STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Appearance	Definition
Line		$-\lvert y\rvert$ { exch pop abs neg }
LineX		x { pop }
LineY		y { exch pop }
Round		$if\ \lvert x\rvert + \lvert y\rvert \leq 1\ then\ 1 - (x^2 + y^2)$ $else\ (\lvert x\rvert - 1)^2 + (\lvert y\rvert - 1)^2 - 1$ { 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		$let\ w = (3 \times \lvert x\rvert) + (4 \times \lvert y\rvert) - 3$ $if\ w < 0\ then\ 1 - \frac{x^2 + \left(\frac{\lvert y\rvert}{0.75}\right)^2}{4}$ $else\ if\ w > 1\ then \frac{(1 - \lvert x\rvert)^2 + \left(1 - \frac{\lvert y\rvert}{0.75}\right)^2}{4} - 1$ $else\ 0.5 - w$ { 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 }

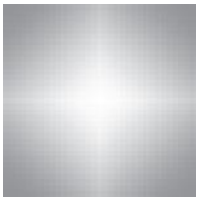
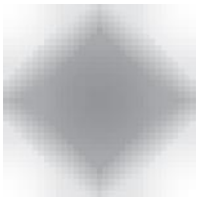
STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Appearance	Definition
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 }
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 }

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Appearance	Definition
Rhomboid		$\frac{0.9 \times \lvert x\rvert + \lvert y\rvert}{2}$ { abs exch abs 0.9 mul add 2 div }
Diamond		$if\ \lvert x\rvert + \lvert y\rvert \leq 0.75\ then\ 1 - (x^2 + y^2)$ $else\ if\ \lvert x\rvert + \lvert y\rvert \leq 1.23\ then\ 1 - (0.85 \times \lvert x\rvert + \lvert y\rvert)$ $else\ (\lvert x\rvert - 1)^2 + (\lvert y\rvert - 1)^2 - 1$ { 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 64 — Various halftoning effects" illustrates the effects of some of the predefined spot functions.

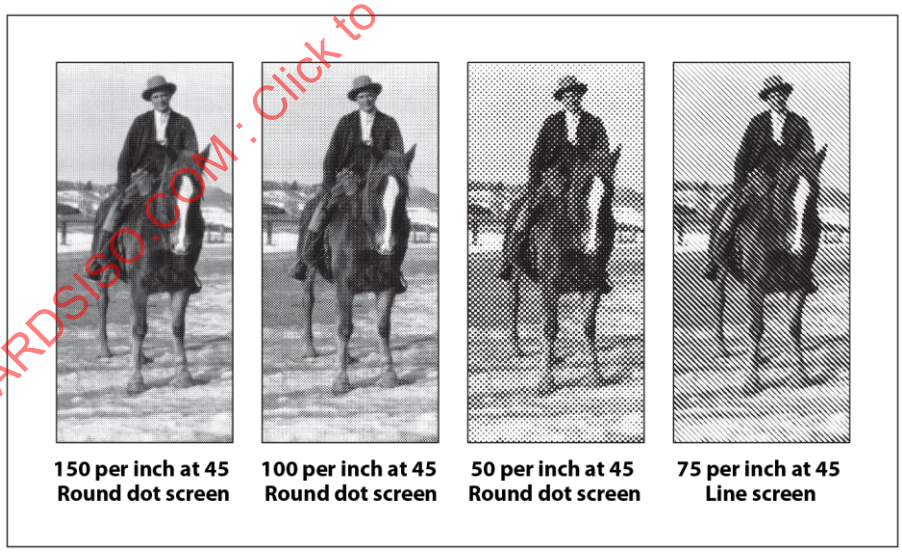


Figure 64 — Various halftoning effects

10.6.4 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.)

10.6.5 Halftone dictionaries

10.6.5.1 General

In PDF 1.2, the graphics state includes a current halftone parameter. A PDF processor may choose to use this halftone to perform painting operations if required for the current output device. Alternatively the PDF processor may choose to ignore some or all of the halftones specified in the PDF file and use different halftones that may be better suited for that device (see Annex N, "Best practice for halftones").

The current halftone may be specified as the value of the **HT** entry in a graphics state parameter dictionary; see "Table 57 — Entries in a graphics state parameter dictionary". 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.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 127 — PDF halftone types".

Table 127 — 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 colourant or colour component (including both primary and spot colourants). The keys in this dictionary are names of colourants; the values are halftone dictionaries of other types, each defining the halftone screen for a single colourant.
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 non-zero 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 non-zero screen angle.

NOTE 1 The dictionaries representing these halftone types contain the same entries as the corresponding PostScript language halftone dictionaries (as described in clause 7.4 of the PostScript Language Reference, Third Edition), with the following exceptions:

The PDF dictionaries can 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 language procedures. Threshold arrays are specified as streams instead of files.

In Type 5 halftone dictionaries, the keys for colourants shall be name objects; they cannot be strings as is allowed in PostScript.

Halftone dictionaries have an optional entry, **HalftoneName**, that identifies the halftone by name. 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 may 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 may be defined by the other entries in the dictionary, if any. If no other entries are present, the default halftone may be used.

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 halftones in the transparent imaging model.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

10.6.5.2 Type 1 halftones

"Table 128 — Entries in a Type 1 halftone dictionary" 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 128 — 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, name or array	(*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 a spot function, or an array of names of spot functions (*PDF 2.0*). A name should be one of the predefined spot functions (see "Table 126 — Predefined spot functions"). If the value of **SpotFunction** is an array the PDF processor shall use the first name within the array that it recognizes. If a name is provided that is not defined in "Table 126 — Predefined spot functions" and the PDF processor does not understand that name, or if none of the names in an array are defined in "Table 126 — Predefined spot functions" or are understood by the PDF processor, the processor shall use the default halftone.
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*.
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 10.6.5.6, "Type 5 halftones") and represents either a nonprimary or nonstandard primary colour component (see 10.5, "Transfer functions"). The name *Identity* may be used to specify the identity function. The **TransferFunction** key should only be used to convey tone reproduction compensation (sometimes called gamma curve correction); it should not be used to specify artistic intent. If the key is required by some other clause in this document but no compensation is necessary then the value should be the name *Identity*.

If the **AccurateScreens** entry has a value of *true*, a highly precise halftoning algorithm shall be

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 quantised to device pixels. High accuracy is important mainly for making colour separations on high-resolution devices. However, it ican 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 unfavourable angles can 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
```

10.6.5.3 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 129 — Additional entries specific to a Type 6 halftone dictionary" in addition to the usual entries common to all streams (see "Table 5 — Entries common to all stream dictionaries"). The **Width** and **Height** entries shall specify the dimensions of the threshold array in device pixels; the stream shall contain $Width \times Height$ 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 49 — Source image coordinate system"), with the first value at device coordinates (0, 0) and horizontal coordinates changing faster than vertical coordinates.

Table 129 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. The name *Identity* may be used to specify the identity function (see 10.5, "Transfer functions"). NOTE PDF versions to 1.7 required that this entry be present if the dictionary is a component of a Type 5 halftone (see 10.6.5.6, "Type 5 halftones") and represents either a non-primary or non-standard primary colour component.

10.6.5.4 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 non-zero screen angles as well.

Halftone cells at non-zero 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 65 — Halftone cell with a non-zero angle" 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.

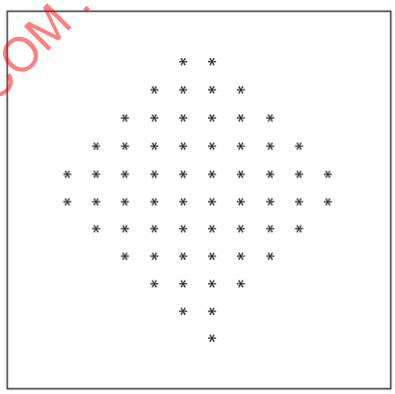


Figure 65 — Halftone cell with a non-zero angle

NOTE 2 "Figure 66 — Angled halftone cell divided into two squares" 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 67 — Halftone cell and two squares tiled across device space"). The last row in the first square is immediately adjacent to the first row in the second square and starts in the same column.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

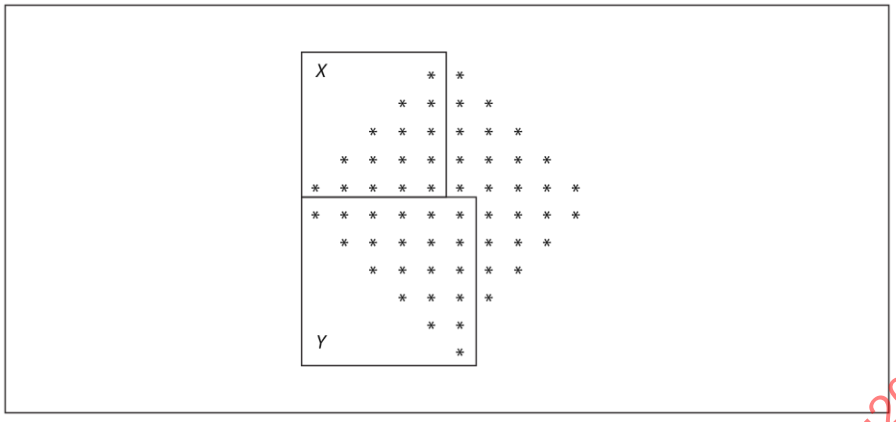


Figure 66 — Angled halftone cell divided into two squares

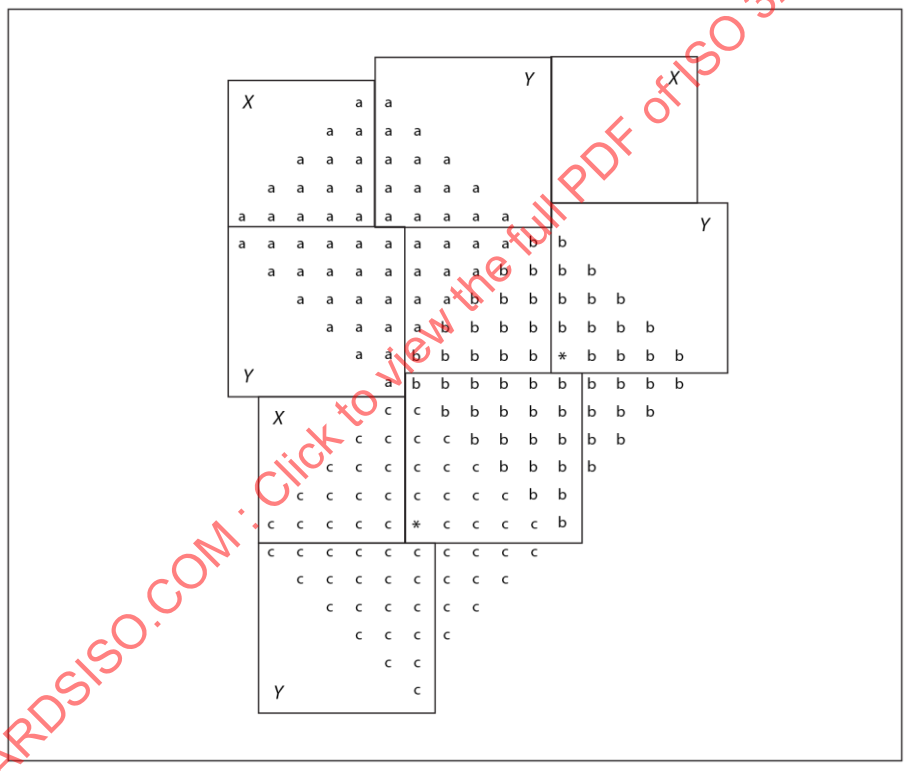


Figure 67 — Halftone cell and two squares tiled across device space

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 67 — Halftone cell and two squares tiled across device space". 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

$$\text{frequency} = \frac{\text{resolution}}{\sqrt{X^2 + Y^2}}$$

and the angle by

$$\text{angle} = atan\left(\frac{Y}{X}\right)$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 130 — Additional entries specific to a Type 10 halftone dictionary" in addition to the usual entries common to all streams (see "Table 5 — Entries common to all stream dictionaries"). The **Xsquare** and **Ysquare** entries replace the Type 6 halftone's **Width** and **Height** entries.

Table 130 — 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 10.6.5.6, "Type 5 halftones") and represents either a nonprimary or nonstandard primary colour component (see 10.5, "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 49 — Source image coordinate system"), with the first value at device coordinates (0, 0) and horizontal coordinates changing faster than vertical coordinates.

10.6.5.5 Type 16 halftones

Like Type 10, a Type 16 halftone (*PDF 1.3*) defines a halftone screen with a threshold array and allows non-zero 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 68 — Tiling of device space in a Type 16 halftone". 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

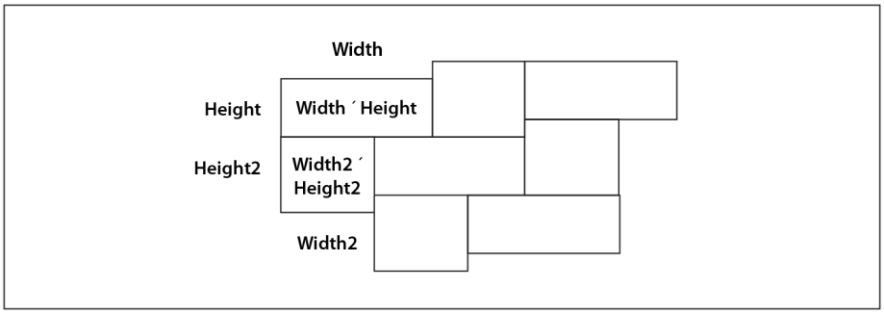


Figure 68 — Tiling of device space in a Type 16 halftone

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 131 — Additional entries specific to a Type 16 halftone dictionary" in addition to the usual entries common to all streams (see "Table 5 — Entries common to all stream dictionaries"). 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 49 — Source image coordinate system"), 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 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 10.6.5.6, "Type 5 halftones") and represents either a nonprimary or nonstandard primary colour component (see 10.5, "Transfer functions"). The name *Identity* may be used to specify the identity function.

10.6.5.6 Type 5 halftones

Some devices, particularly colour printers, require separate halftones for each individual colourant. 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 colourants or colour components.

A Type 5 halftone dictionary (see "Table 132 — Entries in a Type 5 halftone dictionary") is a composite dictionary containing independent halftone definitions for multiple colourants. Its keys shall be name objects representing the names of individual colourants 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 colourant or colour component. The component halftone dictionaries shall not be of halftone Type 5.

Table 132 — 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	integer	(*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 colourant name	dictionary or stream	(*Required, one per colourant*) The halftone corresponding to the colourant or colour component named by the key. The halftone may be of any Type other than 5.
Default	dictionary or stream	(*Required*) A halftone that shall be used for any colourant or colour component that does not have an entry of its own. The value shall not be 5. If there are any nonprimary colourants, the default halftone shall have a transfer function.

The colourants 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**;).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- Nonstandard colour components for use as spot colourants in **Separation** and **DeviceN** colour spaces. Some of these may also be used as process colourants 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 colourant 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 colourants 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 colourants.

EXAMPLE 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
       >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
endobj

35 0 obj
       <</Type /Halftone
         /HalftoneType 1
         /Frequency 90.0
         /Angle 45
         /SpotFunction /Round
         /AccurateScreens true
      >>
endobj
```

10.7 Scan conversion details

10.7.1 General

The final step of rendering shall be *scan conversion*. The PDF processor 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 can perform scan conversion in different ways; techniques that are appropriate for one device could 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.

10.7.2 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 69 — Flatness tolerance". Flatness may be specified as the operand of the **i** operator (see "Table 56 — Graphics state operators") or as the value of the **FL** entry in a graphics state parameter dictionary (see "Table 57 — Entries in a graphics state parameter dictionary"). It shall be a positive number.

PDF processors may choose to ignore any flatness tolerance specified within a PDF file.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

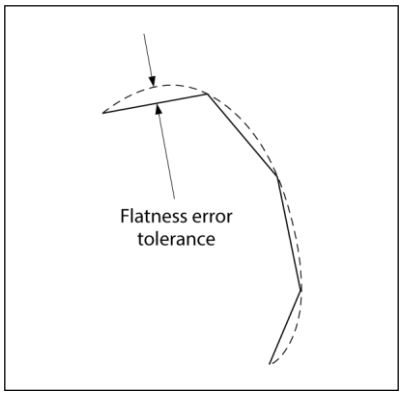


Figure 69 — Flatness tolerance

10.7.3 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 57 — Entries in a graphics state parameter dictionary").

Each output device may have internal limits on the maximum and minimum tolerances attainable. Setting smoothness to 1.0 can result in an internal smoothness of 0.5 on a high-quality colour device, although setting it to 0.0 on the same device can result in an internal smoothness of 0.01 if an error of that magnitude is imperceptible on the device.

NOTE 1 The smoothness tolerance can 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 can 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 will normally 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.

10.7.4 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

real-number coordinates are (x, y), let i = floor(x) and j = 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 (8.5.3.3, "Filling") and stroking (8.5.3.2, "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 half-open 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 non-zero 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 PDF processor 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. If the resolution of the source image is higher than that of device space, some source samples might 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 2 Font rendering algorithms use hints in the glyph descriptions and techniques that are specialised to glyph rasterization.

10.7.5 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 quantised to device pixels. A line of a given width can intersect with different numbers of device pixels, depending on where it is positioned. "Figure 70 — Rasterization without stroke adjustment" 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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.

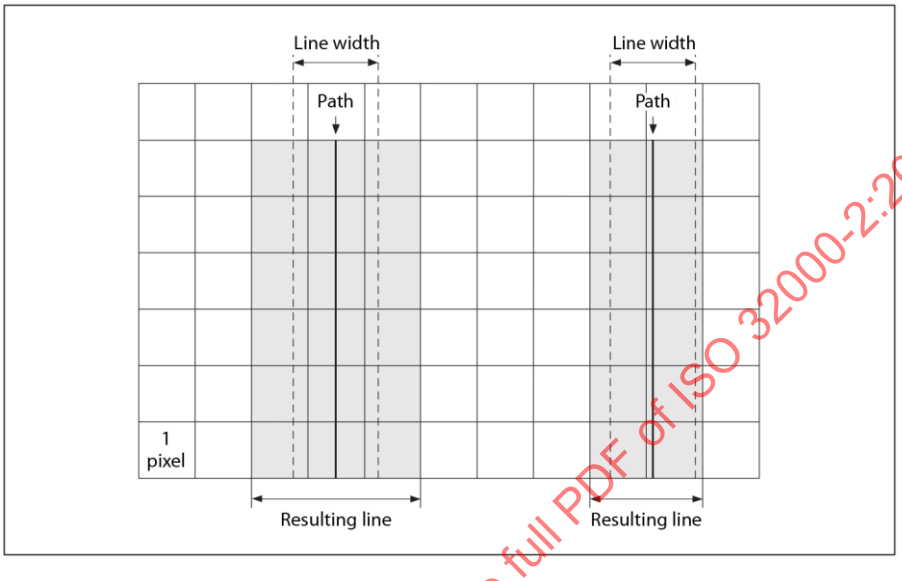


Figure 70 — Rasterization without stroke adjustment

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 10.7.4, "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").

10.8 Rendering for separations

10.8.1 General

A device onto which the PDF is being rendered has a set of colourants known by the renderer, the knowledge of which is needed to produce the desired output. If those are subtractive colourants (see 8.6.4, "Device colour spaces") then the output for the device may take a form called *separations* where one output for each device colourant is produced separately, usually as bitmaps. Whether separations are produced is up to the processing software.

10.8.2 Separations

Certain exceptions to the standard opaque and transparent imaging models are supported when preparing device separations controlled by the overprint controls (see 8.6.7, "Overprint control"). For example, using a **Separation** or **DeviceN** colour space, ink values may be introduced into one separation without affecting any of the other separations. In this way "overprinting" of colourants can

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

be controlled very explicitly; thus providing detailed control when producing output that will be used to control the colourant printing stations on a printing press.

Alternate colour spaces are supplied for **DeviceN** and **Separation** colour spaces so that files prepared for generation of separations can be displayed on other devices and, in many cases, the overprint controls are ignored. This can result in dramatically different colours in some cases. For example, if two separate painting operations are performed on the same area of the page with the overprinting controls turned on, one using a Cyan **Separation** colour space and the second using a Yellow **Separation** colour space, then that area will appear green when produced using separations, overprinting of the cyan and yellow inks, whereas displaying it on the screen, ignoring the overprint controls, will generally produce yellow, the last colour painted.

10.8.3 Separation simulation

If it is important for the colours of the display for a PDF, on a device that normally would not be used to produce separations, to more closely match those produced when using separations, then a simulation of the separation process can be performed for the output to the non-separation device. Such a simulation is defined here.

The results of the simulation should match those produced by the following steps:

a) Process the PDF as if separations were to be created for a simulated device that supports subtractive process colourants and possibly spot colours. The PDF processor determines what process colours and possible spot colours the simulated device is to have. A default **DestOutputProfile**, if available for a subtractive device, or **ColorantTable** values, if available for a subtractive device, should be consulted to determine the process colours to use (see 14.11.5, "Output intents").

b) Convert each separation into "flat XYZ" (no gamma) and using a background matte of all white.

c) Blend the resulting separations into a single result using a multiply blend (see "Table 133 — Variables used in the basic compositing formula").

d) Convert the result to the actual device colour space and output it.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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, but a processor need not implement these exactly provided the results are equivalent.

NOTE Transparency was added to PDF in version 1.4.

The following 6 clauses are organised 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 — realised 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.

In the transparent imaging model, all of the objects on a page may potentially contribute to the result. Objects at a given point form a *transparency stack* (or *stack* for short). The objects are 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 PDF processor 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Annex Q, "Method for determining transparency on a page" defines a standardised method for determining if transparency is present on a page. This method is not required by this document, however other PDF-related standards may wish to mandate the use of this algorithm to ensure consistent behaviour.

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 11.4, "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 could 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 the group backdrop. An additional blend mode, blending colour space, shape, and opacity may also be associated with the group as a whole and used when compositing the objects inside of the group as well as the group itself with the group backdrop. Groups may be nested within other groups, forming a tree-structured hierarchy.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

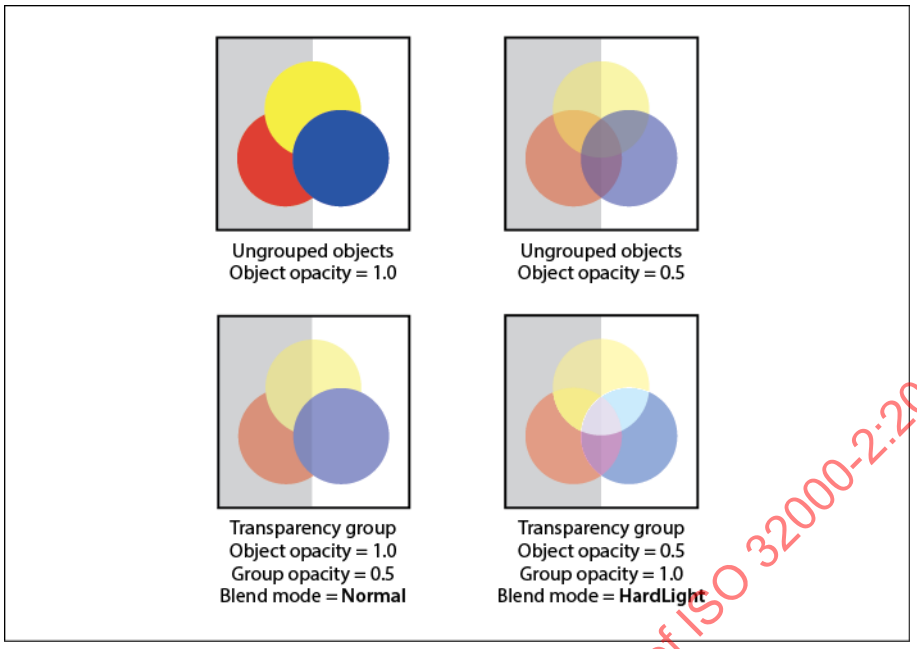


Figure 71 — Transparency groups

EXAMPLE "Figure 71 — Transparency groups" 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 8.5.4, "Clipping path operators").

11.3 Basic compositing computations

11.3.1 General

This subclause 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;

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 "opaqueness")—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 although 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. In addition, the convention that $0 \div 0 = 0$ shall 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 subclause 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:

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times \left((1 - \alpha_b) \times C_s + \alpha_b \times B(C_b, C_s)\right)$$

where the variables have the meanings shown in "Table 133 — Variables used in the basic compositing formula".

NOTE 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, extended to include a blend mode in the region of overlapping coverage.

Table 133 — Variables used in the basic compositing formula

Variable	Meaning
C_b	Backdrop colour
C_s	Source colour
C_r	Result colour
α_b	Backdrop alpha
α_s	Source alpha
α_r	Result alpha

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Variable	Meaning
$B(C_b, C_s)$	Blend function

The following subclauses 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 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. The backdrop and source colours are normally converted to the blending colour space before the compositing computation.

Of the PDF colour spaces described in 8.6, "Colour spaces", the following shall be supported as blending colour spaces:

- **DeviceGray**
- **DeviceRGB**
- **DeviceCMYK**
- **CalGray**
- **CalRGB**
- **ICCBased** bi-directional 'GRAY', 'RGB ', and 'CMYK' colour spaces

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 bi-directional; that is, the ICC profile shall be capable of both device to PCS and PCS to device transformations.

When performing blending, process colours (see 10.2, "Raster output device native colour") shall be converted to the blending colour space. 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 8.6.6.4, "Separation colour spaces" and 8.6.6.5, "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**, **ICCBased** 'CMYK', **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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

11.3.5 Blend mode

11.3.5.1 General

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 customise the blending operation. PDF defines a standard set of named blend functions, or blend modes, listed in "Table 134 — Standard separable blend modes" and "Table 135 — Standard non-separable blend modes". "Figure 72 — RGB blend modes" and "Figure 73 — CMYK blend modes" illustrate the resulting visual effects for *RGB* and *CMYK* colours, respectively.

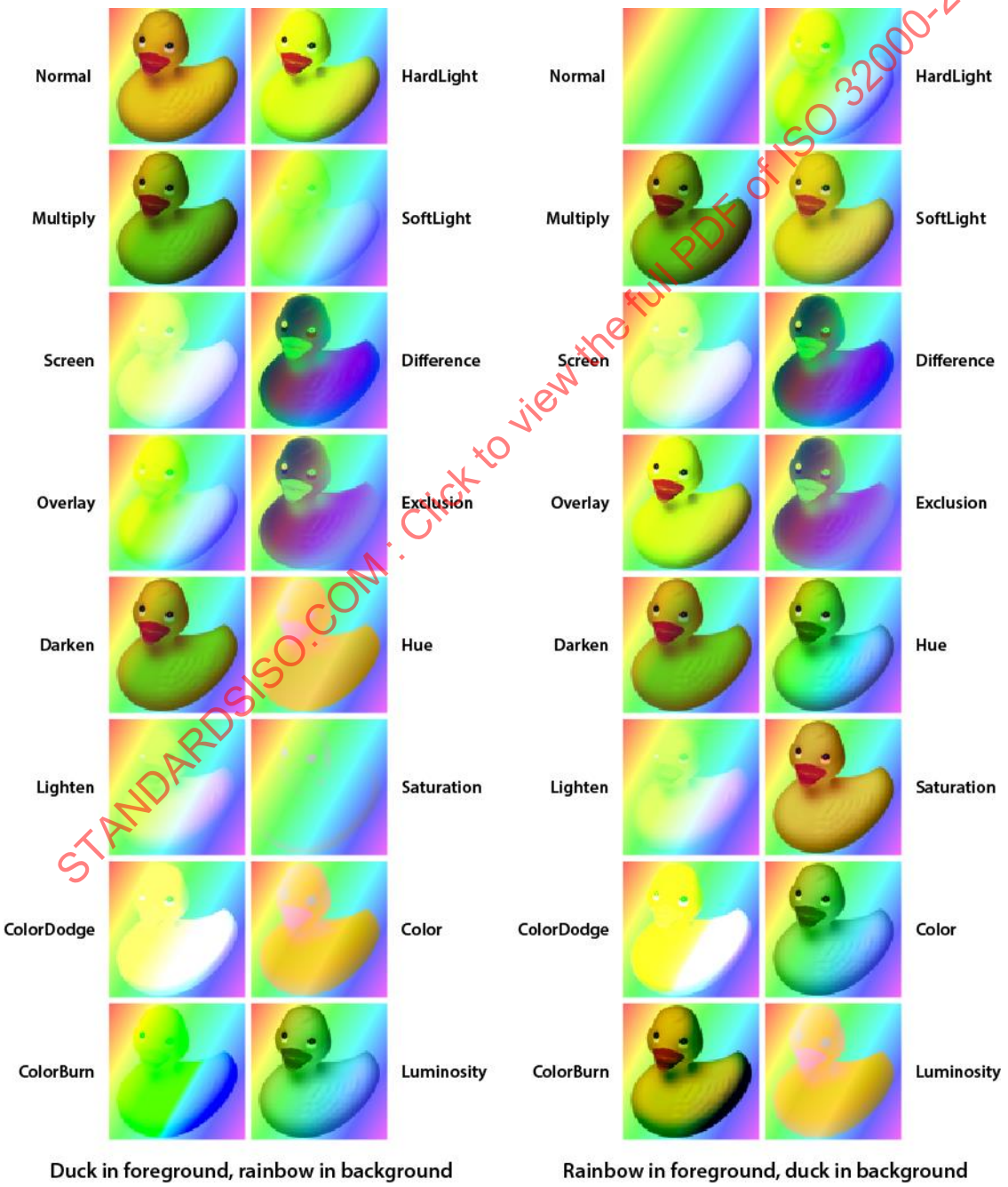
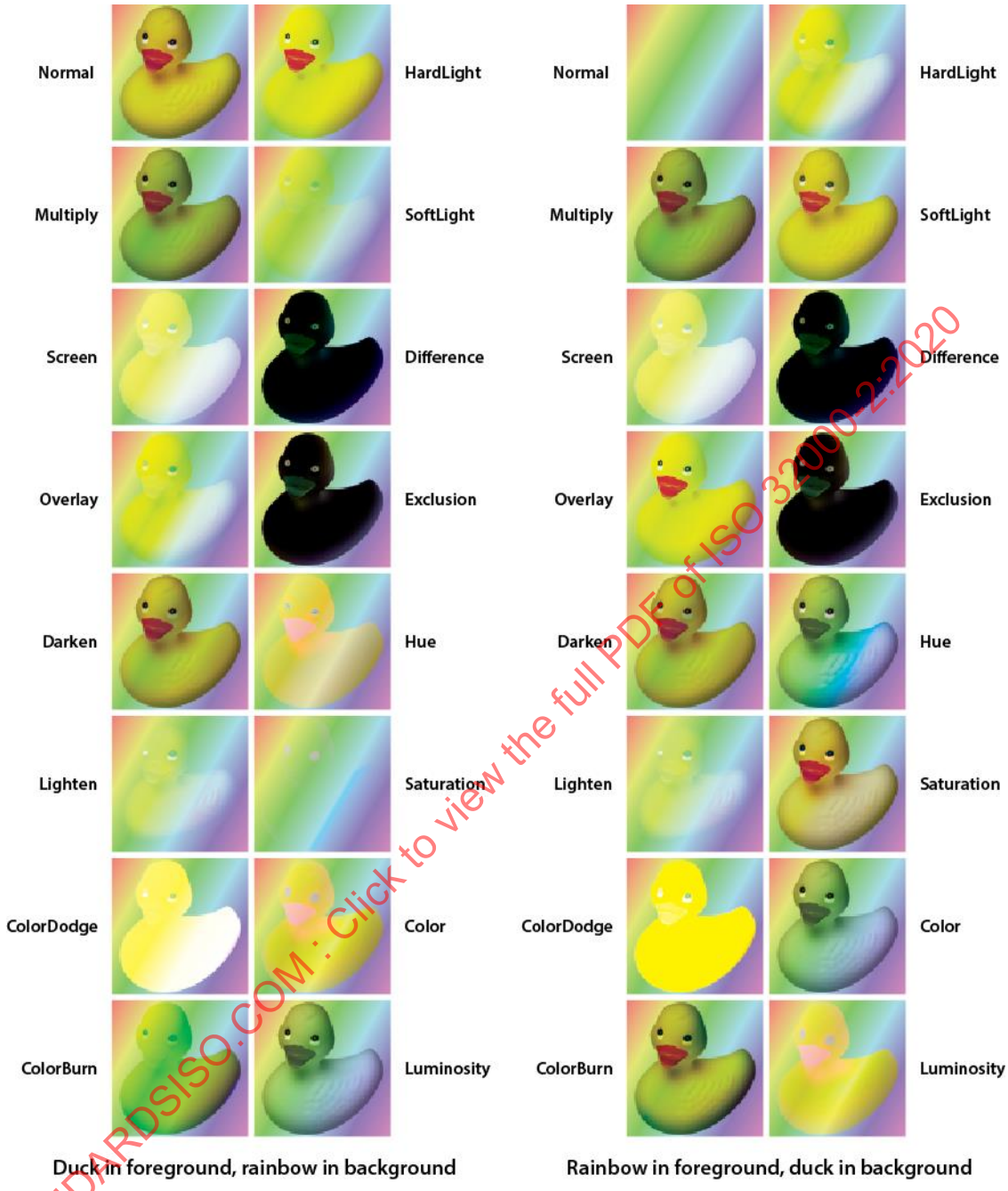


Figure 72 — RGB blend modes

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved



STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Figure 73 — CMYK blend modes

11.3.5.2 Separable blend modes

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

© ISO 2020 – All rights reserved

applies independently to any number of components. Only white-preserving separable blend modes shall be used for blending spot colours as discussed in 11.7.3, "Spot colours and transparency".

NOTE 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 134 — Standard separable blend modes" lists the standard separable blend modes available in PDF and the algorithms/formulas that shall be used in the calculation of blended colours.

Table 134 — Standard separable blend modes

Name	Result
Normal	$B(c_b, c_s) = c_s$ NOTE 1 Selects the source colour, ignoring the backdrop.
Compatible	(*Deprecated in PDF 2.0*) Same as **Normal**. This mode was introduced in an earlier PDF version and shall not be used by PDF writers.
Multiply	$B(c_b, c_s) = c_b \times c_s$ NOTE 2 Multiplies the backdrop and source colour values. NOTE 3 The result colour is always at least as dark as either of the two constituent colours. When working with additive colours, multiplying any colour with black produces black while multiplying with white leaves the original colour unchanged. For subtractive colours, the maximum tint value used for all colourants of the colour space acts as black does for additive spaces. Painting successive overlapping objects with a colour other than black or white produces progressively darker colours.
Screen	$B(c_b, c_s) = 1 - \big((1 - c_b) \times (1 - c_s)\big) = c_b + c_s - (c_b \times c_s)$ NOTE 4 Multiplies the complements of the backdrop and source colour values, then complements the result. NOTE 5 The result colour is always at least as light as either of the two constituent colours. When working with additive colours, screening any colour with white produces white while screening with black leaves the original colour unchanged. For subtractive colours, the maximum tint value of all colourants of the color space acts as black does for additive spaces. The effect is similar to projecting multiple photographic slides simultaneously onto a single screen.
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Result
ColorDodge	$B(c_b, c_s) = \begin{cases} 0 & \text{if } c_b = 0 \\ 1 & \text{if } c_b \geq 1 - c_s \\ c_b/(1 - c_s) & \text{otherwise} \end{cases}$ NOTE 10 Brightens the backdrop colour to reflect the source colour. Painting with black produces no change. NOTE 11 This function is formulated in a different way here than it is in ISO 32000-1:2008. However, it produces the same results except in one special edge case. For **ColorDodge**, the special case is $c_b = 0$ and $c_s = 1$, where the result is now 0 instead of 1. The rationale for the change is that for any given c_b, the result should be a continuous function of c_s.
ColorBurn	$B(c_b, c_s) = \begin{cases} 1 & \text{if } c_b = 1 \\ 0 & \text{if } 1 - c_b \geq c_s \\ 1 - ((1 - c_b)/c_s) & \text{otherwise} \end{cases}$ NOTE 12 Darkens the backdrop colour to reflect the source colour. Painting with white produces no change. NOTE 13 This function is formulated in a different way here than it is in ISO 32000-1:2008. However, it produces the same results except in one special edge case. For **ColorBurn**, the special case is $c_b = 1$ and $c_s = 0$, where the result is now 1 instead of 0. The rationale for the change is that for any given c_b, the result should be a continuous function of c_s.
HardLight	$B(c_b, c_s) = \begin{cases} \text{Multiply}(c_b, 2 \times c_s) & \text{if } c_s \leq 0.5 \\ \text{Screen}(c_b, 2 \times c_s - 1) & \text{if } c_s > 0.5 \end{cases}$ NOTE 14 Multiplies or screens the colours, depending on the source colour value. The effect is similar to shining a harsh spotlight on the backdrop.
SoftLight	$B(c_b, c_s) = \begin{cases} c_b - (1 - 2 \times c_s) \times c_b \times (1 - c_b) & \text{if } c_s \leq 0.5 \\ c_b + (2 \times c_s - 1) \times (D(c_b) - c_b) & \text{if } c_s > 0.5 \end{cases}$ $D(x) = \begin{cases} ((16 \times x - 12) \times x + 4) \times x & \text{if } x \leq 0.25 \\ \sqrt{x} & \text{if } x > 0.25 \end{cases}$ NOTE 15 Darkens or lightens the colours, depending on the source colour value. The effect is similar to shining a diffused spotlight on the backdrop.
Overlay	$B(c_b, c_s) = \text{HardLight}(c_s, c_b)$ NOTE 16 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.
Difference	$B(c_b, c_s) = \lvert c_b - c_s \rvert$ NOTE 17 Subtracts the darker of the two constituent colours from the lighter colour: Painting with white inverts the backdrop colour; painting with black produces no change. For subtractive colours, the maximum tint value for all colourants of the colour space acts as black does for additive spaces. NOTE 18 This blend mode is not white-preserving.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Name	Result
Exclusion	$B(c_b, c_s) = c_b + c_s - 2 \times c_b \times c_s$ NOTE 19 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. For subtractive colours, the maximum tint value for all colourants of the colour space acts as black does for additive spaces. NOTE 20 This blend mode is not white-preserving.

11.3.5.3 Non-separable blend modes

"Table 135 — Standard non-separable blend modes" lists the standard non-separable blend modes. Since the non-separable 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 colour spaces that are allowed as blending colour spaces (see 11.3.4, "Blending colour space").

All of the non-separable blend modes conceptually entail the following steps:

- Convert the backdrop and source colours from the blending colour space to an intermediate HSL (hue-saturation-luminosity) representation.
- Create a new colour from some combination of hue, saturation, and luminosity components selected from the backdrop and source colours.
- Convert the result back to the (blending) colour space.

However, the following formulas 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 non-separable 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 in gray colour spaces (**DeviceGray**, **CalGray** and **ICCBased** gray) shall be done by conversion to RGB, blending in RGB, and then converting back to gray. Blending of *CMYK* colour spaces requires special treatment, and is described later in this subclause. The following formulas apply to RGB spaces (including **DeviceRGB**, **CalRGB** and **ICCBased** 'RGB '):

These functions shall have the following definitions:

$$Lum(C) = 0.3 \times C_{red} + 0.59 \times C_{green} + 0.11 \times C_{blue}$$

$SetLum(C, l)$

$let\ d = l - Lum(C)$

$C_{red} = C_{red} + d$

$C_{green} = C_{green} + d$

$C_{blue} = C_{blue} + d$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$return\ ClipColor(C)$

$ClipColor(C)$

$let\ l = Lum(C)$

$let\ n = min(C_{red}, C_{green}, C_{blue})$

$let\ x = max(C_{red}, C_{green}, C_{blue})$

$if\ n < 0.0$

$C_{red} = l + \left(((C_{red} - l) \times l)/(l - n)\right)$

$C_{green} = l + \left(\left((C_{green} - l) \times l\right)/(l - n)\right)$

$C_{blue} = l + \left(((C_{blue} - l) \times l)/(l - n)\right)$

$endif$

$if\ x > 1.0$

$C_{red} = l + \left(((C_{red} - l) \times (1 - l))/(x - l)\right)$

$C_{green} = l + \left(\left((C_{green} - l) \times (1 - l)\right)/(x - l)\right)$

$C_{blue} = l + \left(((C_{blue} - l) \times (1 - l))/(x - l)\right)$

$endif$

$return\ C$

$Sat(C) = max(C_{red}, C_{green}, C_{blue}) - min(C_{red}, C_{green}, C_{blue})$

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.

$SetSat(C, s)$

$if\ C_{max} > C_{min}$

$C_{mid} = \left(((C_{mid} - C_{min}) \times s)/(C_{max} - C_{min})\right)$

$C_{max} = s$

$else$

$C_{mid} = C_{max} = 0.0$

$endif$

$C_{min} = 0.0$

$return\ C$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE The *SetSat* algorithm has a distinct discontinuity between neutral and near neutral colours which makes it very sensitive to variations between implementations with respect to colour management and quantisation of colour values. Therefore the visual appearance of elements can be very different between different PDF processors used for viewing or printing pages containing elements that use the Hue and Saturation blend modes that use the SetSat algorithm.

Table 135 — Standard non-separable blend modes

Name	Result
Hue	$B(C_b, C_s) = SetLum\left(SetSat(C_s, Sat(C_b)), Lum(C_b)\right)$ NOTE 1 Creates a colour with the hue of the source colour and the saturation and luminosity of the backdrop colour.
Saturation	$B(C_b, C_s) = SetLum\left(SetSat(C_b, Sat(C_s)), Lum(C_b)\right)$ 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.
Color	$B(C_b, C_s) = SetLum(C_s, Lum(C_b))$ 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	$B(C_b, C_s) = SetLum(C_b, Lum(C_s))$ 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The formulas in this subclause apply to *RGB* spaces (including **DeviceRGB**, **CalRGB** and **ICCBased** RGB). Blending in *CMYK* spaces (including both **DeviceCMYK** and **ICCBased** *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 by subtracting each from 1.0. The formulae in this subclause shall be applied to the *RGB* colour values. The results shall be complemented back to *C*, *M* and *Y* in the same way.
- 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

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times \left((1 - \alpha_b) \times C_s + \alpha_b \times B(C_b, C_s)\right)$$

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_b and α_s.

NOTE 1 This formula represents a simplification of the following formula, which presents the relative contributions of backdrop, source, and blended colours in a more straightforward way:

© ISO 2020 – All rights reserved

$$\alpha_r \times C_r = \left((1 - \alpha_s) \times \alpha_b \times C_b\right) + \left(\left((1 - \alpha_b) \times \alpha_s \times C_s\right) + \left(\alpha_b \times \alpha_s \times B(C_b, C_s)\right)\right)$$

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 realised or is toned down by mixing the result with the backdrop and source colours.

The result alpha, α_r, actually represents a computed result, described in 11.3.7, "Shape and opacity computations". The result colour shall be normalised 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 2 If α_r is zero, the result colour is undefined.

The simplification requires a substitution based on the alpha compositing formula, which is presented in 11.3.7.3, "Result shape and opacity". 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 $\alpha_b = 0.0$ results in no blend mode effect; setting $\alpha_s = 1.0$ and $\alpha_b = 1.0$ 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:

$$\alpha_b = f_b \times q_b$$
$$\alpha_r = f_r \times q_r$$
$$\alpha_s = f_s \times q_s$$

This subclause 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 11.6.4, "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 8.9.6.3, "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 (as defined in 11.3.7.3, "Result shape and opacity") 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

the boundaries can 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 136 — Variables used in the source shape and opacity formulas".

Table 136 — 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Variable	Meaning
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 11.6.7, "Patterns and transparency").

11.3.7.3 Result shape and opacity

In addition to a result colour, the painting operation shall also compute an associated result shape and result opacity. These computations shall be based on the union function

$$\begin{aligned}\text{Union}(b, s) &= 1 - \big((1 - b) \times (1 - s)\big) \\ &= b + s - (b \times s)\end{aligned}$$

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 137 — Variables used in the result shape and opacity formulas".

Table 137 — 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 (normalised by) the result shape.

NOTE 2 Since alpha is just the product of shape and opacity, it can easily be shown that

$$\alpha_r = Union(\alpha_b, \alpha_s)$$

This formula can be used whenever the independent shape and opacity are not needed.

11.3.8 Summary of basic compositing computations

This subclause is a summary of all the computations presented in subclause 11.3.1 through subclause 11.3.7. 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 "Table 133 — Variables used in the basic compositing formula", "Table 136 — Variables used in the source shape and opacity formulas", and "Table 137 — Variables used in the result shape and opacity formulas" for the meanings of the variables used in these formulas.

$$\begin{aligned}\text{Union}(b,s) &= 1 - \big((1-b)\times(1-s)\big)\\ &= b + s - (b \times s)\end{aligned}$$

$$f_s = f_j \times f_m \times f_k$$
$$q_s = q_j \times q_m \times q_k$$
$$f_r = \text{Union}(f_b, f_s)$$

$$\alpha_b = f_b \times q_b$$
$$\alpha_s = f_s \times q_s$$
$$\alpha_r = \text{Union}(\alpha_b, \alpha_s)$$

$$q_r = \frac{\alpha_r}{f_r}$$

$$C_r = \left(1 - \frac{\alpha_s}{\alpha_r}\right) \times C_b + \frac{\alpha_s}{\alpha_r} \times \big((1-\alpha_b)\times C_s + \alpha_b \times B(C_b, C_s)\big)$$

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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. An isolated group may specify its own blending colour space, independent of that of the group's backdrop.
- The group may be *knockout* or *non-knockout*, which shall determine whether the objects within the group stack are composited with one another or only with 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 11.5, "Soft masks").

11.4.2 Notation for group compositing computations

This subclause introduces some notation for dealing with group compositing. Subsequent subclauses describe the group compositing formulas for a non-isolated, non-knockout group and the special properties of isolated and knockout groups.

Since multiple objects are being dealt with 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_b, q_b, α_b, C_r, f_r, q_r, and α_r; 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, ..., n:

$$f_{s_i} = f_{j_i} \times f_{m_i} \times f_{k_i}$$
$$q_{s_i} = q_{j_i} \times q_{m_i} \times q_{k_i}$$

$$\alpha_{s_i} = f_{s_i} \times q_{s_i}$$
$$\alpha_i = \text{Union}(\alpha_{i-1}, \alpha_{s_i})$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

$$f_i = \text{Union}(f_{i-1}, f_{s_i})$$
$$q_i = \frac{\alpha_i}{f_i}$$

$$C_i = \left(1 - \frac{\alpha_{s_i}}{\alpha_i}\right) \times C_{i-1} + \frac{\alpha_{s_i}}{\alpha_i} \times \left((1 - \alpha_{i-1}) \times C_{s_i} + \alpha_{i-1} \times B_i(C_{i-1}, C_{s_i})\right)$$

where the variables have the meanings shown in "Table 138 — Revised variables for the basic compositing formulas".

NOTE Compare these formulas with those shown in 11.3.8, "Summary of basic compositing computations".

Table 138 — 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*
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*
$\mathrm{B_i(C_{i-1}, C_{s_i})}$	Blend function for object *i*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF processor 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:

$$\langle C, f, \alpha \rangle = Composite(C_0, \alpha_0, G)$$

where the variables have the meanings shown in "Table 139 — Arguments and results of the group compositing function".

Table 139 — Arguments and results of the group compositing function

Variable	Meaning
G	The transparency group: a compound object consisting of all elements E_1,..., E_n of the group — the n constituent objects' colours, shapes, opacities, and blend modes

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Variable	Meaning
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 subclause. 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 $E_i \in G$ $(i = 1, \ldots, n)$:

  $$\langle C_{s_i}, f_{j_i}, \alpha_{j_i} \rangle = \begin{cases} \text{Composite}(C_{i-1}, \alpha_{i-1}, E_i) & \text{if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape} \times \text{opacity) of } E_i & \text{otherwise} \end{cases}$$

  $$f_{s_i} = f_{j_i} \times f_{m_i} \times f_{k_i}$$
  $$\alpha_{s_i} = \alpha_{j_i} \times (f_{m_i} \times q_{m_i}) \times (f_{k_i} \times q_{k_i})$$

  $$f_{g_i} = \text{Union}(f_{g_{i-1}}, f_{s_i})$$
  $$\alpha_{g_i} = \text{Union}(\alpha_{g_{i-1}}, \alpha_{s_i})$$
  $$\alpha_i = \text{Union}(\alpha_0, \alpha_{g_i})$$

  $$\mathrm{C_i} = \left(1 - \frac{\alpha_{\mathrm{s_i}}}{\alpha_{\mathrm{i}}}\right) \times \mathrm{C_{i-1}} + \frac{\alpha_{\mathrm{s_i}}}{\alpha_{\mathrm{i}}} \times \left((1 - \alpha_{\mathrm{i-1}}) \times \mathrm{C_{s_i}} + \alpha_{\mathrm{i-1}} \times \mathrm{B_i}(\mathrm{C_{i-1}}, \mathrm{C_{s_i}})\right)$$

- Result:

  $$C = C_n + (C_n - C_0) \times \left(\frac{\alpha_0}{\alpha_{g_n}} - \alpha_0\right)$$
  $$f = f_{g_n}$$
  $$\alpha = \alpha_{g_n}$$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

where the variables have the meanings shown in "Table 140 — Variables used in the group compositing formulas" (in addition to those in "Table 139 — Arguments and results of the group compositing function").

For an element E_i that is an elementary object, the colour, shape, and alpha values C_{s_i}, f_j, 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 140 — 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
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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause.

NOTE 3 Special attention is 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 can be 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 11.4.6, "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 the group backdrop.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

lower in the group's stack but not on the group's backdrop.

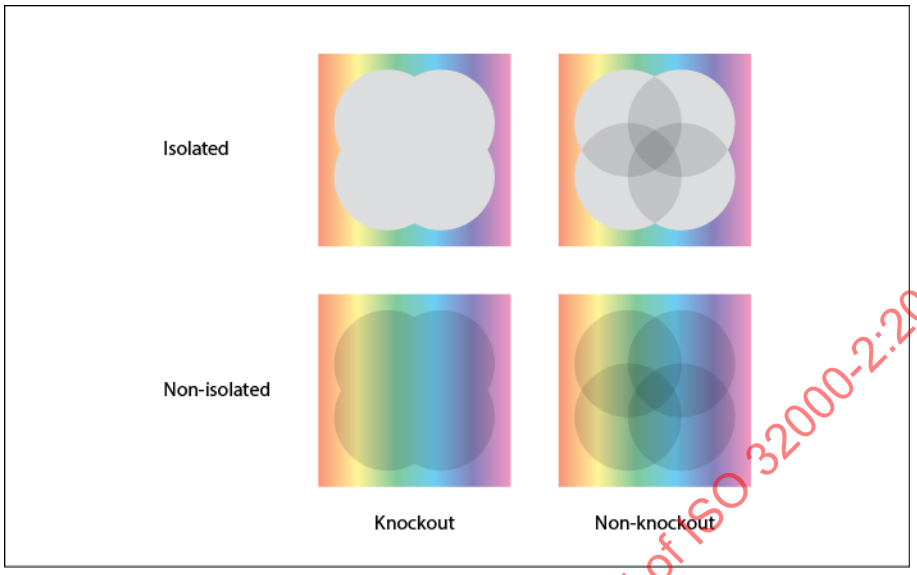


Figure 74 — Isolated and knockout groups

"Figure 74 — Isolated and knockout groups" (see 11.4.5, "Isolated groups" and 11.4.6, "Knockout groups") 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 11.4.6, "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.

For an isolated group, the group compositing formulas shall be altered by adding one statement to the initialization:

$a_0 = 0.0$ 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.

When compositing the group's results with the group backdrop, any required colour conversion from the blending space to the colour space in effect for the back drop shall use the rendering intent and black point compensation values specified in the current graphics state (see 8.4, "Graphics state") at the time the group was invoked.

11.4.6 Knockout groups

In a knockout group, each individual element shall be composited with the group's initial backdrop

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 74 — Isolated and knockout 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 unnormalised 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.

$$\alpha_t = \text{Union}(\alpha_{g_b}, q_{s_i})$$

$$C_t = (1 - q_{s_i}) \times \alpha_b \times C_b + q_{s_i} \times \left((1 - \alpha_b) \times C_{s_i} + \alpha_b \times \text{B}_i(C_b, C_{s_i})\right)$$

b) Compute a weighted average of this result with the object's immediate backdrop, using the source shape as the weighting factor. Then normalise the result colour by the result alpha:

$$\alpha_{g_i} = (1 - f_{s_i}) \times \alpha_{g_{i-1}} + f_{s_i} \times \alpha_t$$

$$\alpha_i = \text{Union}(\alpha_0, \alpha_{g_i})$$

$$\text{C}_\text{i} = \frac{(1 - f_{\text{s}_\text{i}}) \times \alpha_{\text{i}-1} \times \text{C}_{\text{i}-1} + f_{\text{s}_\text{i}} \times \text{C}_\text{t}}{\alpha_\text{i}}$$

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}.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 (in a colour space chosen by the PDF processor), although varying according to the actual properties of the medium. However, some interactive PDF processors 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, for example, when used as a reference XObject (see 8.10.4, "Reference XObjects"). In this situation the PDF "page" shall not be composited with the media colour; instead it shall be treated as a transparency group using the page Group attributes dictionary and is composited with its backdrop in the usual way according to the page Group attributes dictionary settings.

The remainder of this subclause 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:

$$\langle C_g, f_g, \alpha_g \rangle = \text{Composite}(U, 0, P)$$
$$C = (1 - \alpha_g) \times W + \alpha_g \times C_g$$

where the variables have the meanings shown in "Table 141 — Variables used in the page group compositing formulas". 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 141 — Variables used in the page group compositing formulas

Variable	Meaning
P	The page group, consisting of all elements E_1,..., 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)

The page group's initial blending colour space is inherited from the native colour space of the actual, assumed or simulated output device.

NOTE 1 A PDF processor can choose to simulate an output device other than the actual one being used, for example when doing soft proofing.

NOTE 2 A PDF processor rendering for a device with a native colour space that cannot be represented as a PDF colour space can choose to require that the output be treated as a simulation of a print characterization that can be represented as a PDF colour space.

That initial colour space shall serve as the default blending colour space for each page, unless the page explicitly specifies an alternative default by means of its page dictionary containing a **Group** key that contains a **CS** key whose value represents a different colour space from the initial blending colour space.

If a PDF processor chooses to simulate an output device other than the actual one being used, for example when doing "Soft Proofing", then the native colour space of that simulated output device should be used as the default blending colour space for each page of the document.

A PDF writer should provide a CIE-based colour space, either by the page group's **CS** entry or some other means, to ensure more predictable results of the compositing computations within the page group.

NOTE 3 PDF/X-4 (ISO 15930-7) establishes the destination profile in the PDF/X OutputIntent as the implied default page blending colour space, which allows the page group's **CS** entry to be a device colour but still provide a CIE-based colour space.

All page-level compositing shall be done in the default blending colour space of the page, and the entire result shall then, if the colour spaces are not equivalent, be converted to the native colour space of the output device before being composited with the context-dependent backdrop.

If the page group needs to be converted to the colour space of the output device, the colour conversion shall use a rendering intent of *RelativeColorimetric* unless the processor has an implementation-dependent way of specifying it otherwise. Additionally, the use of black point compensation in this colour conversion process is implementation-dependent.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

11.4.8 Summary of group compositing computations

This subclause is a restatement of the group compositing formulas that also takes isolated groups and knockout groups into account. See "Table 139 — Arguments and results of the group compositing function" and "Table 140 — Variables used in the group compositing formulas" 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 $Ei \in G$ $(i = 1, \ldots, n)$:

$$b = \begin{cases} 0 & \text{if the group is a knockout} \\ i - 1 & \text{otherwise} \end{cases}$$

$$\langle C_{s_i}, f_{j_i}, \alpha_{j_i} \rangle = \begin{cases} \text{Composite}(C_b, \alpha_b, E_i) & \text{if } E_i \text{ is a group} \\ \text{intrinsic color, shape, and (shape×opacity) of } E_i & \text{otherwise} \end{cases}$$

$$f_{s_i} = f_{j_i} \times f_{m_i} \times f_{k_i}$$
$$\alpha_{s_i} = \alpha_{j_i} \times (f_{m_i} \times q_{m_i}) \times (f_{k_i} \times q_{k_i})$$

$$f_{g_i} = \text{Union}(f_{g_{i-1}}, f_{s_i})$$
$$\alpha_{g_i} = (1 - f_{s_i}) \times \alpha_{g_{i-1}} + (f_{s_i} - \alpha_{s_i}) \times \alpha_{g_b} + \alpha_{s_i}$$
$$\alpha_i = \text{Union}(\alpha_0, \alpha_{g_i})$$

$$C_t = (f_{s_i} - \alpha_{s_i}) \times \alpha_b \times C_b + \alpha_{s_i} \times \left((1 - \alpha_b) \times C_{s_i} + \alpha_b \times B_i(C_b, C_{s_i})\right)$$
$$C_i = \frac{(1 - f_{s_i}) \times \alpha_{i-1} \times C_{i-1} + C_t}{\alpha_i}$$

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 can use the simplified derived formulas.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

11.5 Soft masks

11.5.1 General

As stated in earlier subclauses, 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 8.5.4, "Clipping path operators"). 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 can 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 can then be used to derive the mask in either of two ways, as described in the following subclauses.

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 customised.

NOTE 1 Since the group's colour is not used in this case, there is no need to compute it.

NOTE 2 The shape and alpha of an empty group is 0.0.

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

$$\langle C_g, f_g, \alpha_g \rangle = \text{Composite}(C_0, 1, G)$$
$$C = (1 - \alpha_g) \times C_0 + \alpha_g \times C_g$$

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 colorimetrically correct luminosity.

EXAMPLE 1 In the case of a PDF **CalRGB** space, the formula is

$$Y = Y_A \times A^{G_R} + Y_B \times B^{G_G} + Y_C \times C^{G_B}$$

using components of the **Gamma** and **Matrix** entries of the colour space dictionary (see "Table 63 — Entries in a CalRGB colour space dictionary" in 8.6.5, "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.

EXAMPLE 2 This method makes no pretence of colorimetric 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:

$$Y = 0.30 \times R + 0.59 \times G + 0.11 \times B$$

$$Y = 1 - \min(1, 0.3 \times C + 0.59 \times M + 0.11 \times Y + K)$$

Following this conversion, the result shall be passed through a separately specified transfer function, allowing the masking effect to be customised.

NOTE 2 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

Subclauses 11.1 through 11.5 inclusive have presented the transparent imaging model at an abstract level, with little mention of its representation in PDF. This subclause describes the facilities available for specifying transparency in PDF.

11.6.2 Specifying source and backdrop colours

Single graphics objects, as defined in 8.2, "Graphics objects", shall be treated as elementary objects for transparency compositing purposes (subject to special treatment for text objects, as described in 9.3.8, "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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 blend mode $B(C_b, C_s)$ shall be determined by the current blend mode parameter in the graphics state (see 8.4, "Graphics state"), which is specified by the **BM** entry in a graphics state parameter dictionary (8.4.5, "Graphics state parameter dictionaries"). Its value shall be either a name object, designating one of the standard blend modes listed in "Table 134 — Standard separable blend modes" and "Table 135 — Standard non-separable blend modes" in 11.3.5, "Blend mode" or its value shall be an array of such names. However, a value which is an array of names is deprecated in PDF 2.0, and should not be used. If encountered, a PDF processor shall use the first blend mode in the array that it recognizes (or **Normal** if it recognizes none of them).

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 subclauses 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 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 (8.5.3, "Path-painting operators", and 9.4.3, "Text-showing operators"), the shape shall always be 1.0 inside and 0.0 outside the path.
- For images (8.9, "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 (8.9.6.3, "Explicit masking" and 8.9.6.4, "Colour key masking").
- For image masks (8.9.6.2, "Stencil masking"), the shape shall be 1.0 for painted areas and 0.0 for masked areas.
- For objects painted with a tiling pattern (8.7.3, "Tiling patterns") or a shading pattern (8.7.4, "Shading patterns"), the shape shall be further constrained by the objects that define the pattern (see 11.6.7, "Patterns and transparency").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- For objects painted with the **sh** operator (8.7.4.2, "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 8.7.4.3, "Shading dictionaries").

All elementary objects shall have an intrinsic opacity q_i 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 "11.6.4.3, "Mask shape and opacity" and 11.6.4.4, "Constant shape and opacity".

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 8.4, "Graphics state").

NOTE 1 This is a boolean flag, set with the **AIS** ("alpha is shape") entry in a graphics state parameter dictionary (8.4.5, "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 11.6.5.1, "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. It shall also mean that any existing mask shall be removed from the current graphics state. In this case, the mask shape or opacity shall be implicitly 1.0 everywhere.

NOTE 2 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 11.6.4.4, "Constant shape and opacity").

- An image XObject may contain its own *soft-mask image* in the form of a subsidiary image XObject referenced as the value of the **SMask** entry of the parent image dictionary (see 8.9.5, "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, for this image object only, the current soft mask in the graphics state.
- An image XObject that has an announced **JPXDecode** filter may specify an **SMaskInData** entry, indicating that the soft mask is embedded in the data stream as an opacity channel (see 7.4.9, "JPXDecode filter"). If **SMaskInData** is present with a value other than 0, the embedded soft-mask shall override any explicit or colour key mask specified by the image dictionary's **Mask** entry, and it shall override, for this image object only, the current soft mask in the graphics state.

11.6.4.4 Constant shape and opacity

The current alpha constant parameter in the graphics state (see 8.4, "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 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The stroking and nonstroking alpha constants shall be set, respectively, by the **CA** and **ca** entries in a graphics state parameter dictionary (see 8.4.5, "Graphics state parameter dictionaries"). As described previously for the soft mask, the AIS ("alpha is shape") entry in a graphics state parameter dictionary shall determine whether the alpha constants are interpreted as shape values (*true*) or opacity values (*false*).

11.6.5 Specifying soft masks

11.6.5.1 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 8.4, "Graphics state"). "Table 142 — Entries in a soft-mask dictionary" 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 11.6.6, "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 may be omitted. The value of the **TR** key in the **SoftMask** dictionary, which is a transfer function, 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 value of the **TR** key in the **SoftMask** dictionary, which is a transfer function, 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 8.10.2, "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 8.6.6.4, "Separation colour spaces" and 8.6.6.5, "DeviceN colour spaces").

NOTE The special colourant names All and None do not specify spot colour components and are therefore not subject to this requirement.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 142 — 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 that shall 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 11.5.2, "Deriving a soft mask from group alpha"). *Luminosity* The group's computed colour shall be converted to a single-component luminosity value (see 11.5.3, "Deriving a soft mask from group luminosity").
G	stream	(*Required*) A transparency group XObject (see 11.6.6, "Transparency group XObjects") that shall 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 that shall 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 11.6.6, "Transparency group XObjects"). Default value: the colour space's initial value, representing black.
TR	function or name	(*Optional*) A function object (see 7.10, "Functions") specifying the transfer function that shall 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

11.6.5.2 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 8.9.5, "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 87 — Additional entries specific to an image dictionary" in 8.9.5, "Image dictionaries"), subject to the restrictions listed in "Table 143 — Restrictions on the entries in a soft-mask image dictionary". 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 pre-blended 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*.)

© ISO 2020 – All rights reserved

If the image data are pre-blended, the matte colour shall be specified by a **Matte** entry in the soft-mask image dictionary (see "Table 143 — Restrictions on the entries in a soft-mask image dictionary"). The pre-blending 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 143 — Restrictions on the entries in a soft-mask image dictionary

Key	Restriction
Type	If present, shall be XObject.
Subtype	Required, shall be Image.
Width	If a **Matte** entry (see "Table 144 — Additional entry in a soft-mask image dictionary") 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	Optional; default value: [0 1].
Interpolate	Optional.
Alternates	Ignored.
Name	Ignored.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Restriction
StructParent	Ignored.
ID	Ignored.
OPI	Ignored.

Table 144 — 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 pre-blended. The array shall consist of *n* numbers, where *n* is the number of components in the colour space specified by the **ColorSpace** entry (or the base entry of the colour space, if the colour space is **Indexed**) 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 pre-blended.

When pre-blended image data are 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 PDF processor 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.

NOTE When deriving c from *c*' and α is 0.0, the inverted formula will result in a divide by zero. In this special case an arbitrary value for *c* can be chosen within the range of colour component values for the image colour space. Because α is 0.0, the arbitrary value of *c* does not affect output as described in 11.3.2 "Basic notation for compositing computations".

The pre-blending 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 pre-blending shall precede the colour conversion. If the image colour space is an **Indexed** space (see 8.6.6.3, "Indexed colour spaces"), the colour values in the colour table (not the index values themselves) shall be pre-blended.

11.6.6 Transparency group XObjects

A transparency group is represented in PDF as a special type of group XObject (see 8.10.3, "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 8.10.2, "Form dictionaries"). The value of this entry is a group attributes dictionary defining the properties of the group. The format and meaning of the dictionary's contents is 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 145 — Additional entries specific to a transparency group attributes dictionary".

A page object (see 7.7.3.3, "Page objects") may also have a **Group** entry, whose value is a group attributes dictionary specifying the attributes of the page group (see 11.4.7, "Page group"). Some of the

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 145 — 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.
CS	name or array	(*Sometimes required for* Luminosity *groups, otherwise optional*) 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 11.3.4, "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 8.6.5.6, "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 11.6.5.1, "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. 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 12.5.5, "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 11.4.5, "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.
K	boolean	(*Optional*) A flag specifying whether the transparency group is a knockout group (see 11.4.6, "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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 8.10, "Form XObjects"):

- Initial backdrop: If the transparency group is non-isolated, 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 immediate 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.
- Initial blend mode: Before execution of the transparency group XObject's content stream, the current blend mode in the graphics state shall be initialised 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.

- Compositing: 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.
- Group colour space:
    - For isolated groups, if a group colour space (**CS**) is specified in the group attributes dictionary, all painting operators shall convert source colours in a colour space (that are not equivalent to the group colour space) to the group colour space before compositing objects into the group. The resulting colour at each point shall be interpreted in the group colour space. Rules for handling **Separation** and **DeviceN** colour spaces are described in 11.7.3, "Spot colours and transparency".
    - For non-isolated groups, or if no group colour space is specified, the group colour space shall be inherited from the parent group or page. If not otherwise specified, the page group's colour space shall be defined as described in 11.4.7, "Page group". All painting operators shall convert source colours in a colour space that is not equivalent to the inherited colour space, to that inherited colour space before compositing objects onto the backdrop.
- Final compositing: 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 colour conversion needs to take place in order to composite the group into its parent, the rendering intent and black point compensation from the graphics state at the point of invocation of the **Do** operator shall be used for the conversion.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 2 Applications that perform caching of rendered form XObjects need to 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 contains a group attributes subdictionary whose group subtype (**S**) is *Transparency*. An ordinary form XObject — one having no **Group** entry — or having a **Group** entry with a subtype other than *Transparency* — shall not be subject to any grouping behaviour for transparency purposes.

11.6.7 Patterns and transparency

In the transparent imaging model, the graphics objects making up the pattern cell of a tiling pattern (see 8.7.3, "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 (8.7.4, "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 initialised as follows:

- As always for transparency groups, those parameters related to transparency (blend mode, soft mask, and alpha constant) shall be initialised to their standard default values.
- Any parameters that are not so specified shall be inherited from the graphics state that was in effect at the beginning of the content stream in which the shading pattern is set to be the current colour in the graphics state or in which the **sh** operator is used.
- 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 8.7.4, "Shading patterns"). Only those parameters that affect the **sh** operator, such as the current transformation matrix, black point compensation 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_i) 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 can 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 re-executed 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 optimised 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 2 In a raster-based implementation of tiling, it is advisable to treat all tiles 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

The following subclauses describe 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", an isolated transparency group shall either have an explicitly declared colour space or inherit that of its parent group. If the colour space of the transparency group is a device colour space, and some ancestor of the group has a CIE-based colour space with the same number of colourants, then the colour space of this group shall be the CIE-based space of the nearest such ancestor.

If an isolated transparency group or page has an **ICCBased** 'CMYK' colour space, **DeviceCMYK** shall be redefined within the transparency group to be the same as the blending colour space and references to the process colourants **Cyan**, **Magenta**, **Yellow** and **Black** are defined to be references to the corresponding colourants in the blending colour space, even where the actual or simulated output device is not CMYK.

Non-isolated groups shall inherit their colour space from the nearest ancestor isolated parent group (subject to special treatment for the page group, as described in 11.4.7, "Page group").

NOTE 1 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 is not always feasible (since some colour conversions can be performed only in one direction), and even if feasible, it would entail an excessive number of colour conversions.

If the colour space of a graphics object within the group is not equivalent to the group's blending colour space, then it shall be converted to the group's colour space, and all blending and compositing computations shall be done in that space (see 11.3.4, "Blending colour space"). The resulting colours shall then be interpreted in the group's colour space when the group is subsequently composited with its backdrop.

When converting colours, if the colour space of any graphics object is a device colour space, and the current group or an ancestor of the current group is defined with a CIE-based colour space with the same number of colourants, then, for compositing purposes only, the colour space of the graphics object shall be the CIE-based space of the nearest such ancestor. Nevertheless for graphics objects that

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

are specified in **DeviceCMYK**, the full semantics of **DeviceCMYK** shall still apply, including overprinting.

The choice of a group colour space has significant effects on the results that are produced because the result 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.

NOTE 2 A consequence of choosing a CIE-based group colour space is that only CIE-based spaces can be used to predictably specify the colours of objects within the group. This is because conversion from device to CIE-based colours is implementation-dependent. See further discussion subsequently.

NOTE 3 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 might not match the user's expectations.

NOTE 4 The CIE-based *sRGB* colour space (see 8.6.5, "CIE-Based colour spaces") is nonlinear and hence can be unsuitable for use as a group colour space.

NOTE 5 To minimise the accumulation of round off 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 for painting objects of device colour spaces with a different number of colourants than the group's colour space shall be subject to special treatment. These 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 8.6.5.6, "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 or mapped to a CIE-based colour space in an implementation-dependent fashion. The restrictions concerning device colourants do not apply if the group's colour space is implicitly converted to **DeviceCMYK**, as discussed in 8.6.5.7, "Implicit conversion of CIE-Based colour spaces".

NOTE 6 See Annex P, "An algorithm to determine the actual blending colour space of a transparency group" for a diagram that illustrates an algorithm to determine the actual blending colour space for a transparency group.

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 colourants. 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

specify them, process colours may 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 colourant to be applied to the composite page (see 8.6.6.4, "Separation colour spaces" and 8.6.6.5, "DeviceN colour spaces"). The colour component value, or tint, for a spot colour specifies the concentration of the corresponding spot colourant. 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 colourants when the corresponding spot colourant 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 11.7.4, "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 should 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 paints 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-

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 1 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 can 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.

NOTE 2 The special colourant **All** is defined as applying tint values to all available colourants at once (see 8.6.6.4, “Separation colour spaces”). When **All** is used inside a transparency group, "all available colourants" is no longer the same as "all colourants available on the output device." Painting **All** inside a transparency group can only apply tint values to the process colourants that are defined by the group blending colour space and to the spot colourants available on the output device (as the spot colours are maintained separate from the group blending colour space). Indeed, inside a transparency group, the process colourants of the output device are irrelevant and unknown, only the colourants of the group blending colour space (and the spot colours of the output device) are available.

Once a tint of Separation **All** has been applied to all colourants available in the transparency group, this result is then subject to the usual colour conversion rules when the group as a whole is composited on its own backdrop. The practical implication of this is that applying **All** inside a transparency group does not necessarily result in an equal tint percentage for all process colour components on that location on the page.

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 8.6.7, "Overprint control"). Painting an object causes some specific set of device colourants to be marked, as determined by the current colour space and current colour in the graphics state. The remaining colourants 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 colourants 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 colourants, 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 colourants or for very dark colours, such as black. It is also invoked during late-stage production operations such as trapping (see 14.11.6, "Trapping support"), when the actual set of device colourants has already been determined.

NOTE 2 Consequently, it is best to think of transparency as taking place in appearance space, but overprinting of device colourants in device space. This means that colourant overprint decisions are made at output time, based on the actual resultant colourants 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 11.7.4.2, “Blend modes and overprinting”.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 colourants in the transparency group's colour space as well as any available spot colourants. 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 may 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 the group colour space is different from the native colour space of the output device, its components are not the device's actual process colourants; the blending computations shall affect the process colourants 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 colourants 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 are not always 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 could be specifically desired for those components.

The PDF graphics state specifies only one current blend mode parameter, which shall always apply to process colourants and sometimes to spot colourants as well. Specifically, only separable, white-preserving blend modes shall be used for spot colours. If the specified blend mode is not separable or not 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 134 — Standard separable blend modes" 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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

For compatibility with the methods of overprint control used in the opaque imaging model, a PDF processor may consider implementing a special blend mode 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; rather, it may 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 blend mode described here. Because this mode can be invoked implicitly whenever appropriate, it is never necessary to specify the **Compatible** blend mode for use in compositing.

The value of the blend function $B(C_b, C_s)$ in the this 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 equivalent, 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 can be specified by the current colour space parameter in the graphics state (see 8.6.2, "Colour values"), implicitly by colour operators such as rg (8.6.8, "Colour operators"), or by the **ColorSpace** entry of an image XObject (8.9.5, "Image dictionaries"). In the case of an **Indexed** space, it refers to the base colour space (see 8.6.6.3, "Indexed colour spaces"); likewise for **Separation** and **DeviceN** spaces that revert to their alternate colour space, as described under 8.6.6.4, "Separation colour spaces" and 8.6.6.5, "DeviceN colour spaces".

If the current blend mode is any mode other than **Normal** when invoking this special overprinting blend mode, 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 this special 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 special blend mode while painting the object produces equivalent results. There are some additional cases in which the implicit transparency group can be optimised out.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

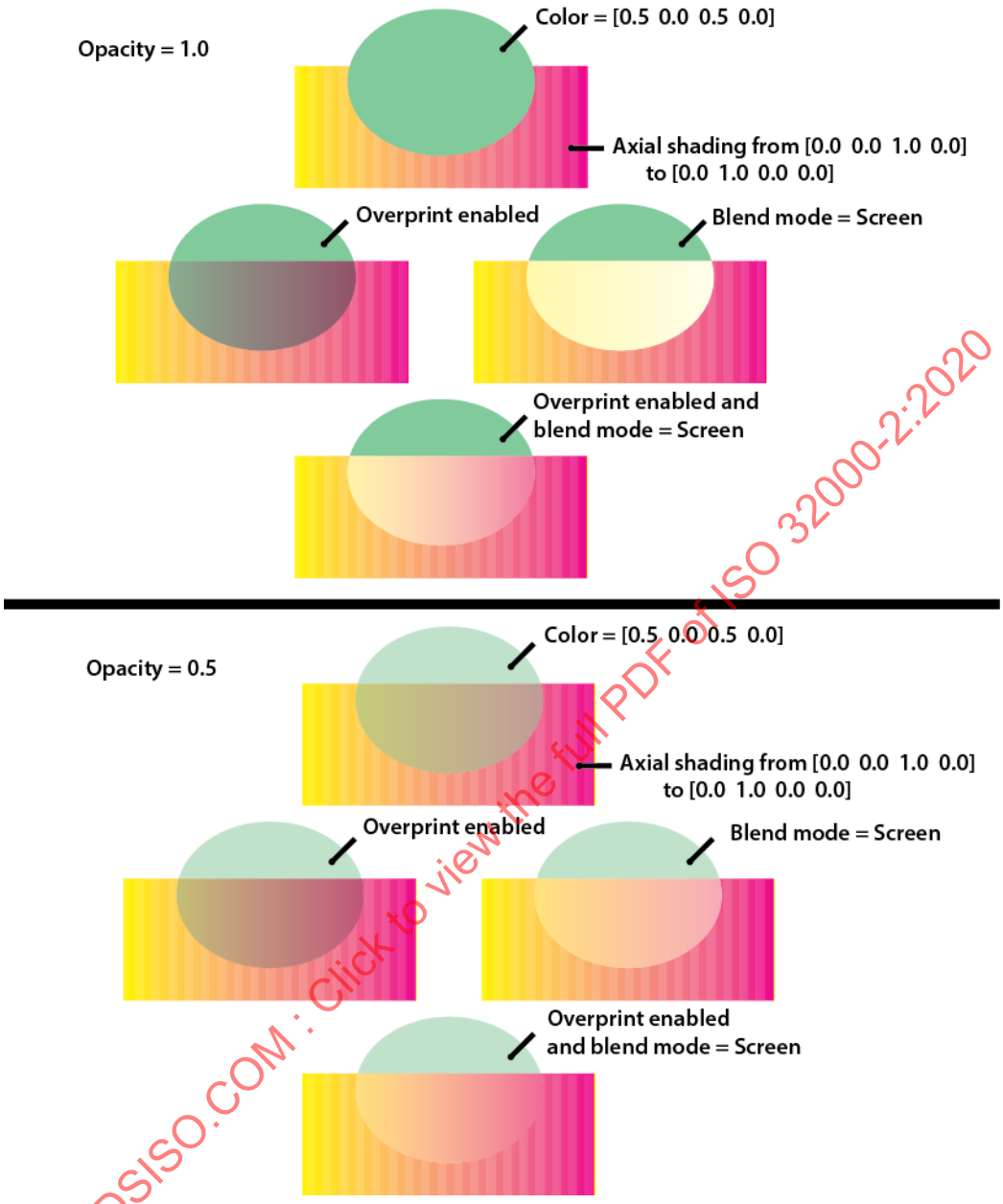


Figure 75 — Blending and overprinting

EXAMPLE 2 "Figure 75 — Blending and overprinting" 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 8.5.3, "Path-painting operators") and the painting of glyphs with text rendering mode 2 or 6 (9.3.6, "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:

© ISO 2020 – All rights reserved

- If overprinting is enabled (the overprint parameters for both stroking and non-stroking operations in the graphics state are *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 special overprinting blend mode described in 11.7.4.3, “Compatibility with opaque overprinting”. 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 9.3.8, "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 behaviour 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 behavior

“Table 146 — Overprinting behaviour in the transparent imaging model” shows overprinting behavior in the transparent imaging model.

Table 146 — Overprinting behaviour in the transparent imaging model

Source colour space	**Affected colour component of group 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 colourant	$C_s(= 0.0)$	C_b	C_b
Any process colour space (including other cases of **DeviceCMYK**)	Process colour component	C_s	C_s	C_s
	Spot colourant	$C_s(= 0.0)$	C_b	C_b

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Source colour space	Affected colour component of group space	Value of blend function $B(C_b, C_s)$ expressed as tint		
		OP false	OP true, OPM 0	OP true, OPM 1
Separation or **DeviceN**	Process colour component	$C_s(= 0.0)$	C_b	C_b
	Spot colourant named in source space	C_s	C_s	C_s
	Spot colourant 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 colourants such as inks rather than to additive ones such as phosphors on a display screen. The special overprinting blend mode is therefore described as if it took subtractive arguments and returned subtractive results. In reality, however, the special overprinting blend mode (like all blend modes) shall treat colour components as additive values; subtractive components shall be complemented before and after application of the special blend function.

NOTE 1 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 11.7.3, "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 146 — Overprinting behaviour in the transparent imaging model" 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 unavailable in this case; the special overprinting blend mode reverts to **Normal**, with no consideration of the overprint and overprint mode parameters. A transparency-aware PDF 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 10.6.5, "Halftone dictionaries"), transfer functions (10.5, "Transfer functions"), rendering intent (8.6.5.8, "Rendering intents"), black point compensation (8.6.5.9, "Use of black point compensation", and black-generation and undercolour-removal functions (10.4.2.3,

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

"Conversion between DeviceGray and 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 undercolour 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 may 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 may 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 is 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**.
- The current soft mask in the graphics state is **None**.
- If the object is an image XObject and there is not an **SMask** entry in its image dictionary.
- The foregoing four 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 (see "Table 52 — Device-dependent graphics state parameters").

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. An object is opaque

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

for a given component only if overprinting yields the source colour (not the backdrop colour) for that component. See 11.7.4, "Overprinting and transparency".

11.7.5.3 Rendering intent, black point compensation and colour conversions

The rendering intent, black-generation, undercolour-removal and black point compensation 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 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) is 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.

Black point compensation is needed during colour conversions in cases where the detail in dark regions or shadow section of an image can otherwise be lost. Black point compensation addresses this conversion problem by adjusting for differences between the darkest level of black achievable in one colour space and the darkest level of black achievable on another. The graphics state parameter **UseBlackPtComp** ("Table 57 — Entries in a graphics state parameter dictionary) enables or disables the use of black point compensation when doing colour conversions.

A similar approach works for the black-generation and undercolour-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 undercolour-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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12 Interactive features

12.1 General

This clause describes the PDF features that allow a user to interact with a document on the screen (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 (subclauses 12.3, "Document-level navigation" and 12.4, "Page-level navigation")
- *Annotations* for adding text notes, links, rich media 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, "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")
- A *geospatial coordinate system* is introduced in subclause 12.10, "Geospatial features" along with a number of PDF constructs to support geospatially registered content.
- Subclause 12.11, "Document requirements" describes how a document may specify requirements that an interactive PDF processor should satisfy in order for the *document to function as intended* by the author.

12.2 Viewer preferences

The **ViewerPreferences** entry in a document's catalog dictionary (see 7.7.2, "Document catalog dictionary") 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, PDF processors should behave in accordance with their own current user preference settings. "Table 147 — Entries in a viewer preferences dictionary" shows the contents of the viewer preferences dictionary.

Table 147 — Entries in a viewer preferences dictionary

Key	Type	Value
HideToolbar	boolean	*(Optional)* A flag specifying whether to hide the interactive PDF processor's tool bars when the document is active. Default value: *false*.
HideMenubar	boolean	*(Optional)* A flag specifying whether to hide the interactive PDF processor's menu bar when the document is active. Default value: *false*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. D efault 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 centre 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 **dc:title** element of the XMP metadata stream (see 14.3.2, "Metadata streams"). 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 dictionary") is FullScreen; it shall be ignored otherwise. Default value: *UseNone*.
Direction	name	*(Optional; PDF 1.3)* The predominant logical content 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; deprecated in PDF 2.0)* 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 31 — Entries in a page object". 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". The presence of this value in a PDF may cause a PDF to display differently from how it will be printed.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
ViewClip	name	*(Optional; PDF 1.4; deprecated in PDF 2.0)* 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 31 — Entries in a page object". 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". The presence of this value in a PDF may cause a PDF to display differently from how it will be printed.
PrintArea	name	*(Optional; PDF 1.4; deprecated in PDF 2.0)* 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 31 — Entries in a page object". 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". The presence of this value in a PDF may cause a PDF to display differently from how it will be printed.
PrintClip	name	*(Optional; PDF 1.4; deprecated in PDF 2.0)* 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 31 — Entries in a page object". 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". The presence of this value in a PDF may cause a PDF to display differently from how it will be printed.
PrintScaling	name	*(Optional; PDF 1.6)* The page scaling option that shall be selected when a print dialogue is displayed for this document. Valid values are *None*, which indicates no page scaling, and *AppDefault*, which indicates the interactive PDF processor's default print scaling. If this entry has an unrecognised value, *AppDefault* shall be used. Default value: *AppDefault*. If the print dialogue is suppressed and its parameters are provided from some other source, this entry nevertheless shall be honoured.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Duplex	name	(*Optional; PDF 1.7*) The paper handling option that shall be used when printing the PDF file from the print dialogue. 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: implementation dependent
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 dialogue presented by an interactive PDF processor. If **PickTrayByPDFSize** is *true*, the check box in the print dialogue 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: implementation dependent
PrintPageRange	array	(*Optional; PDF 1.7*) The page numbers used to initialise the print dialogue box when the PDF 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: implementation dependent NOTE Although **PrintPageRange** uses 1-based page numbering, other features of PDF use zero-based page numbering.
NumCopies	integer	(*Optional; PDF 1.7*) The number of copies that shall be printed when the print dialog is opened for this PDF file. Default value: implementation dependent, but typically *1*
Enforce	array	*(Optional; PDF 2.0)* An array of names of Viewer preference settings that shall be enforced by PDF processors and that shall not be overridden by subsequent selections in the application user interface. "Table 148 — Names defined for an Enforce array" specifies names that shall be valid to use in this array.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The **Enforce** array shall only include names that occur in "Table 148 — Names defined for an Enforce array". Future additions to this table shall be limited to keys in the viewer preferences dictionary with the following qualities:

- can be assigned values (default or specified) that cannot be used in a denial-of-service attack, and
- have default values that cannot be overridden using the application user interface.

© ISO 2020 – All rights reserved

Table 148 — Names defined for an Enforce array

Name	Description
PrintScaling	(*Optional; PDF 2.0*) This name may appear in the **Enforce** array only if the corresponding entry in the viewer preferences dictionary ("Table 147 — Entries in a viewer preferences dictionary") specifies a valid value other than *AppDefault*.

12.3 Document-level navigation

12.3.1 General

The features described in this subclause allow an interactive PDF processor 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 catalog dictionary (7.7.2, "Document catalog dictionary") 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 149 — Destination syntax" 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", or an embedded go-to action; see 12.6.4.4, "Embedded 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.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE The above paragraph was corrected to also include embedded go-to actions (2020).

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 149 — 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, centring 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, centring 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, centring 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 Structure destinations

A destination provides a view within a given document, however there is no direct connection between the location of the view on the page and the content displayed. Structure elements (see 14.7.2, "Structure hierarchy") allow specific sequences of content on a page to be identified. A *structure destination* provides the same view mechanism as a destination, but references a structure element instead of a page. A structure destination shall use the same syntax as a destination (see "Table 149 — Destination syntax"), except that the first entry in the array shall be an indirect reference to a structure element dictionary instead of to a page dictionary.

EXAMPLE A regular destination of the following syntax [*page /FitH 500*] could be represented as a structure destination with the following syntax [*elem /FitH 500*] where *elem* represents an indirect reference to a structure element dictionary.

The structure element shall be used to identify the page to which the content belongs and, using that page, the structure destination shall behave identically to a destination. To identify the page to which a structure destination refers, the following algorithm shall be used. The kids of the structure element shall be processed in linear array order. If the first kid is a marked-content reference or an object reference (see 14.6, "Marked content"), then the page to which that reference belongs shall be used as the page. If the first kid is a structure element, then processing shall continue down to that element using the same algorithm recursively. If no content or object reference is found under the first entry, processing should proceed to next entry, repeating the process. This shall continue until all entries have been processed or until the first page is identified. In the case where no page content is identified, then the page reference shall be assumed to be the first page in the document.

No structure element dictionary can be specified for a structure destination associated with a remote go-to action (see 12.6.4.3, "Remote Go-To actions") or embedded go-to actions (see 12.6.4.4, "Embedded Go-To actions") because the destination structure element is in a different PDF document. In this case, the indirect reference to the structure element dictionary shall be replaced by a byte string representing a structure element **ID** (see "Table 355 — Entries in a structure element dictionary").

NOTE 1 The above paragraph was corrected to also include embedded go-to actions (2020).

NOTE 2 There is no requirement that a given structure element will have an **ID** associated with it, nor that it will remain consistent across edits to a PDF document. For remote go-to actions which rely on a target PDF having an **ID**, it is important that such an **ID** exist and that it remain consistent across versions of the target document.

12.3.2.4 Named destinations

Instead of being defined directly with the explicit syntax shown in "Table 149 — Destination syntax", 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 1 A link to the beginning of Chapter 6 in another document can 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 (12.6.4.7, "Thread actions"), the destination is in the PDF file specified by the action's **F** entry, if any; if there is no **F** entry, the destination is in the current PDF file.

In PDF 1.1, the correspondence between name objects and destinations shall be defined by the **Dests** entry in the document catalog dictionary (see 7.7.2, "Document catalog dictionary"). 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 149 — Destination syntax", or a dictionary with a **D** entry whose value is such an array and may optionally contain an **SD** entry as defined in "Table 201 — Action types".

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 can 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.

When trying to locate a named destination in a names tree, either in the same or in remote PDF files, the algorithms specified in J.3.3, "String objects" or J.3.4, "Name objects" shall be used (depending on the type of object being compared).

NOTE 3 The above paragraph was added in this document (2020).

NOTE 4 The use of strings as destination names is a PDF 1.2 feature. If compatibility with versions of PDF prior to PDF 1.2 is required, only name objects can be used to refer to named destinations. A document that supports PDF 1.2 or later can contain both types. However, if backward compatibility to PDF 1.2 is not a consideration, it is recommended that applications 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 interactive PDF processor 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 interactive PDF processor 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 catalog dictionary (see 7.7.2, "Document catalog dictionary"). "Table 150 — Entries in

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

the outline dictionary" shows the contents of this dictionary. Each individual outline item within the hierarchy shall be defined by an *outline item dictionary* ("Table 151 — Entries in an outline item dictionary"). 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 150 — 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 151 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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.
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"). NOTE This value is not intended for navigation. Structure Destinations (**Dest** entry) are the method to provide structure-based navigation.
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 152 — Outline item flags"). 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 152 — Outline item flags" shows the meanings of the flags; all other bits of the integer shall be 0.

Table 152 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. An interactive PDF processor 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 can 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 87 — Additional entries specific to an image dictionary" 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** colour 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.3.5 Collections

12.3.5.1 General

Beginning with PDF 1.7, PDF documents may specify how an interactive PDF processor'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 organisational characteristics of portable collections. If this dictionary is present in a PDF document, the interactive PDF processor shall present the document as a portable collection. The **EmbeddedFiles** name tree specifies file attachments (see "Table 32 — Entries in the name dictionary").

When an interactive PDF processor first opens a PDF document containing a collection, it shall display the contents according to the **View** key of the collection dictionary. The initial document may be the container PDF or one of the embedded documents as specified by the **D** key in the collection dictionary.

NOTE 2 The page content in the initial document needs to 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. For a PDF document that is an unencrypted wrapper for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document"), the **EmbeddedFiles** name tree shall contain exactly one entry, for the encrypted payload document.

Beginning with PDF 2.0, a portable collection may include an interactive layout, or presentation, of the collection contents. The collection layout or presentation is called a *navigator*. For more information about navigators, see 12.3.6, "Navigators".

When a navigator is used the collection dictionary shall contain some entries that support navigators. The **Navigator** entry shall be an indirect reference to a navigator dictionary that describes the interactive layout. The value of the **Colors** entry shall be a collection colors dictionary that specifies a suggested set of colours for use by a collection layout. The **Folders** entry shall be an indirect reference to the root folder of the collection's folder structure.

"Table 153 — Entries in a collection dictionary" describes the entries in a collection dictionary.

Table 153 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Schema	dictionary	(*Optional*) A collection schema dictionary (see "Table 154 — Entries in a collection schema dictionary"). If absent, the interactive PDF processor 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 is not a valid byte string, the initial document shall be the one that contains the collection dictionary. If the **D** entry is a valid byte string that does not match any file in the **EmbeddedFiles** name tree, the interactive PDF processor shall select the first item from the list of files to display in its user interface; if no files exist in the name tree, the interactive PDF processor shall display an empty preview window. *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **D** entry is required, and shall identify the encrypted payload entry in the **EmbeddedFiles** name tree.
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 interactive PDF processor shall provide means for the user to view the collection by some explicit action. The PDF processor should display the document specified by the **D** entry. NOTE How the PDF processor chooses to display the collection is implementation specific. *C* *(PDF 2.0, valid only when **Navigator** is present)* The collection view shall be presented by the navigator specified by the **Navigator** entry. Default value: *D* *(PDF 2.0)* For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **View** entry is required, and shall have a value of *H*.
Navigator	dictionary	*(Required if the value of **View** is C; PDF 2.0)* An indirect reference to the navigator dictionary that describes the navigator that provides the collection view. See "Table 160 — Entries in a navigator dictionary".
Colors	dictionary	*(Optional; PDF 2.0)* A collection colors dictionary specifying a suggested set of colours for use by a collection layout. See "Table 157 — Entries in a collection colors dictionary". NOTE It is recommended that a layout use the colours provided.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 156 — Entries in a collection sort dictionary").
Folders	dictionary	*(Required if the collection has folders; PDF 2.0)* An indirect reference to a folder dictionary that is the single common ancestor of all other folders in a portable collection. See "Table 159 — Entries in a folder dictionary".
Split	dictionary	*(Optional; PDF 2.0)* A collection split dictionary that specifies the orientation of the splitter bar in the user interface provided by the interactive PDF processor. See "Table 158 — Entries in a collection split dictionary". If **Split** is not present, the preferred orientation is determined by the value of the **View** key. A value of *D* (or no value) shall indicate a horizontal orientation, while a value of *T* shall indicate a vertical orientation. No splitter shall be used if the **View** key has a value of *H* or *C*.

A collection schema dictionary consists of a variable number of individual collection field dictionaries. Each collection field dictionary has a key chosen by the interactive PDF writer, which shall be used to associate a field with data in a file specification. "Table 154 — Entries in a collection schema dictionary" describes the entries in a collection schema dictionary.

Table 154 — 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 PDF 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 43 — Entries in a file specification dictionary").

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 155 — Entries in a collection field dictionary" describes the entries in a collection field dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 155 — 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 (see 7.9.4, "Dates"). *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 43 — Entries in a file specification dictionary"). *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 43 — Entries in a file specification dictionary"). *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 45 — Entries in an embedded file parameter dictionary"). *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 45 — Entries in an embedded file parameter dictionary"). *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 45 — Entries in an embedded file parameter dictionary"). *CompressedSize* *(PDF 2.0)* The field data shall be the length of the embedded file stream, as identified by the **Length** entry in the embedded file stream dictionary (see 7.11.4, "Embedded file streams"), and the two values shall be identical.
N	text string	(*Required*) The textual field name that shall be presented to the user by the interactive PDF processor.
O	integer	(*Optional*) The relative order of the field name in the user interface. Fields shall be sorted by the interactive PDF processor 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 interactive PDF processor 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:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- Text strings shall be ordered lexically from smaller to larger, if ascending order is specified.

NOTE 3 Lexical ordering is an implementation dependency for interactive PDF processors.

- 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 156 — Entries in a collection sort dictionary" describes the entries in a collection sort dictionary.

Table 156 — 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 interactive PDF processor 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 interactive PDF processor 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*.

Table 157 — Entries in a collection colors dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes; if present, shall be *CollectionColors* for a collection colors dictionary.
Background	array	*(Optional; PDF 2.0)* An array of three numbers in the range 0.0 to 1.0, representing a **DeviceRGB** colour used for the background of the view.
CardBackground	array	*(Optional; PDF 2.0)* An array of three numbers in the range 0.0 to 1.0, representing a **DeviceRGB** colour used for the background of the card.
CardBorder	array	*(Optional; PDF 2.0)* An array of three numbers in the range 0.0 to 1.0, representing a **DeviceRGB** colour used for the border of the card.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
PrimaryText	array	*(Optional; PDF 2.0)* An array of three numbers in the range 0.0 to 1.0, representing a **DeviceRGB** colour used for the primary text in a navigator.
SecondaryText	array	*(Optional; PDF 2.0)* An array of three numbers in the range 0.0 to 1.0, representing a **DeviceRGB** colour used for other text in a navigator.

When displaying a collection, an interactive PDF processor presents an initial view in which the available display area may be divided by a splitter bar into two areas; one area containing a display of the navigation controls of the collection as defined by the **View** and related entries of the collection dictionary, and one area containing a preview of the initial or currently selected document of the collection. The visibility, orientation or position of the splitter bar may be interactively adjusted by user action subsequent to its initial view as defined by the collection split dictionary, if provided.

Table 158 — Entries in a collection split dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes; if present, shall be *CollectionSplit* for a collection split dictionary.
Direction	name	*(Optional; PDF 2.0)* The orientation of the splitter bar. The following values are valid: *H* indicates that the window is split horizontally *V* indicates that the window is split vertically. *N* indicates that the window is not split. The entire window region shall be dedicated to the file navigation view.
Position	number	*(Optional; PDF 2.0)* The initial position of the splitter bar, specified as a percentage of the available window area. Values shall range from 0 to 100. The entry shall be ignored if **Direction** is set to *N*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.3.5.2 Collection hierarchical folders

Beginning with PDF 2.0, a portable collection can contain a Folders object for the purpose of organising files into a hierarchical structure. The structure is represented by a tree with a single root folder acting as the common ancestor for all other folders and files in the collection. The single root folder is referenced in the **Folders** entry of "Table 153 — Entries in a collection dictionary".

Table 159 — Entries in a folder dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes; if present, shall be *Folder* for a folder dictionary.

© ISO 2020 – All rights reserved

Key	Type	Value
ID	integer	*(Required; PDF 2.0)* A non-negative integer value representing the unique folder identification number. Two folders, in the same PDF document, shall not share the same **ID** value. The folder **ID** value appears as part of the name tree key of any file associated with this folder. A detailed description of the association between folder and files can be found after this table.
Name	text string	*(Required; PDF 2.0)* A file name representing the name of the folder. Two sibling folders shall not share the same name following case normalization. NOTE Descriptions of file name and case normalization follow this table.
Parent	dictionary	*(Required for child folders; PDF 2.0)* An indirect reference to the parent folder of this folder. This entry shall be absent for a root folder.
Child	dictionary	*(Required if the folder has any descendents; PDF 2.0)* An indirect reference to the first child folder of this folder.
Next	dictionary	*(Required for all but the last item at each level; PDF 2.0)* An indirect reference to the next sibling folder at this level. Siblings should be ordered according to the collection **Sort** key ("Table 153 — Entries in a collection dictionary") or by folder name if no collection **Sort** key is present.
CI (uppercase ci)	dictionary	*(Optional; PDF 2.0)* The collection item dictionary. 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 46 — Entries in a collection item dictionary" describes the entries in a collection item dictionary.
Desc	text string	*(Optional; PDF 2.0)* A text description associated with this folder.
CreationDate	date	*(Optional; PDF 2.0)* The date the folder was first created.
ModDate	date	*(Optional; PDF 2.0)* The date of the most recent change to immediate child files or folders of this folder.
Thumb	stream	*(Optional; PDF 2.0)* A stream object defining the thumbnail image for the folder See 12.3.4, "Thumbnail images".
Free	array	*(Optional; only used by root folder; PDF 2.0)* An array containing **ID** values that are not currently in use by the folder structure. The array shall contains zero or more pairs of numbers, a low value followed by a high value. Each pair represents an endpoint-inclusive range of values that are available for use when a new folder is added. Each low value shall be less than or equal to its corresponding high value.

New values for the **ID** key shall be obtained by an interactive PDF processor by accessing the **Free** entry in the root folder. If an **ID** value is used from the **Free** entry array, the array shall be updated.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

As previously described, the **Name** entry is a file name for a folder. A folder, as well as its associated files, have naming restrictions. Strings that conform to these restrictions are known as file names. A valid file name conforms to the following requirements:

- The string shall be a PDF text string.
- The string shall not contain any embedded NULL (U+0000) characters.
- The number of characters in the string shall be between 1 and 255 inclusive.
- The string shall not contain any of the eight special characters: SOLIDUS (U+002F) (/), REVERSE SOLIDUS (U+005C) (\), COLON (U+003A) (:), ASTERISK (U+002A) (*), QUOTATION MARK (U+0022) ("), LESS-THAN SIGN (U+003C) (<), GREATER-THAN SIGN (U+003E) (>) and VERTICAL LINE (U+007C) (|).
- The last character shall not be a FULL STOP (U+002E) (.).

An interactive PDF processor may choose to support invalid names or not. If not, an appropriate error message shall be provided.

In addition to the restriction on naming folders, as just described, it is further required that two file names in the same folder do not map to the same string following case normalization. Two file names that differ only in case are disallowed within the same folder. See "Unicode Standard Annex #21, Case Mappings" for information on case normalization.

The **CI** entry, a collection item dictionary, allows user-defined metadata to be associated with a folder, just as it does for embedded files in a collection.

Folders are indirect objects, and relationships between folders in the tree are specified using **Parent**, **Child**, and **Next** keys.

When folders are used, all files in the **EmbeddedFiles** name tree (see "Table 32 — Entries in the name dictionary") shall be treated as members of the folder structure by an interactive PDF processor. The association between files and folders is accomplished using a special naming convention on the key strings of the name tree. See 7.9.6, "Name trees", for a discussion of the key strings. If no folder structure is specified, interactive PDF processors should show all files in the collection in a flat list.

As previously mentioned, files in the **EmbeddedFiles** name tree are associated with folders by a special naming convention applied to the name tree key strings. Strings that conform to the following rules serve to associate the corresponding file with a folder:

- The name tree keys are PDF text strings.
- The first character, excluding any byte order marker, is LESS-THAN SIGN (U+003C) (<).
- The following characters shall be one or more digits (0 to 9) followed by the closing GREATER-THAN SIGN (U+003E) (>)
- The remainder of the string is a file name.

The section of the string enclosed by LESS-THAN SIGN GREATER-THAN SIGN(<>) is interpreted as a numeric value that specifies the **ID** value of the folder with which the file is associated. The value shall correspond to a folder ID. The section of the string following the folder **ID** tag shall represent the file name of the embedded file.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Files in the **EmbeddedFiles** name tree that do not conform to these rules shall be treated as associated with the root folder.

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 organisational data associated with each email is described in a collection schema dictionary. Most actual organisational 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 (Tom Jones)
   /to (Marry Jones)
   /subject <<
      /Type /CollectionSubitem
      /P (Re:)
      /D (Let's have lunch on Friday!)
            >>
   /date (D:20050621094703-07'00)
    >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.3.6 Navigators

A portable collection can include an interactive layout, or *presentation*, for the collection contents (PDF 2.0) called a *navigator*. When a navigator is specified, a PDF processor should display that interactive layout and allow it to drive the presentation of the collection.

Navigators are specified by identifying a named layout, used to identify a presentation for the content within a collection. These interactive layouts provide more choice in presenting the collection contents than the simple views specified by the **View** key in the collection dictionary (see "Table 153 — Entries in a collection dictionary"). When a navigator dictionary is present, a PDF processor should use the value of the **Layout** entry to present the collection to the user.

Named layouts provide a number of options for presenting the contents of the collection to a user. These options are provided to allow the choice of a navigator that is best suited to presenting the collection's contents, including the folder structures and file attachments. The **Layout** key may consist of a single name value, to specify the named layout, or an array of names. When an array of names is provided, a PDF processor should select the first named layout it recognises. This mechanism is inherently extensible and allows inclusion of custom named layouts, but at least one of the values of

© ISO 2020 – All rights reserved

the **Layout** entry shall be one of the values listed in the entry (see "Table 160 — Entries in a navigator dictionary").

Table 160 — Entries in a navigator dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes; if present, shall be *Navigator* for a navigator dictionary.
Layout	name or array of names	(*Required; PDF 2.0*) One or more names specifying the named layout of the navigator that should be used. When multiple names are provided, an interactive PDF processor should present the first one it is capable of displaying in the order present in the array. One of the following names shall always be present, either singly or as the final entry in the array. *D* Corresponding to the value of *D* in the **View** key in "Table 153 — Entries in a collection dictionary". *T* Corresponding to the value of *T* in the **View** key in "Table 153 — Entries in a collection dictionary". *H* Corresponding to the value of *H* in the **View** key in "Table 153 — Entries in a collection dictionary". *FilmStrip* A layout which displays a strip of thumbnails, providing an index to the file attachments within the collection. The selected attachment should be previewed alongside the index. *FreeForm* A layout which places thumbnails of the file attachments within the collection randomly in the view. *Linear* A layout which provides a large size preview of one file attachment in the collection and displays alongside the preview the metadata for the file attachment, including the name, description and other collection schema entries. *Tree* A layout presenting the contents of the collection in a tree view, showing the folder structure and the files as leaf nodes of the tree, akin to a traditional file system folder view.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The *D, T* and *H* values for the **Layout** entry match those present in the **View** entry of a collection dictionary (see "Table 153 — Entries in a collection dictionary"). An interactive PDF processor should present the same display mode when encountering these values as it would if processing the **View** entry.

The *FilmStrip* layout describes a presentation of the collection contents in the form of a single strip of thumbnails. These thumbnails provide an index into the files and folders present within the collection. When a user selects a file from the index, an interactive PDF processor should either display that file or provide a preview of the file.

NOTE 1 A common implementation for this is to have the strip of thumbnails across the bottom of the view and a large preview of the selected file or folder shown above it.

The *FreeForm* layout provides a simple layout, in which thumbnails for each item in the collection contents are displayed at a random location on the view. When a thumbnail is selected, an interactive PDF processor should display the attachment.

© ISO 2020 – All rights reserved

The *Linear* layout provides a view of an attachment, which is usually larger than just a thumbnail, with the metadata for the file displayed alongside it. An interactive PDF should display the first page of the file and should use the file schema and file specification dictionary to provide information about the attachment.

The *Tree* layout is intended to provide a classic folder view of the contents of a collection, akin to that found on many operating systems. An interactive PDF should present the folder structure as the nodes of the tree, with the attachments being presented as the leaves of the tree.

12.4 Page-level navigation

12.4.1 General

This subclause 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 numerals, 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. Labelling ranges shall not overlap, so that each page shall have only one label. 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 can be labelled 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 catalog dictionary (see 7.7.2, "Document catalog dictionary"). 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

in that range. The tree shall include a value for page index 0. "Table 161 — Entries in a page label dictionary" shows the contents of a page label dictionary.

Table 161 — 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 labelled 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 labelled 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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. Interactive PDF processors 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 catalog dictionary (see 7.7.2, "Document catalog dictionary") holds an array of thread dictionaries ("Table 162 — Entries in a thread dictionary") defining the document's articles. Each individual bead within a thread shall be represented by a bead dictionary ("Table 163 — Entries in a bead dictionary"). 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 162 — 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").
Metadata	stream	(*Optional; PDF 2.0; shall be an indirect reference*) A metadata stream containing information about the thread, such as its title, author, and creation date (see 14.3.2, "Metadata streams").

Table 163 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 in default user space.

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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.4.4 Presentations

12.4.4.1 General

Some interactive PDF processors 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 (12.4.4.2, "Sub-page navigation").

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 164 — Entries in a transition dictionary" shows the contents of the transition dictionary. (Some of the entries shown are needed only for certain transition styles, as indicated in the table.)

© ISO 2020 – All rights reserved

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 1 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 164 — 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 centre, 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 centre, 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 (upper case i) *O* Outward from the centre of the page (upper case o) Default value: *I*.
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 2 "Figure 76 — Presentation timing" 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.

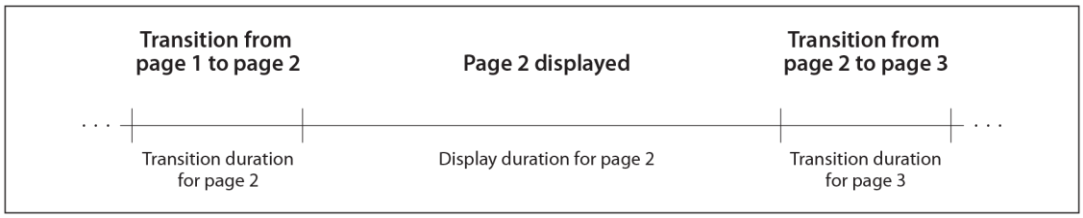


Figure 76 — Presentation timing

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 centre 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 Interactive PDF processors need to 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 165 — Entries in a navigation node dictionary") 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 31 — Entries in a page object").

NOTE 3 An interactive PDF processor needs to respect navigation nodes only when in presentation mode (see 12.4.4, "Presentations").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 165 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
Prev	dictionary	*(Optional)* The previous navigation node, if any.
Dur	number	*(Optional)* The maximum number of seconds before the interactive PDF processor shall automatically advance forward to the next navigation node. If this entry is not specified, no automatic advance shall occur.

An interactive PDF processor 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:

c) 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.

d) 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 12.6.4.15, "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 interactive PDF processor 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*, *link* or *rich media* 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".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 popup window displaying a text note ("Figure 77 — Open annotation") or by playing a sound or a movie.

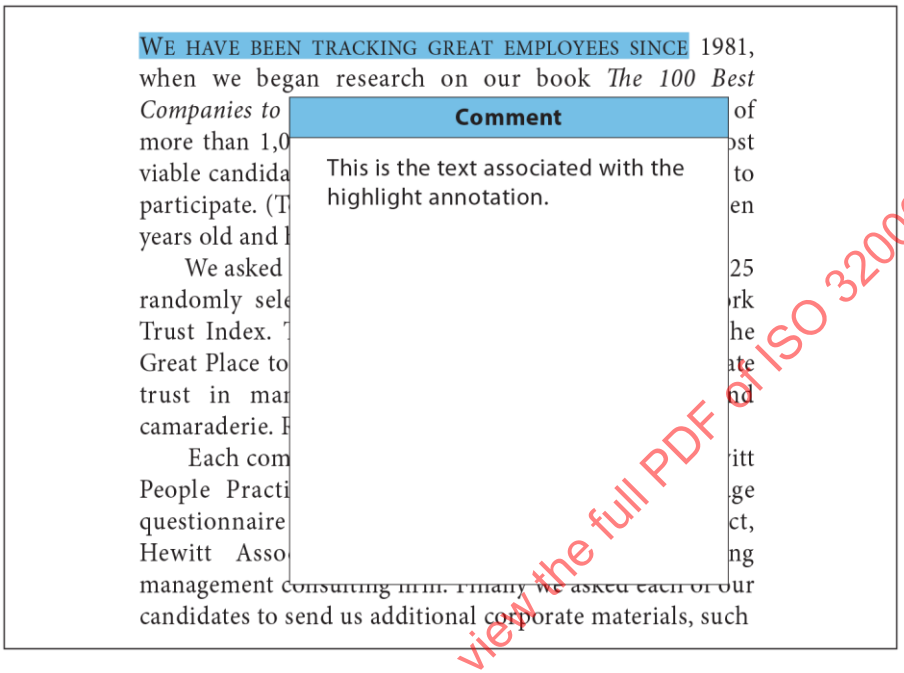


Figure 77 — Open annotation

Interactive PDF processors 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 31 — Entries in a page object"). This entry has the following values:

- *R* (row order): Annotations shall be visited in rows running horizontally across the page. The direction within a row is defined 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 is determined in a manner of the interactive PDF processor's choosing.
- *A* (annotation array order): All annotations shall be visited in the order in which they appear in the page **Annots** array. New in PDF 2.0. (See "Table 31 — Entries in a page object".)
- *W* (widgets order): **Widget** annotations shall be visited in the order in which they appear in the page **Annots** array, followed by other annotation types in row order. (See "Table 31 — Entries in

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

a page object".) New in PDF 2.0. For information about row order, see the R (row order) description.

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 PDF processor shall provide annotation handlers for all of the conforming annotation types. The set of annotation types is extensible. An interactive PDF processor shall provide certain expected behaviour for all annotation types that it does not recognise, 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 166 — Entries common to all annotation dictionaries" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 166 — 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 171 — Annotation types" 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 alternative 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 12.6.4.14, "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 interactive PDF processors shall accept and display a string in any format.

© ISO 2020 – All rights reserved

Key	Type	Value
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"). A PDF writer shall include an appearance dictionary when writing or updating the PDF file except for the two cases listed below. Every annotation (including those whose **Subtype** value is *Widget*, as used for form fields), except for the two cases listed below, shall have at least one appearance dictionary. • Annotations where the value of the **Rect** key consists of an array where the value at index 1 is equal to the value at index 3 and the value at index 2 is equal to the value at index 4. NOTE (2020) The bullet point above was changed from "or" to "and" in this document to match requirements in other published ISO PDF standards (such as PDF/A). • Annotations whose **Subtype** value is *Popup, Projection* or *Link*.
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 12.5.5, "Appearance streams").
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"). The dash phase shall not be specified and shall be assumed to be 0. 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]*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 popup 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 14.7.5.4, "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 not be drawn. (See 8.11.3.3, "Optional content in XObjects and annotations".)
AF	array of dictionaries	*(Optional; PDF 2.0)* An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this annotation). See 14.13, "Associated files" and 14.13.9, "Associated files linked to an annotation dictionary" for more details.
ca	number	*(Optional; PDF 2.0)* When regenerating the annotation's appearance stream, this is the opacity value (11.2, "Overview of transparency") that shall be used for all nonstroking operations on all visible elements of the annotation in its closed state (including its background and border) but not the popup window that appears when the annotation is opened. Default value: *1.0* The specified value shall not be used if the annotation has an appearance stream (see 12.5.5, "Appearance streams"); in that case, the appearance stream shall specify any transparency. If no explicit appearance stream is defined for the annotation, and the processor is not able to regenerate the appearance, the annotation 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
CA	number	*(Optional; PDF 1.4, PDF 2.0 for non-markup annotations)* When regenerating the annotation's appearance stream, this is the opacity value (11.2, "Overview of transparency") that shall be used for stroking all visible elements of the annotation in its closed state, including its background and border, but not the popup window that appears when the annotation is opened. If a **ca** entry is not present in this dictionary, then the value of this **CA** entry shall also be used for nonstroking operations as well. Default Value: *1.0* The specified value shall not be used if the annotation has an appearance stream (12.5.5, "Appearance streams"); in that case, the appearance stream shall specify any transparency. If no explicit appearance stream is defined for the annotation, and the processor is not able to regenerate the appearance, the annotation 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.
BM	name	*(Optional; PDF 2.0)* The blend mode that shall be used when painting the annotation onto the page (see 11.3.5, "Blend Mode" and 11.6.3, "Specifying Blending Colour Space and Blend Mode"). If this key is not present, blending shall take place using the **Normal** blend mode. The value shall be a name object, designating one of the standard blend modes listed in "Table 134 — Standard separable blend modes" and "Table 135 — Standard non-separable blend modes" in 11.3.5, "Blend mode".
Lang	text string	(*Optional; PDF 2.0*) A language identifier overriding the document's language identifier to specify the natural language for all text in the annotation except where overridden by other explicit language specifications (see 14.9.2, "Natural language specification").

A PDF reader shall render the appearance dictionary without regard to any other keys and values in the annotation dictionary and shall ignore the values of the **C, IC, Border, BS, BE, BM, CA, ca, H, DA, Q, DS, LE, LL, LLE,** and **Sy** keys.

NOTE Requiring an appearance dictionary for each annotation ensures the reliable rendering of the annotations.

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 167 — Annotation flags" shows the meanings of the flags; all other bits of the integer shall be set to 0.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 167 — Annotation flags

Bit position	Name	Meaning
1	Invisible	Applies only to annotations which do not belong to one of the standard annotation types and for which no annotation handler is available. If set, do not render the unknown annotation and do not print it even if the Print flag is set. If clear, render 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 render 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 render auxiliary popup information similar in function to online help systems.
3	Print	*(PDF 1.2)* If set, print the annotation when the page is printed unless the Hidden flag is also set. If clear, never print the annotation, regardless of whether it is rendered on the screen. If the annotation does not contain any appearance streams this flag shall be ignored. NOTE 2 This can be useful for annotations representing interactive push-buttons, 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 render 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 rendered 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 226 — Entries common to all field dictionaries").
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Bit position	Name	Meaning
9	ToggleNoView	*(PDF 1.5)* If set, invert the interpretation of the NoView flag for annotation selection and mouse hovering, causing the annotation to be visible when the mouse pointer hovers over the annotation or when the annotation is selected.
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 is defined 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 PDF processor 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 78 — Coordinate adjustment with the NoRotate flag" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

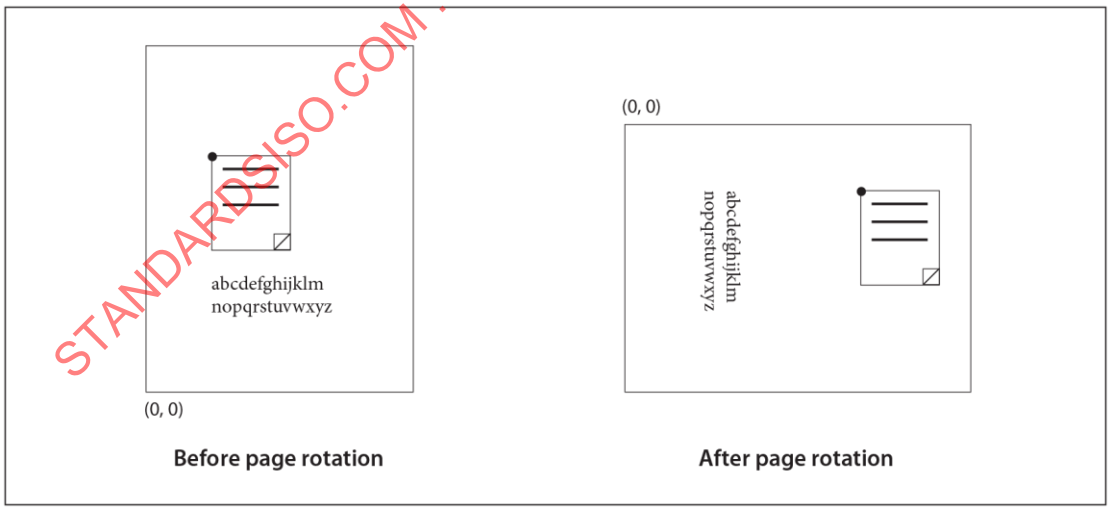


Figure 78 — Coordinate adjustment with the NoRotate flag

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 ("Table 166 — Entries

© ISO 2020 – All rights reserved

common to all annotation dictionaries"). 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 168 — Entries in a border style dictionary" summarises 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 168 — 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 be drawn. Default value: *1*.
S	name	*(Optional)* The border style: *S* (Solid) A solid rectangle surrounding the annotation. Default value. *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. An interactive PDF processor shall tolerate other border styles that it does not recognise and shall use the default value (which is *S*).
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 shall not be 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]*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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, free text annotations may also have a **BE** entry "Table 169 — Entries in a border effect dictionary" that describes the entries in a border effect dictionary.

© ISO 2020 – All rights reserved

Table 169 — Entries in a border effect dictionary

Key	Type	Value
S	name	*(Optional)* A name representing the border effect to apply. Values are: *C* The border should appear "cloudy"; that is, the border should be drawn as a series of convex curved line segments in a manner that simulates the appearance of a cloud. The width and dash array specified by **BS** shall be honoured. Default value: *S*. *S* No effect: the border shall be as described by the annotation dictionary's **BS** entry.
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 subclause shall be used to map from the coordinate system of the appearance XObject (as defined by its **Matrix** entry; see "Table 93 — Additional entries specific to a Type 1 form dictionary") to the annotation's rectangle in default user space:

Algorithm: appearance streams

1. 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.

2. 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.

3. **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:

   $$AA = Matrix \times A$$

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 be 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 the values of the **BM**, **ca** and **CA** entries in the annotation dictionary (see "Table 166 — Entries common to all annotation dictionaries") 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 166 — Entries common to all annotation dictionaries"). "Table 170 — Entries in an appearance dictionary" shows the contents of the appearance dictionary.

Table 170 — 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

>>
>>

where f*ormXObject1* and f*ormXObject2* define the check box's normal appearance in its checked and unchecked states, and *formXObject3* and *formXObject4* 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 166 — Entries common to all annotation dictionaries").

If a PDF processor does not have native support for a particular annotation type, the PDF processor shall render the annotation with its normal (**N**) appearance. PDF processors shall also attempt to provide reasonable behaviour (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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.5.6 Annotation types

12.5.6.1 General

PDF supports the standard annotation types listed in "Table 171 — Annotation types". The following subclauses describe each of these types in detail.

The values in the first column of "Table 171 — Annotation types" 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 subclause also provides more information about the value of the **Contents** entry for different annotation types.

Table 171 — Annotation types

Annotation type	Description	Markup	Discussed in subclause
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"

© ISO 2020 – All rights reserved

Annotation type	Description	Markup	Discussed in subclause
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"
Caret	*(PDF 1.5)* Caret annotation	Yes	12.5.6.11, "Caret annotations"
Stamp	*(PDF 1.3)* Rubber stamp annotation	Yes	12.5.6.12, "Rubber stamp annotations"
Ink	*(PDF 1.3)* Ink annotation	Yes	12.5.6.13, "Ink annotations"
Popup	*(PDF 1.3)* Popup annotation	No	12.5.6.14, "Popup annotations"
FileAttachment	*(PDF 1.3)* File attachment annotation	Yes	12.5.6.15, "File attachment annotations"
Sound	*(PDF 1.2; deprecated in PDF 2.0)* Sound annotation	Yes	12.5.6.16, "Sound annotations"
Movie	*(PDF 1.2; deprecated in PDF 2.0)* Movie annotation	No	12.5.6.17, "Movie annotations"
Screen	*(PDF 1.5)* Screen annotation	No	12.5.6.18, "Screen annotations"
Widget	*(PDF 1.2)* Widget annotation	No	12.5.6.19, "Widget annotations"
PrinterMark	*(PDF 1.4)* Printer's mark annotation	No	12.5.6.20, "Printer's mark annotations"
TrapNet	*(PDF 1.3; deprecated in PDF 2.0)* 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"
Projection	*(PDF 2.0)* Projection annotation	Yes	12.5.6.24, "Projection annotations"
RichMedia	*(PDF 2.0)* RichMedia annotation	No	13.7.2, "RichMedia annotations"

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.5.6.2 Markup annotations

As mentioned in 12.5.2, "Annotation dictionaries", the meaning of an annotation's **Contents** entry

© ISO 2020 – All rights reserved

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 alternative 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 172 — Additional entries in an annotation dictionary specific to markup annotations"). These annotations have text that appears as part of the annotation and may be displayed in other ways by an interactive PDF processor, 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 popup window that may contain text. The annotation's **Contents** entry specifies the text that shall be displayed when the popup 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 popup window but may also have associated text specified by the **Contents** entry.
- Projection annotations valid within the context of an associated run-time environment, such as an activated 3D model (see 12.5.6.24, "Projection annotations").

NOTE 1 The **RC** entry performs a similar role to the **Contents** entry except that the content's textual representation is formatted. When both **Contents** and **RC** entries are present, it is expected that the contents of both entries are textually equivalent.

When separating text into paragraphs, a CARRIAGE RETURN (0Dh) shall be used and not, for example, a LINE FEED character (0Ah).

NOTE 2 A subset of markup annotations is called text markup annotations (12.5.6.10, "Text markup annotations").

The remaining annotation types are not considered markup annotations:

- The popup annotation type shall not appear by itself; it shall be associated with a markup annotation that uses it to display text.

NOTE 3 The **Contents** entry for a popup annotation is relevant only if it has no parent; in that case, it represents the text of the annotation.

- For all other annotation types (**Link, Movie, Widget, RichMedia, PrinterMark,** and **TrapNet**), the **Contents** entry may provide an alternative 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 172 — Additional entries in an annotation dictionary specific to markup annotations" lists annotation dictionary entries that apply to all markup annotations.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 172 — Additional entries in an annotation dictionary 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 popup 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 popup annotation for entering or editing the text associated with this annotation.
RC	text string or text stream	*(Optional; PDF 1.5)* A rich text string (see *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*) that shall be displayed in the popup 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 12.7.8, "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 is considered a reply to the annotation specified by **IRT**. Interactive PDF processors 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
IT	name	*(Optional; PDF 1.6)* A name describing the *intent* of the markup annotation. Intents allow interactive PDF processors to distinguish between different uses and behaviours 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 an interactive PDF processor. Free text annotations ("Table 177 — Additional entries specific to a free text annotation"), line annotations ("Table 178 — Additional entries specific to a line annotation"), polygon annotations ("Table 181 — Additional entries specific to a polygon or polyline annotation"), (PDF 1.7) polyline annotations ("Table 181 — Additional entries specific to a polygon or polyline annotation") and stamp annotations ("Table 184 — Additional entries specific to a rubber stamp annotation") have defined intents, whose values are enumerated in the corresponding tables.

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 interactive PDF processors as acting on the entire group.

NOTE 4 A primary annotation can have replies that are not subordinate annotations; that is, that do not have an **RT** value of *Group*.

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. "Table 173 — Additional entries in markup annotation dictionaries specific to external data" lists additional markup annotation dictionary entries that apply to external data.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 173 — Additional entries in markup annotation dictionaries specific to external data

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** *(Required)* shall be *ExData*. NOTE (2020) This document clarified that the **Type** key is always required. **Subtype** *(Required)* a name specifying the type of data that the markup annotation shall be associated with. Values are: - *Markup3D (PDF 1.7)* for a 3D comment. Additional entries in this dictionary are listed in "Table 309 — Additional entries specific to a 3D annotation" - *3DM (PDF 2.0)* for a 3D measurement. Additional entries in this dictionary are listed in "Table 331 — Additional entries in a 3D measurement/markup dictionary for a 3D comment note and "Table 332 — Entries in the external data dictionary of a projection annotation". - *MarkupGeo (PDF 2.0)* for geospatial markup. This **Subtype** does not define any additional entries.

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 176 — Additional entries specific to a link annotation"). States shall be grouped into a number of state models, as shown in "Table 174 — Annotation states".

Table 174 — 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).

State changes made by a user shall be indicated in a text annotation with the following entries:

- The **T** entry (see "Table 172 — Additional entries in an annotation dictionary specific to markup annotations") shall specify the user.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- The **IRT** entry (see "Table 176 — Additional entries specific to a link annotation") shall refer to the original annotation.
- **State** and **StateModel** (see "Table 175 — Additional entries specific to a text annotation") 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 popup window containing the text of the note in a font and size chosen by the interactive PDF processor. Text annotations shall not scale and rotate with the page; they shall behave as if the NoZoom and NoRotate annotation flags (see "Table 167 — Annotation flags") were always set. "Table 175 — Additional entries specific to a text annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 175 — 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. Interactive PDF processors 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*.
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"

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 176 — Additional entries

© ISO 2020 – All rights reserved

specific to a link annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 176 — 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").
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 (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* (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 12.6.4.8, "URI actions") formerly associated with this annotation. When a PDF processor changes an annotation from a **URI** (12.6.4.8, "URI actions") to a go-to action (12.6.4.2, "Go-To actions"), it may use 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 × *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 PDF processor does not recognise it, or if any coordinates in the **QuadPoints** array lie outside the region specified by **Rect** then the activation region for the link annotation shall be defined by its **Rect** entry. NOTE The last paragraph above was clarified in this document (2020).
BS	dictionary	(*Optional; PDF 1.6*) A border style dictionary (see "Table 168 — Entries in a border style dictionary") specifying the line width and dash pattern that shall be used in drawing the annotation's border.

EXAMPLE The following example shows a link annotation that jumps to a destination elsewhere in the document.

```
93 0 obj
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        <</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 popup window, the text shall be always visible. "Table 177 — Additional entries specific to a free text annotation" shows the annotation dictionary entries specific to this type of annotation. Subclause 12.7.4.3, "Variable text", describes the process of using these entries to generate the appearance of the text in these annotations.

Table 177 — 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 12.7.4.3, "Variable text"). The annotation dictionary's **AP** entry, if present, shall take precedence over the **DA** entry (see "Table 170 — Entries in an appearance dictionary" 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* Centred *2* Right-justified Default value: *0* (left-justified).
RC	text string or text stream	*(Optional; PDF 1.5)* A rich text string (see *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*) that shall be used to generate the appearance of the annotation. NOTE As freetext annotations do not have an open state this cannot apply to the popup window as described for the **RC** key in "Table 172 — Additional entries in an annotation dictionary specific to markup annotations".
DS	text string	*(Optional; PDF 1.5)* A default style string, as described in *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*.
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 79 — Free text annotation with callout". Four numbers [x_1 y_1 x_2 y_2] represent the starting and ending coordinates of the line.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
IT	name	(*Optional; PDF 1.6*) A name describing the intent of the free text annotation (see also the **IT** entry in "Table 172 — Additional entries in an annotation dictionary specific to markup annotations"). 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 169 — Entries in a border effect dictionary") used in conjunction with the border style dictionary specified by the **BS** entry.
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 168 — Entries in a border style dictionary") specifying the line width and dash pattern that shall be used in drawing the annotation's border.
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 179 — Line ending styles" shows the possible line ending styles. Default value: *None*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

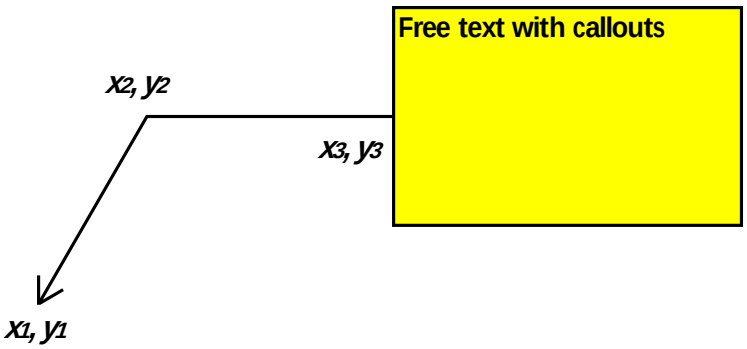


Figure 79 — Free text annotation with callout

© ISO 2020 – All rights reserved

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 popup window containing the text of the associated note. "Table 178 — Additional entries specific to a line annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 178 — 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 80 — Leader lines".
BS	dictionary	*(Optional)* A border style dictionary (see "Table 168 — Entries in a border style dictionary") specifying the width and dash pattern that shall be used in drawing the line.
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 179 — Line ending styles" shows the permitted 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 colour* that shall be used to fill the annotation's line endings (see "Table 179 — Line ending styles"). 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 80 — Leader lines". 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 80 — Leader lines". Default value: *0* (no leader line extensions).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 81 — Lines with captions appearing as part of the line" and "Figure 82 — Line with a caption appearing as part of the offset". 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 172 — Additional entries in an annotation dictionary specific to markup annotations"). 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.
LLO (capital letters 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 centred 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 266 — Entries in a measure dictionary") that shall specify the scale and units that apply to the line annotation.
CO (capital letters 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

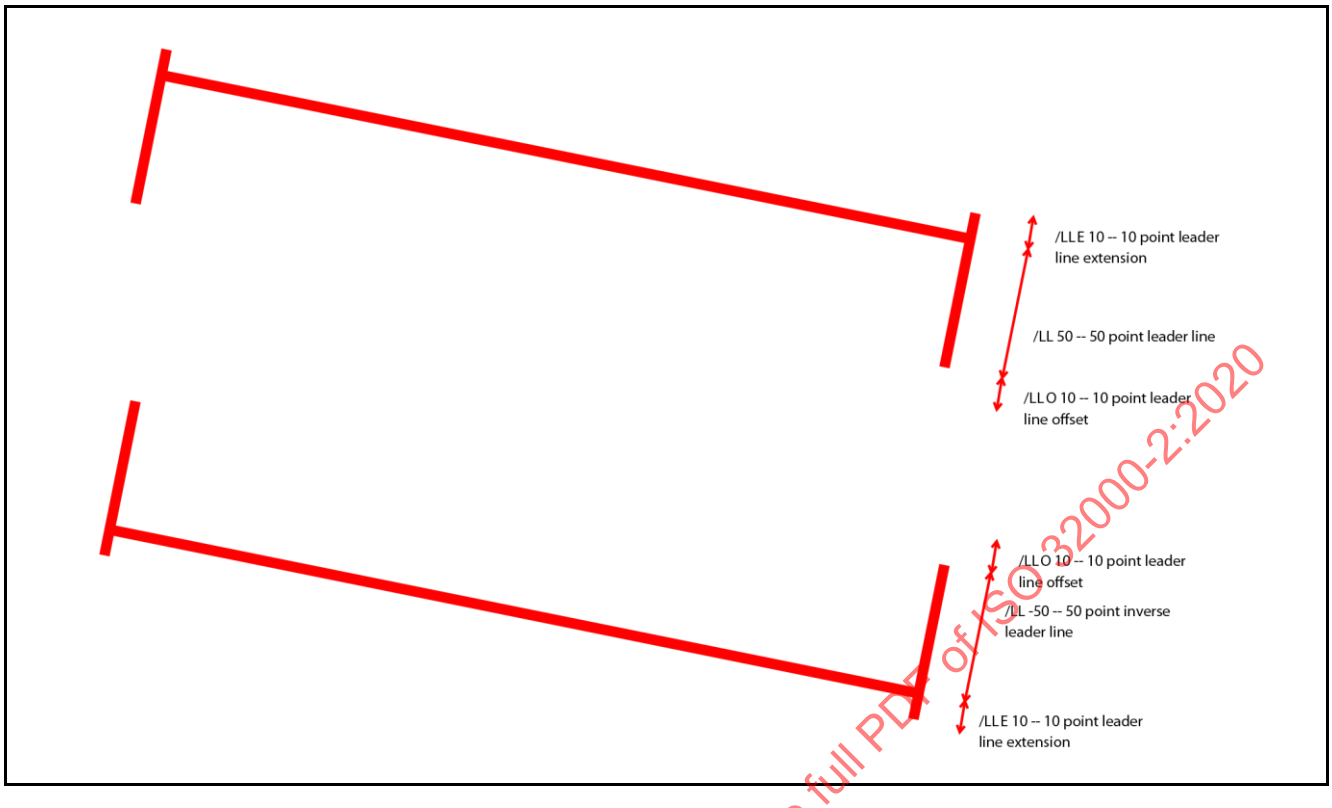


Figure 80 — Leader lines

"Figure 81 — Lines with captions appearing as part of the line" illustrates the effect of including a caption to a line annotation, which is specified by setting **Cap** to *true*.

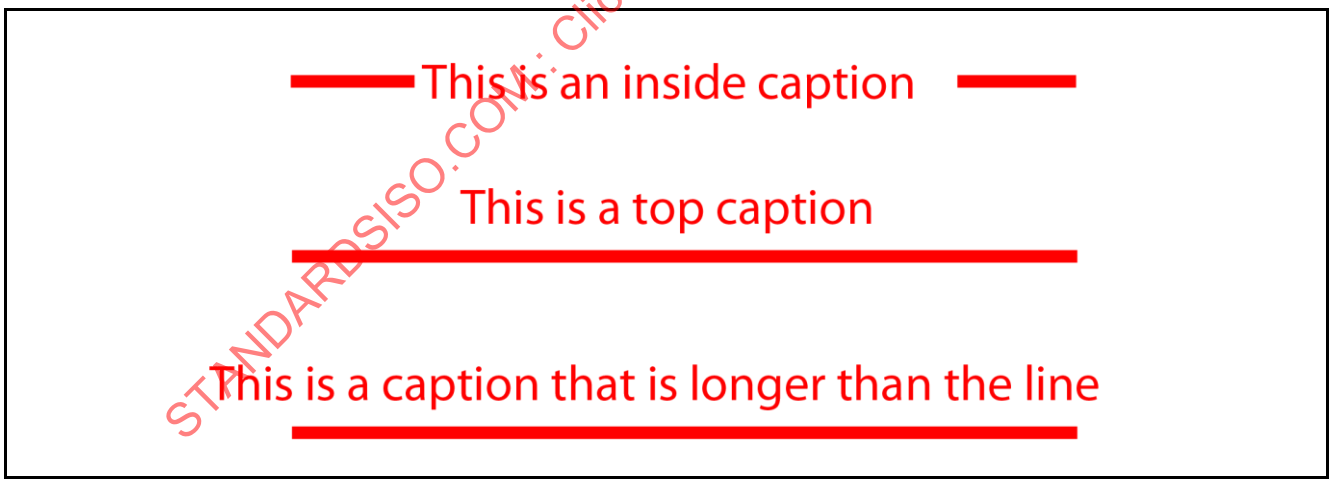


Figure 81 — Lines with captions appearing as part of the line

"Figure 82 — Line with a caption appearing as part of the offset" illustrates the effect of applying a caption to a line annotation that has a leader offset.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

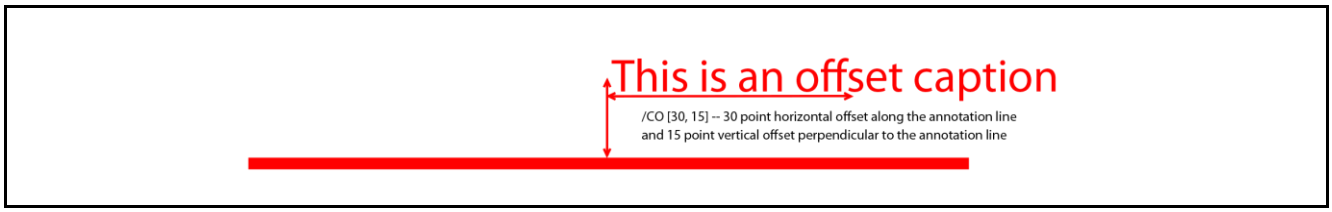


Figure 82 — Line with a caption appearing as part of the offset

Table 179 — Line ending styles

Name	Appearance	Description
Square		A square filled with the annotation's interior colour, if any
Circle		A circle filled with the annotation's interior colour, if any
Diamond		A diamond shape filled with the annotation's interior colour, if any
OpenArrow		Two short lines meeting in an acute angle to form an open
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 colour*, if any
None		No line ending
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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 popup 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 170 — Entries in an appearance dictionary").

"Figure 83 — Square and circle annotations" 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 180 — Additional entries specific to a square or circle annotation" shows the annotation dictionary entries specific to these types of annotations.

© ISO 2020 – All rights reserved

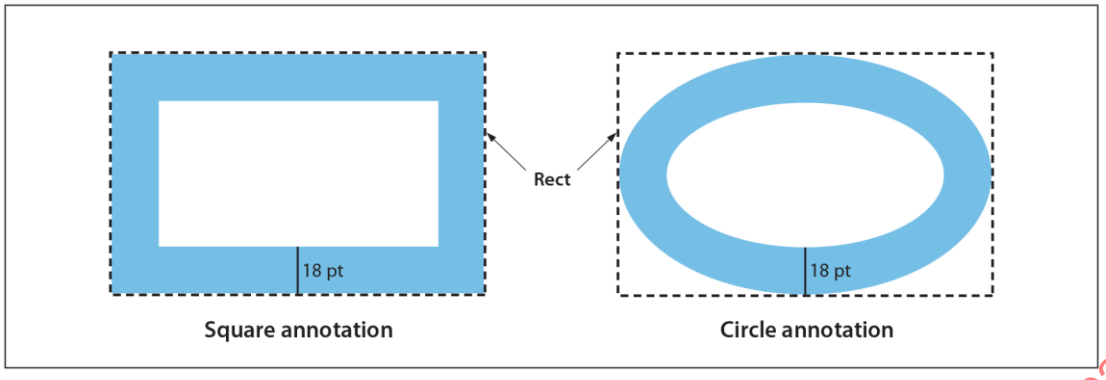


Figure 83 — Square and circle annotations

Table 180 — 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 168 — Entries in a border style dictionary") specifying the line width and dash pattern that shall be used in drawing the rectangle or ellipse.
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 colour* 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 169 — Entries in a border effect dictionary").
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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.5.6.9 Polygon and polyline annotations

Polygon annotations (*PDF 1.5*) display closed polygons on the page. Such polygons may have many 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 181 — 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 unless a **Path** key is present, in which case it shall be ignored)* An array of numbers specifying 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 179 — Line ending styles" shows the allowed values. Default value: [/None /None].
BS	dictionary	*(Optional)* A border style dictionary (see "Table 168 — Entries in a border style dictionary") specifying the width and dash pattern that shall be used in drawing the line.
IC	array	*(Optional)* 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 179 — Line ending styles"). 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** For *Polyline* annotations, the value of the **IC** key is used to fill only the line ending. However, for *Polygon* annotations, the value of the **IC** key is used to fill the entire shape, much as the **F** operator would fill a shape in a content stream.
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 169 — Entries in a border effect dictionary").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
IT	name	*(Optional; PDF 1.6)* A name that shall describe the intent of the polygon or polyline annotation (see also "Table 172 — Additional entries in an annotation dictionary specific to markup annotations"). 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 266 — Entries in a measure dictionary") that shall specify the scale and units that apply to the annotation.
Path	array	*(Optional; PDF 2.0)* An array of *n* arrays, each supplying the operands for a path building operator (**m**, **l** or **c**). If this key is present the **Vertices** key shall not be present. Each of the *n* arrays shall contain pairs of values specifying the points (*x* and *y* values) for a path drawing operation. The first array shall be of length 2 and specifies the operand of a **moveto** operator which establishes a current point. Subsequent arrays of length 2 specify the operands of **lineto** operators. Arrays of length 6 specify the operands for **curveto** operators. Each array is processed in sequence to construct the path. The current graphics state shall control the path width, dash pattern, etc.

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 popup window containing the text of the associated note. "Table 182 — Additional entries specific to text markup annotations" shows the annotation dictionary entries specific to these types of annotations.

Table 182 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 84 — QuadPoints specification"). The text shall be oriented with respect to the edge connecting points (x_1, y_1) and (x_2, y_2).

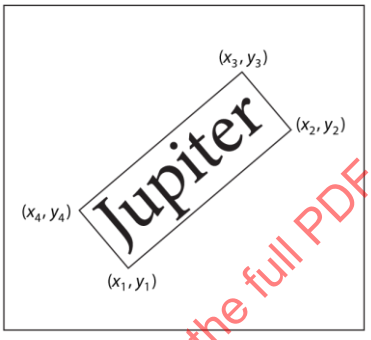


Figure 84 — QuadPoints specification

12.5.6.11 Caret annotations

A caret annotation (*PDF 1.5*) is a visual symbol that indicates the presence of text edits. "Table 183 — Additional entries specific to a caret annotation" lists the entries specific to caret annotations.

Table 183 — 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**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Sy	name	*(Optional)* A name specifying a symbol that shall be associated with the caret: *P* A new paragraph symbol (¶) shall be associated with the caret. *None* No symbol shall 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 popup window containing the text of the associated note. "Table 184 — Additional entries specific to a rubber stamp annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 184 — 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. PDF writers should include this entry and PDF readers should 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*. If the **IT** key is present and its value is not *Stamp*, this **Name** key shall not be present.
IT	name	*(Optional; PDF 2.0)* A name that shall describe the intent of the stamp. The following values shall be valid: *StampSnapshot* The appearance of this annotation has been taken from preexisting PDF content. *StampImage* The appearance of this annotation is an Image. *Stamp* The appearance of this annotation is a rubber stamp. Default value: *Stamp*

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 popup window containing the text of the associated note. "Table 185 — Additional entries specific to an ink annotation" shows the annotation dictionary entries specific to this type of annotation.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 185 — 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 168 — Entries in a border style dictionary") specifying the line width and dash pattern that shall be used in drawing the paths.
Path	array	*(Optional; PDF 2.0)* An array of *n* arrays, each supplying the operands for a path building operator (**m**, **l** or **c**). Each of the *n* arrays shall contain pairs of values specifying the points (*x* and *y* values) for a path drawing operation. The first array shall be of length 2 and specifies the operand of a **moveto** operator which establishes a current point. Subsequent arrays of length 2 specify the operands of **lineto** operators. Arrays of length 6 specify the operands for **curveto** operators. Each array is processed in sequence to construct the path. The current graphics state shall control the path width, dash pattern, etc.

12.5.6.14 Popup annotations

A *popup annotation* (*PDF 1.3*) displays text in a popup 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 172 — Additional entries in an annotation dictionary specific to markup annotations"). "Table 186 — Additional entries specific to a popup annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 186 — Additional entries specific to a popup annotation

Key	Type	Value
Subtype	name	*(Required)* The type of annotation that this dictionary describes; shall be *Popup* for a popup annotation.
Parent	dictionary	*(Optional; shall be an indirect reference)* The parent annotation with which this popup annotation shall be associated. If this entry is present, the parent annotation's **Contents**, **M**, **C**, and **T** entries (see "Table 170 — Entries in an appearance dictionary") shall override those of the popup annotation itself. NOTE See also the **Popup** entry in "Table 172 — Additional entries in an annotation dictionary specific to markup annotations".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Open	boolean	*(Optional)* A flag specifying whether the popup 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 can 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 187 — Additional entries specific to a file attachment annotation" 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. Interactive PDF processors shall use this entry rather than the optional **Desc** entry (*PDF 1.6*) in the file specification dictionary (see "Table 43 — Entries in a file specification dictionary") identified by the annotation's **FS** entry.

Table 187 — 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. PDF writers should include this entry and PDF readers should provide predefined icon appearances for at least the following standard names: *Graph*, *PushPin*, *Paperclip*, *Tag* Additional names may be supported as well. Default value: *PushPin*.

12.5.6.16 Sound annotations

The features described in this subclause are deprecated in PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

A *sound annotation* (*PDF 1.2*) is 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 188 — Additional entries specific to a sound annotation" shows the annotation dictionary entries specific to this type of annotation. Sound objects are discussed in 13.3, "Sounds".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 188 — 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. PDF writers should include this entry and PDF readers should provide predefined icon appearances for at least the standard names *Speaker* and *Mic*. Additional names may be supported as well. Default value: *Speaker*.

12.5.6.17 Movie annotations

The features described in this subclause are deprecated in PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

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 189 — Additional entries specific to a movie annotation" shows the annotation dictionary entries specific to this type of annotation. Movies are discussed in 13.4, "Movies".

Table 189 — 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 (12.6.4.10, "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. 12.6.4.14, "Rendition actions" discusses the relationship between screen annotations and rendition actions. "Table 190 — Additional entries

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

specific to a screen annotation" shows the annotation dictionary entries specific to this type of annotation.

Table 190 — 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 192 — Entries in an appearance characteristics dictionary"). The **I** entry of this dictionary provides the icon used in generating the appearance referred to by the screen annotation's **AP** entry.
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 190 — Additional entries specific to a screen annotation", screen annotations may use the common entries in the annotation dictionary (see "Table 166 — Entries common to all annotation dictionaries") 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 170 — Entries in an appearance dictionary") 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, "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 (12.7.4, "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 191 — Additional entries specific to a widget annotation" shows the annotation dictionary entries specific to this type of annotation; interactive forms and fields are discussed at length in 12.7.4, "Field dictionaries".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 191 — 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 colours used to display the contents of the annotation rectangle. *O* (Outline) Stroke the colours used to display the annotation border. That is, for each colour channel in the colour space used for display of the annotation value, colour values shall be transformed by the function $f(x) = 1 - x$ for display. *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 192 — Entries in an appearance characteristics dictionary") 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.
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 168 — Entries in a border style dictionary") specifying the width and dash pattern that shall be used in drawing the annotation's border.
Parent	dictionary	(*Required if this widget annotation is one of multiple children in a field; optional 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 192 — Entries in an appearance characteristics dictionary" shows the contents of this dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 192 — 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 push-button fields (see 12.7.5.2.2, "Push-buttons"), the **CA** entry may be used with any type of button field, including check boxes (see 12.7.5.2.3, "Check boxes") and radio buttons (12.7.5.2.4, "Radio buttons").
RC	text string	*(Optional; push-button 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; push-button fields only)* The widget annotation's *alternate (down) caption*, which shall be displayed when the mouse button is pressed within its active area.
I	stream	*(Optional; push-button 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; push-button 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; push-button 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; push-button fields only)* An icon fit dictionary (see "Table 250 — Entries in an icon fit dictionary") 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).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TP	integer	*(Optional; push-button 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

The features described in this subclause are deprecated in PDF 2.0.

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 colourants 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*) with a Fixed Print dictionary shall be used to represent graphics that are to be printed at a fixed size relative to the target media, and fixed relative position on the target media, regardless of the dimensions of that media. The **FixedPrint** entry of a watermark annotation dictionary (see "Table 193 — Additional entries specific to a watermark annotation") shall be a dictionary that contains values for specifying the size and position of the annotation (see "Table 194 — Entries in a fixed print dictionary").

Watermark annotations shall have no popup window nor other interactive elements. When displaying a watermark annotation on-screen, interactive PDF processors shall use the dimensions of the media box (see "Table 29 — Entries in the catalog dictionary") as the media dimensions so that the scroll and zoom behaviour is the same as for other annotations.

NOTE 1 Since many printing devices have nonprintable margins, such margins need to be taken into consideration when positioning watermark annotations near the edge of a page.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 193 — 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 194 — Entries in a fixed print dictionary") 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 31 — Entries in a page object").

Table 194 — 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 the media, this entry should be used to provide a reasonable offset to allow for unprintable 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 media. 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 media. Default value: *0*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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" (see 12.5.5, "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 media (e.g., printed page)

© ISO 2020 – All rights reserved

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 media size, watermark annotations with a *FixedPrint* entry shall be printed in the following manner:

- When page tiling is selected in a PDF processor (that is, a single PDF page is printed on multiple pages), watermark annotations shall be printed at the specified size and position on each page to ensure that the content of the watermark annotation 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 content of the watermark annotation does not overlap content from other pages, thus rendering it illegible.

NOTE 2 There is no guarantee that the location of a fixed print annotation on any given output page will cover content.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 PDF processor to remove all content identified by the redaction annotation, as well as the annotation itself. Interactive PDF processors 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 interactive PDF processors shall also be diligent in their consideration of all content that can exist in a PDF document. "Table 195 — Additional entries specific to a redaction annotation" shows the additional entries specific to redaction annotations.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 195 — 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 182 — Additional entries specific to text markup annotations" 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 **OverlayTex**t is present, ignored otherwise*) The appearance string that shall be used in formatting the overlay text when it is drawn after the affected content has been removed (see 12.7.4.3, "Variable text"). This entry is ignored if the **RO** entry is present.
Q	integer	(*Optional*) A code specifying the form of quadding (justification) that shall be used in laying out the overlay text: *0* Left-justified *1* Centred *2* Right-justified This entry is ignored if the **RO** entry is present. Default value: *0* (left-justified).

12.5.6.24 Projection annotations

A *projection annotation (PDF 2.0)* is a markup annotation subtype (see 12.5.6.2, "Markup annotations") that has much of the functionality of other markup annotations. However, a projection annotation is only valid within the context of an associated run-time environment, such as an activated 3D model.

A projection annotation shall have a **Subtype** of *Projection*. The entries of a annotation dictionary for a projection annotation are those listed in "Table 166 — Entries common to all annotation dictionaries" and "Table 172 — Additional entries in an annotation dictionary specific to markup annotations".

Projection annotations provide a way to save 3D and other specialised measurements and comments as markup annotations. These measurements and comments then persist in the document.

When a projection annotation is used in conjunction with a 3D measurement (13.6.7.4, "3D measurements and projection annotations"), it has an **ExData** dictionary with a **Subtype** of *3DM*. (See "Table 332 — Entries in the external data dictionary of a projection annotation".) Otherwise, the **ExData** dictionary is optional.

A projection annotation with a **Rect** entry that has zero height or zero width shall not have an **AP** dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.5.6.25 3D and RichMedia annotations

3D and RichMedia annotations are defined in 13.6.2, "3D annotations" and 13.7.2, "RichMedia annotations".

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 outline item dictionary (see "Table 151 — Entries in an outline item dictionary") and the dictionaries of some annotation types (see "Table 176 — Additional entries specific to a link annotation", "Table 190 — Additional entries specific to a screen annotation", "Table 191 — Additional entries specific to a widget annotation" and "Table 249 — Entries in an FDF field dictionary") 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 catalog dictionary (7.7.2, "Document catalog dictionary") 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 196 — Entries common to all action dictionaries" 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 196 — 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 201 — Action types" 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 Note 1 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 can 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 can contain an array of actions, each of which in turn can have a **Next** entry of its own. The actions can 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. It is recommended that interactive PDF processors attempt to provide reasonable

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

behaviour in anomalous situations. For example, self-referential actions ought not be executed more than once, and actions that close the document or otherwise render the next action impossible ought to terminate the execution sequence. Applications need 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 12.6.4.15, "Transition actions".

NOTE 2 It is recommended that no action 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

Selected types of annotations, page objects, or (beginning with PDF 1.3) interactive form fields 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 catalog dictionary (see 7.7.2, "Document catalog dictionary") may also contain an **AA** entry for trigger events affecting the document as a whole. "Table 197 — Entries in an annotation's additional-actions dictionary", "Table 198 — Entries in a page object's additional-actions dictionary", "Table 199 — Entries in a form field's additional-actions dictionary" and "Table 200 — Entries in the document catalog's additional-actions dictionary" 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 197 — Entries in an annotation's additional-actions dictionary"). 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.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 197 — 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 170 — Entries in an appearance dictionary").
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 198 — Entries in a page object's additional-actions dictionary") and the **OpenAction** entry in the document **Catalog** (see "Table 29 — Entries in the catalog dictionary"), if such actions are present.
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 198 — Entries in a page object's additional-actions dictionary"), 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 interactive PDF processor's user interface.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 198 — 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 dictionary (see 7.7.2, "Document catalog dictionary") 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 199 — Entries in a form field's additional-actions dictionary

Key	Type	Value
K	dictionary	*(Optional; PDF 1.3)* An ECMAScript 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)* An ECMAScript 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)* An ECMAScript 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)* An ECMAScript 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 12.7.3, "Interactive form dictionary").

Table 200 — Entries in the document catalog's additional-actions dictionary

Key	Type	Value
WC	dictionary	*(Optional; PDF 1.4)* An ECMAScript action that shall be performed before closing a document. (The name **WC** stands for "will close.")
WS	dictionary	*(Optional; PDF 1.4)* An ECMAScript action that shall be performed before saving a document. (The name **WS** stands for "will save.")
DS	dictionary	*(Optional; PDF 1.4)* An ECMAScript action that shall be performed after saving a document. (The name **DS** stands for "did save.")
WP	dictionary	*(Optional; PDF 1.4)* An ECMAScript action that shall be performed before printing a document. (The name **WP** stands for "will print.")

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
DP	dictionary	*(Optional; PDF 1.4)* An ECMAScript action that shall be performed after printing a document. (The name **DP** stands for "did print.")

Interactive PDF processors 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 preceding **E** and **D** events.
- 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 12.7.5.2, "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, actions triggered by this event can 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 (deprecated in PDF 2.0) in the interactive form dictionary (see 12.7.3, "Interactive form dictionary"), importation of **FDF** data (12.7.8, "Forms data format"), or **ECMAScript** actions (12.6.4.17, "ECMAScript 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 201 — Action types". The following subclauses describe each of these types in detail.

Table 201 — Action types

Action type	Description	Discussed in subclause
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"
GoToDp	("Go-to document part"; *PDF 2.0*) Go to a specified DPart in the current document.	12.6.4.5, "GoToDp action"

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Action type	Description	Discussed in subclause
Launch	Launch an application, usually to open a file.	12.6.4.6, "Launch actions"
Thread	Begin reading an article thread.	12.6.4.7, "Thread actions"
URI	Resolve a uniform resource identifier.	12.6.4.8, "URI actions"
Sound	*(PDF 1.2; deprecated in PDF 2.0)* Play a sound.	12.6.4.9, "Sound actions"
Movie	*(PDF 1.2; deprecated in PDF 2.0)* Play a movie.	12.6.4.10, "Movie actions"
Hide	*(PDF 1.2)* Set an annotation's Hidden flag.	12.6.4.11, "Hide actions"
Named	*(PDF 1.2)* Execute a predefined action.	12.6.4.12, "Named actions"
SubmitForm	*(PDF 1.2)* Send data to a uniform resource locator.	12.7.6.2, "Submit-form action"
ResetForm	*(PDF 1.2)* Set fields to their default values.	12.7.6.3, "Reset-form action"
ImportData	*(PDF 1.2)* Import field values from a file.	12.7.6.4, "Import-data action"
SetOCGState	*(PDF 1.5)* Set the states of optional content groups.	12.6.4.13, "Set-OCG-state actions"
Rendition	*(PDF 1.5)* Controls the playing of multimedia content.	12.6.4.14, "Rendition actions"
Trans	*(PDF 1.5)* Updates the display of a document, using a transition dictionary.	12.6.4.15, "Transition actions"
GoTo3DView	*(PDF 1.6)* Set the current view of a 3D annotation	12.6.4.16, "Go-To-3D-View actions"
JavaScript	*(PDF 1.3)* Execute an ECMAScript script.	12.6.4.17, "ECMAScript actions"
RichMediaExecute	*(PDF 2.0; RichMedia annotation only)* Specifies a command to be sent to the annotation's handler.	12.6.4.18, "Rich-Media-Execute actions"

12.6.4.2 Go-To actions

A *go-to action* changes the view to a specified destination (page, location, and magnification factor). "Table 202 — Additional entries specific to a go-to action" shows the action dictionary entries specific to this type of action.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 202 — 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").
SD	array	*(Optional; PDF 2.0)* The structure destination to jump to (see 12.3.2.3, "Structure destinations"). If present, the structure destination should take precedence over destination in the **D** entry.

© ISO 2020 – All rights reserved

NOTE Specifying a go-to action in the **A** entry of a link annotation or outline item (see "Table 176 — Additional entries specific to a link annotation" and "Table 151 — Entries in an outline item dictionary") has the same effect as specifying the destination directly with the **Dest** entry. For example, the link annotation shown in the Example in 12.5.6.5, "Link annotations" 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 203 — Additional entries specific to a remote go-to action" 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 203 — 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.
SD	Array	*(Optional; PDF 2.0)* The structure destination to jump to (see 12.3.2.3, "Structure destinations"). The first element in the array shall be a byte string representing a structure element **ID** in the remote document, instead of an indirect reference to a structure element dictionary. If present, the structure destination should take precedence over destination in the **D** entry.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 interactive PDF processor 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 204 — Additional entries specific to an embedded go-to action" shows the action dictionary entries specific to embedded go-to actions.

NOTE Embedded go-to actions work only for files of Type PDF.

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). Interactive PDF processors need to attempt to detect such cases and refuse to execute the action if one is 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 204 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 interactive PDF processor should act according to its preference.
T	dictionary	*(Optional if **F** is present; otherwise required)* A *target dictionary* (see "Table 205 — Entries specific to a target dictionary") 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 205 — 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 31 — Entries in a page object") 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 166 — Entries common to all annotation dictionaries").
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 ob                                  %Link to a child
      <</Type /Action
         /S /GoToE
         /D (Chapter 1)
         /T  <</R /C
               /N (Embedded document)>>
      >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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)
                        >>
                  >>
            >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        >>
endobj
```

12.6.4.5 GoToDp action

A *GoToDp action* changes the view to the **Start** page of a specified DPart (see "Table 409 — Entries in a DPart dictionary"). "Table 206 — Entries in a GoToDp dictionary" shows the action dictionary entries specific to this type of action.

Table 206 — Entries in a GoToDp dictionary

Key	Type	Value
S	name	*(Required; PDF 2.0)* The type of action that this dictionary describes; shall be *GoToDp* for a go-to document part action.
Dp	dictionary	*(Required; PDF 2.0)* The indirect reference to a DPart dictionary to go to.

EXAMPLE 1 Using a *GoToDp* action reference

```
92 0 obj
   <<
      /Parent 6 0 R
      /Start 100 0 R
      /End 101 0 R
      /Properties <</PartType (Cover)>>
   >>
endobj

93 0 obj
   <<
      /Type /Annot
      /Subtype /Link
      /Rect [71 717 190 734]
      /Border [16 16 1]
      /A
      <<
         /Type /Action
         /S /GoToDp
         /Dp 92 0 R
      >>
   >>
endobj
```

12.6.4.6 Launch actions

A *launch action* launches an application or opens or prints a document. "Table 207 — Additional entries specific to a launch action" shows the action dictionary entries specific to this type of action.

Previously, **Win**, **Mac**, and **Unix** entries allowed the action dictionary to include platform-specific parameters for launching the designated application, however, they are now deprecated with PDF 2.0. The **F** entry determines the file specification platform to be launched.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 207 — 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 interactive PDF processor does not understand any of the alternative entries, it shall do nothing.
Win	dictionary	*(Optional; deprecated in PDF 2.0)* A dictionary containing Microsoft Windows™ specific launch parameters (see "Table 208 — Entries in a Microsoft Windows™ launch parameter dictionary").
Mac	(undefined)	*(Optional; deprecated in PDF 2.0)* Mac OS–specific launch parameters; not yet defined.
Unix	(undefined)	*(Optional; deprecated in PDF 2.0)* 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 interactive PDF processor 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 208 — Entries in a Microsoft 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 Microsoft Windows™ specific 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.
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.6.4.7 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 209 — Additional entries specific to a thread action" shows the action dictionary entries specific to this type of action.

Table 209 — 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 dictionary (see 7.7.2, "Document catalog dictionary"). The first thread in the array has index 0. The title of the thread as specified in its thread information dictionary (see "Table 162 — Entries in a thread dictionary"). 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.

The title of the thread as specified in its thread information dictionary (see "Table 162 — Entries in a thread dictionary"). If two or more threads have the same title, the one appearing first in the document catalog's **Threads** array shall be used.

12.6.4.8 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 3986.*)

A *URI action* causes a URI to be resolved. "Table 210 — Additional entries specific to a URI action" shows the action dictionary entries specific to this type of action.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 210 — 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 UTF-8.
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:

$$(x_f = x_m - ll_x)$$
$$y_f = \mathrm{ur}_y - y_m$$

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.iso.org/intro?100,200

To support URI actions, a PDF document's catalog dictionary (see 7.7.2, "Document catalog dictionary") may include a **URI** entry whose value is a URI dictionary. Only one entry shall be defined for such a dictionary (see "Table 211 — Entry in a URI dictionary").

Table 211 — 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*.

NOTE The **Base** entry allows the URI of the document to be recorded in situations in which the document is 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.6.4.9 Sound actions

The features described in this subclause are deprecated with PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

A *sound action* (*PDF 1.2*) plays a sound through the computer's speakers. "Table 212 — Additional entries specific to a sound action" shows the action dictionary entries specific to this type of action. Sounds are discussed in 13.3, "Sounds".

Table 212 — 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 interactive PDF processor retains control, allowing no further user interaction other than cancelling 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.6.4.10 Movie actions

The features described in this subclause are deprecated with PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

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 are done explicitly.

The contents of a movie action dictionary are identical to those of a movie activation dictionary (see "Table 307 — Entries in a movie activation dictionary"), with the additional entries shown in "Table

© ISO 2020 – All rights reserved

213 — Additional entries specific to a movie action". 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 213 — 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. The dictionary shall include either an Annotation or a T entry but not both.
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.
Operation	name	*(Optional)* The operation that shall be performed on the movie: *Play* Start playing the movie, using the play mode specified by the dictionary's **Mode** entry (see "Table 307 — Entries in a movie activation dictionary"). 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.6.4.11 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 (see 12.5.5, "Appearance streams" and 12.6.3, "Trigger events") to display popup 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 214 — Additional entries specific to a hide action" shows the action dictionary entries specific to this type of action.

© ISO 2020 – All rights reserved

Table 214 — 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 12.7.4.2, "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*.

12.6.4.12 Named actions

"Table 215 — Named actions" lists several *named actions* (*PDF 1.2*) that interactive PDF processors shall support; further names may be added in the future.

Table 215 — 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 Interactive PDF processors can support additional, nonstandard named actions, but any document using them is not portable.

If the PDF processor encounters a named action that is inappropriate for a viewing platform, or if the viewer does not recognise the name, it shall take no action.

"Table 216 — Additional entries specific to named actions" shows the action dictionary entries specific to named actions.

Table 216 — 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 215 — Named actions").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.6.4.13 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 217 — Additional entries specific to a set-OCG-state action" shows the action dictionary entries specific to this type of action.

Table 217 — 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 99 — Entries in an optional content configuration dictionary") 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 processors are free to accumulate all state changes and apply only the net changes simultaneously to all affected groups before redrawing.

12.6.4.14 Rendition actions

A *rendition action* (*PDF 1.5*) controls the playing of multimedia content (see 13, "Multimedia features"). 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 an ECMAScript script that may perform custom operations.

"Table 218 — Additional entries specific to a rendition action" lists the entries in a rendition action dictionary.

Table 218 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
JS	text string or stream	(*Required if* OP *is not present; otherwise optional*) A text string or stream containing an ECMAScript 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 interactive PDF processor is unable to execute ECMAScripts. If **OP** has an unrecognised 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 interactive PDF processor 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 31 — Entries in a page object").

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 167 — Annotation flags") 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.

12.6.4.15 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. interactive PDF processors 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 interactive PDF processor 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 219 — Additional entries specific to a transition action"). Once this transition completes, drawing shall be suspended again.

Table 219 — 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 164 — Entries in a transition dictionary").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.6.4.16 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 220 — Additional entries specific to a go-to-3D-view action" shows the entries in a go-to-3D-view action dictionary.

Table 220 — 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 311 — Entries in a 3D stream dictionary"). • A text string matching the **IN** entry in one of the views in the **VA** array (see "Table 315 — Entries in a 3D view dictionary"). • 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 311 — Entries in a 3D stream dictionary") 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.

12.6.4.17 ECMAScript actions

JavaScript is referred to as ECMAScript throughout this document and is defined by ISO/DIS 21757-1. For backwards compatibility reasons the term JavaScript is retained in keywords.

Upon invocation of an ECMAScript action, a PDF processor shall execute a script that is written in the ECMAScript programming language. ECMAScript extensions described in ISO/DIS 21757-1 shall also be allowed. Depending on the nature of the script, various interactive form fields in the document may update their values or change their visual appearances. ISO/DIS 21757-1 defines the contents and effects of ECMAScript scripts. "Table 221 — Additional entries specific to an ECMAScript action" shows the action dictionary entries specific to this type of action.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 221 — Additional entries specific to an ECMAScript action

Key	Type	Value
S	name	*(Required)* The type of action that this dictionary describes; shall be *JavaScript* for an ECMAScript action.
JS	text string or text stream	*(Required)* A text string or text stream containing the ECMAScript script to be executed.

To support the use of parameterized function calls in ECMAScript 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 ECMAScript actions. When the document is opened, all of the actions in this name tree shall be executed, defining ECMAScript functions for use by other scripts in the document.

NOTE The name strings associated with individual ECMAScript actions in the name dictionary serve merely as a convenient means for organising and packaging scripts. The names are arbitrary and need not bear any relation to the ECMAScript name space.

12.6.4.18 Rich-Media-Execute actions

A *rich-media-execute action* identifies a rich media annotation and specifies a command to be sent to that annotation's handler (see Clause 13.7, "

Rich media". "Table 222 — Additional entries specific to a rich-media-execute action" shows the entries in a rich-media-execute action dictionary.

Table 222 — Additional entries specific to a rich-media-execute action

Key	Type	Value
S	name	*(Required; PDF 2.0)* The type of action that this dictionary describes; shall be *RichMediaExecute* for a rich-media-execute action.
TA	dictionary	*(Required; PDF 2.0)* An indirect object reference to an annotation dictionary (of Subtype *RichMedia*) upon which to execute the command.
TI	dictionary	*(Optional; PDF 2.0)* A dictionary that shall be an indirect object reference to a *RichMediaInstance dictionary* that is also present in the Instances array of the annotation.
CMD	dictionary	*(Required; PDF 2.0)* A *RichMediaCommand dictionary* containing the command name and arguments to be executed when the rich-media-execute action is invoked. See "Table 223 — Entries in a RichMediaCommand dictionary".

The RichMediaCommand dictionary contains a command name and optional arguments to be passed to the annotation handler specific to the target instance specified by the **TI** key in the parent rich-media-execute action dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 223 — Entries in a RichMediaCommand dictionary

Key	Type	Value
Type	name	*(Optional; PDF 2.0)* The type of PDF object that this dictionary describes; shall be *RichMediaCommand*.
C	text string	*(Required; PDF 2.0)* A text string specifying the script command (a primitive ECMAScript function name). If the target instance is a 3D model, the call shall be made in the global context of the annotation's instance of the 3D ECMAScript engine.
A	(various)	*(Optional; PDF 2.0)* An object that specifies the arguments to the command. The object may either be a single typed value or an array of typed values, each an argument. Valid arguments are objects of type text string, integer, number, or boolean. Default value: no arguments.

12.7 Forms

12.7.1 General

A PDF document may contain both interactive and non-Interactive forms.

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.

NOTE Interactive forms are not to be confused with form XObjects (see 8.10, "Form XObjects"). Despite the similarity of names, the two are different, unrelated types of objects.

A *non-interactive form (PDF 1.7)* is a static representation of form fields. Such forms may have originally contained interactive fields such as text fields and radio buttons but were converted into non-interactive PDF files, they may represent form fields and/or data converted from external sources, or they may have been designed to be printed out and filled in manually. See 12.7.9, "Non-interactive forms".

12.7.2 Interactive Forms

Each field in a document's form shall be defined by a *field dictionary* (see 12.7.4, "Field dictionaries"). For purposes of definition and naming, the fields can be organised 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 in an interactive form 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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 Fields containing text whose contents are not known in advance will need to construct their appearance streams dynamically instead of defining them statically in an appearance dictionary; see 12.7.4.3, "Variable text".

12.7.3 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 catalog dictionary (see 7.7.2, "Document catalog dictionary"). "Table 224 — Entries in the interactive form dictionary" shows the contents of this dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 224 — 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; deprecated in PDF 2.0)* A flag specifying whether to construct appearance streams and appearance dictionaries for all widget annotations in the document (see 12.7.4.3, "Variable text"). Default value: *false*. A PDF writer shall include this key, with a value of *true*, if it has not provided appearance streams for all visible widget annotations present in the document. NOTE Appearance streams are required in PDF 2.0 and later.
SigFlags	integer	*(Optional; PDF 1.3)* A set of flags specifying various document-level characteristics related to signature fields (see "Table 225 — Signature flags", and 12.7.5.5, "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 12.7.4.3, "Variable text").
Q	integer	*(Optional)* A document-wide default value for the Q attribute of variable text fields (see 12.7.4.3, "Variable text").

© ISO 2020 – All rights reserved

Key	Type	Value
XFA	stream or array	*(Optional; deprecated in PDF 2.0)* A stream or array containing an XFA *resource*, whose format shall conform to the *Data Package (XDP) Specification*. See Annex K, “XFA forms”.

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 12.7.5.5, "Signature fields"). Bit positions within the flag word shall be numbered from 1 (low-order) to 32 (high-order). "Table 225 — Signature flags" shows the meanings of the flags; all undefined flag bits shall be reserved and shall be set to 0.

Table 225 — Signature flags

Bit position	Name	Meaning
1	SignaturesExist	If set, the document contains at least one signature field. This flag allows an interactive PDF processor to enable user interface items (such as menu items or push-buttons) 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 PDF file is saved (written) in a way that alters its previous contents, as opposed to an incremental update. Merely updating the PDF file by appending new information to the end of the previous version is safe (see H.7, "Updating example"). Interactive PDF processors 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.4 Field dictionaries

12.7.4.1 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 organised 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 "Table 226 — Entries common to all field dictionaries" and "Table 227 — Field flags common to all field types". 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. The inheritable behavior of field dictionaries are similar to that of Pages (see 7.7.3.4, "Inheritance of page attributes"). An interactive PDF processor shall not limit the range of inheritance for field dictionaries. "Table 226 — Entries common to all field dictionaries" shows those entries that are common to all field dictionaries, regardless of type. Entries that pertain only to a

© ISO 2020 – All rights reserved

particular type of field are described in the relevant subclauses referred to in "Table 226 — Entries common to all field dictionaries".

Table 226 — 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 12.7.5.2, "Button fields") *Tx* Text (see 12.7.5.3, "Text fields") *Ch* Choice (see 12.7.5.4, "Choice fields") *Sig* (PDF 1.3) Signature (see 12.7.5.5, "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.
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	*(Required)* The partial field name (see 12.7.4.2, "Field names").
TU	text string	*(Optional; PDF 1.3)* An alternative 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 227 — Field flags common to all field types"). 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 12.7.6.3, "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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 227 — Field flags common to all field types" are common to all types of fields. **Flags** that apply only to specific field types are discussed in the subclauses describing those types. All undefined flag bits shall be reserved and shall be set to 0.

Table 227 — Field flags common to all field types

Bit position	Name	Meaning
1	ReadOnly	If set, an interactive PDF processor shall not allow a user to change the value of the field. Additionally, any associated widget annotations should not interact with the user; that is, they should not respond to mouse clicks nor change their appearance in response to mouse motions. NOTE: 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 12.7.6.2, "Submit-form action").
3	NoExport	If set, the field shall not be exported by a submit-form action (see 12.7.6.2, "Submit-form action").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.4.2 Field names

The **T** entry in the field dictionary (see "Table 226 — Entries common to all field dictionaries" 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.

A field dictionary that does not have a partial field name (**T** entry) of its own shall not be considered a field but simply a Widget annotation. Such annotations are different representations of the same underlying field; they should differ only in properties that specify their visual appearance. In addition, actual field dictionaries with the same fully qualified field name shall have the same field type (**FT**), value (**V**), and default value (**DV**).

© ISO 2020 – All rights reserved

EXAMPLE A field with the fully qualified field name of PersonalData.Address.ZipCode can have multiple unnamed children representing different Widget annotations in multiple locations. Such children, although visually different, would represent the same field value to users.

12.7.4.3 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 an ECMAScript.

In such cases, the PDF document cannot provide a statically defined appearance stream for displaying the field. Instead, the PDF processor shall construct an appearance stream dynamically at rendering time. The dictionary entries shown in "Table 228 — Additional entries common to all fields containing variable text" provide general information about the field's appearance that can be combined with the specific text it contains to construct an appearance stream.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 228 — 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* Centred *2* Right-justified Default value: *0* (left-justified).
DS	text string	*(Optional; PDF 1.5)* A default style string, as described in *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*.
RV	text string or text stream	*(Optional; PDF 1.5)* A rich text string, as described in *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*.

The new appearance stream becomes the normal appearance (**N**) in the appearance dictionary associated with the field's widget annotation (see "Table 170 — Entries in an appearance dictionary").

In PDF 1.5, form fields that have the RichText flag set (see "Table 231 — Field flags specific to text fields") specify formatting information as described in *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*. 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 initialised as follows:

© ISO 2020 – All rights reserved

- The resource dictionary (**Resources**) shall be created using resources from the interactive form dictionary's **DR** entry (see "Table 224 — Entries in the interactive form dictionary").
- 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 the Example in 12.7.5.3, "Text fields" for an example of their use.

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 — Graphics objects"). 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 224 — Entries in the interactive form dictionary"). A zero value for *size* means that the font shall be *auto-sized*: its size shall be computed as an implementation dependent function.

The default appearance string shall contain at most one **Tm** (text matrix) operator. If this operator is present, the interactive PDF processor 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 interactive PDF processor shall first copy any needed resources from the document's **DR** dictionary (see "Table 224 — Entries in the interactive form dictionary") 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 interactive PDF processor 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 12.7.5.2.3, "Check boxes", 12.7.5, "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.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.7.5 Field types

12.7.5.1 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 *push-buttons*, *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 subclauses describe each of these field types in detail.

12.7.5.2 Button fields

12.7.5.2.1 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 *push-button* is a purely interactive control that responds immediately to user input without retaining a permanent value (see 12.7.5.2.2, "Push-buttons").
- A *check box* toggles between two states, on and off (see 12.7.5.2.3, "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 12.7.5.2.4, "Radio buttons")

For button fields, bits 15, 16, 17, and 26 shall indicate the intended behaviour of the button field. An interactive PDF processor shall follow the intended behaviour, as defined in "Table 229 — Field flags specific to button fields" and clauses 12.7.5.2.2, "Push-buttons", and 12.7.5.2.4, "Radio buttons".

Table 229 — 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 push-button that does not retain a permanent value.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Bit position	Name	Meaning
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 behaviour as HTML radio buttons).

12.7.5.2.2 Push-buttons

A push-button field shall have a field type of *Btn* and the Pushbutton flag (see "Table 229 — Field flags specific to button fields") set to one. Because this type of retains no permanent value, it shall not use the **V** and **DV** entries in the field dictionary (see "Table 226 — Entries common to all field dictionaries").

12.7.5.2.3 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 229 — Field flags specific to button fields") 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**.

The **V** entry in the field dictionary (see "Table 226 — Entries common to all field dictionaries") holds a name object representing the check box's appearance state, which shall be used to select the appropriate appearance from the appearance dictionary. The value of the **V** key shall also be the value of the **AS** key. If they are not equal, then the value of the **AS** key shall be used instead of the **V** key to determine which appearance to use.

EXAMPLE 1 This example shows a typical check box definition.

```
1 0 obj
      <</Type /Annot
         /Subtype /Widget
         /Rect [100 100 120 120]
         /FT /Btn
         /T (Urgent)
         /V /Yes
         /AS /Yes
         /AP <</N <</Yes 2 0 R /Off 3 0 R>>
      >>
endobj

2 0 obj
      <</Type /XObject
         /Subtype /Form
         /BBox [0 0 20 20]
         /Resources 20 0 R
         /Length 104
      >>
stream
   q
      0 0 1 rg
      BT
         /ZaDb 12 Tf
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            0 0 Td
            (4) Tj
         ET
      Q
endstream
endobj

3 0 obj
      <</Type /XObject
         /Subtype /Form
         /BBox [0 0 20 20]
         /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 230 — Additional entry specific to check box and radio button fields"). 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 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 behaviour 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.

For radio buttons, the same behaviour 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 230 — 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 may use numerical position (starting with 0) of the annotation in the **Kids** array, encoded as a name object (for example: /0, /1). This allows distinguishing between the annotations even if two or more of them have the same value in the **Opt** array.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.7.5.2.4 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 229 — Field flags specific to button fields") 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 226 — Entries common to all field dictionaries") 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**. The value of the **V** key shall also be the value of the **AS** key. If they are not equal, then the value of the **AS** key shall be used instead of the **V** key to determine which appearance to use.

EXAMPLE This example shows the object definitions for a set of radio buttons.

```
10 0 obj                                %Radio button field
      <</Type /Annot
         /Subtype /Widget
         /Rect [100 100 120 120]
         /FT /Btn
         /Ff …                          %…Radio flag = 16, Pushbutton = 17 …
         /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
      <</Type /Annot
         /Subtype /Widget
         /Rect [200 200 220 220]
         /Parent 10 0 R
         /AS /Off
         /AP <</N <</cardbrand2 8 0 R
                    /Off 9 0 R
                 >>
             >>
      >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
endobj

8 0 obj                                 %Appearance stream for "on" state
      <</Type /XObject
         /Subtype /Form
         /BBox [0 0 20 20]
         /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
      <</Type /XObject
         /Subtype /Form
         /BBox [0 0 20 20]
         /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 encoding for non-Latin characters (see "Table 230 — Additional entry specific to check box and radio button fields").

12.7.5.3 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 231 — Field flags specific to text fields" shows the flags pertaining to this type of field. A text field shall have a field type of *Tx*. A conforming PDF file, and an interactive PDF processor shall obey the usage guidelines in "Table 231 — Field flags specific to text fields".

Table 231 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Bit position	Name	Meaning
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, it is imperative that PDF processors 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 232 — Additional entry specific to a text field") 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 *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*). If the field has a value, the **RV** entry of the field dictionary ("Table 228 — Additional entries common to all fields containing variable text") 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 12.7.4.3, "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.*
- For Forms Data Format (FDF) submission, the value of the **V** entry in the **FDF** field dictionary (see 12.7.8.3.2, "FDF fields") shall be a file specification (7.11, "File specifications") identifying the selected file.
- XML 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 226 — Entries common to all field dictionaries") and to fields containing variable text (see "Table 228 — Additional entries common to all fields containing variable text"), the field dictionary for a text field may contain the

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

additional entry shown in "Table 232 — Additional entry specific to a text field".

Table 232 — 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
      <</Type /Annot
         /Subtype /Widget
         /Rect [100 100 400 220]
         /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
      <</Type /XObject
         /Subtype /Form
         /BBox [0 0 300 120]
         /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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.5.4 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.

The various types of choice fields are distinguished by flags in the **Ff** entry, as shown in "Table 233 — Field flags specific to choice fields". "Table 234 — Additional entries specific to a choice field" shows the field dictionary entries specific to choice fields.

© ISO 2020 – All rights reserved

Table 233 — 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.
20	Sort	If set, the field's option items shall be sorted alphabetically. This flag is intended for use by PDF writers, not by PDF readers. PDF readers shall display the options in the order in which they occur in the **Opt** array (see "Table 234 — Additional entries specific to a choice field").
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.

Table 234 — 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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. 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 226 — Entries common to all field dictionaries") 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 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 favourite color)
             (Blue)
             ]
>>
```

12.7.5.5 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 235 — Additional entries specific to a signature field", as well as the standard entries described in "Table 226 — Entries common to all field dictionaries". 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 255 — Entries in a signature dictionary").

NOTE 1 This signature form field serves two primary purposes. The first is to define the form field that provides the visual signing properties for display, and it can also 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 (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. PDF processors shall treat such signatures as not visible. PDF processors 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 166 — Entries common to all annotation dictionaries", and annotation flags are described in "Table 167 — Annotation flags".

The location of a signature within a document can have a bearing on its legal meaning. For this reason, signature fields shall never refer to more than one annotation.

NOTE 3 If more than one location is associated with a signature, the meaning can become ambiguous.

For signature fields that are visible, the appearance dictionary (AP) for the widget annotation of these fields should be created at the time of signature creation. This dictionary defines the field's visual appearance on the page (see 12.5.5, "Appearance streams"), but the information included in the appearance dictionary shall not be used by a signature verification handler at the time of signature verification. It is there strictly for the purpose of providing a way for a human verifier to perform their own verification of the visual representation. A PDF processor shall not incorporate the validation status of a signature (e.g. a checkmark for passed or an X for failed) into the appearance of the signature field.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 235 — 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 236 — Entries in a signature field lock dictionary" lists the entries in this dictionary.
SV	dictionary	*(Optional; shall be an indirect reference; PDF 1.5)* A *seed value dictionary* (see "Table 237 — Entries in a signature field seed value dictionary") containing information that constrains the properties of a signature that is applied to this field.

The signature field lock dictionary (described in "Table 236 — Entries in a signature field lock dictionary") contains the names of form fields whose values shall no longer be changed after this signature has been signed.

Table 236 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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.
P	number	*(Optional; PDF 2.0)* The access permissions granted for this document. Valid values shall be: *1* No changes to the document are 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 are the same as for 2, as well as annotation creation, deletion, and modification; other changes shall invalidate the signature. There is no default value; absence of this key shall result in no effect on signature validation rules. If **MDP** permission is already in effect from an earlier incremental save section or the original part of the document, the number shall specify permissions less than or equal to the permissions already in effect based on signatures earlier in the document. That is, permissions can be denied but not added. If the number specifies greater permissions than an **MDP** value already in effect, the new number is ignored. If the document does not have an author signature, the initial permissions in effect are those based on the number 3. The new permission applies to any incremental changes to the document following the signature of which this key is part.

The value of the **SV** entry in the field dictionary is a seed value dictionary whose entries (see "Table 237 — Entries in a signature field seed value dictionary") provide constraining information that shall be used at the time the signature is applied. The **Ff** entry in this signature field seed value dictionary specifies whether the other entries in the dictionary shall be honoured or whether they are merely recommendations.

Table 237 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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**). For PDF 2.0 the following bit flags are added: *8* (**Lockdocument**); and *9* (**AppearanceFilter**). 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 (deprecated with PDF 2.0)*, *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 SHA-1 and this attribute shall be ignored.
V	integer	*(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 recognise all seed value dictionary entries in a PDF 1.5 file. A value of *2* specifies that it shall be able to recognise all seed value dictionary entries specified. A value of *3* specifies that it shall be able to recognise all seed value dictionary entries specified in PDF 2.0 and earlier. The **Ff** entry indicates whether this shall be treated as a required constraint.
Cert	dictionary	*(Optional)* A *certificate seed value dictionary* (see "Table 238 — Entries in a certificate seed value dictionary") containing information about the characteristics of the certificate that shall be used when signing.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 interactive PDF processors. 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 255 — Entries in a signature dictionary").
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 approval 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 257 — Entries in the DocMDP transform parameters dictionary"). 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 timestamp dictionary containing two entries: **URL** An ASCII string specifying the URL of a timestamping server, providing a timestamp that is compliant with *Internet RFC 3161* as updated by *Internet RFC 5816*. **Ff** An integer whose value is *1* (the signature shall have a timestamp) or *0* (the signature need not have a timestamp). Default value: *0*. NOTE 1 Please see 12.8.3.3, "CMS (PKCS #7) signatures" for more details about hashing.
LegalAttestation	array	*(Optional; PDF 1.6)* An array of text strings specifying possible legal attestations (see 12.8.7, "Legal content attestations"). The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
AddRevInfo	boolean	(*Optional; PDF 1.7*) A flag indicating whether revocation checking shall be carried out. If **AddRevInfo** is *true*, the PDF 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 those values the **AddRevInfo** value shall be *true* only if **SubFilter** is *adbe.pkcs7.detached* or *adbe.pkcs7.sha1*. If **SubFilter** is *adbe.x509.rsa_sha1*, this entry shall be omitted or set to *false*. Additional **SubFilter** values 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 2 Revocation information is carried in the signature data as specified by PKCS #7. See 12.8.3.3, "CMS (PKCS #7) signatures". NOTE 3 adbe.pkcs7.detached and adbe.pkcs7.sha1 are deprecated in PDF 2.0.
LockDocument	name	*(Optional; PDF 2.0)* A name value supplying the author's intent for whether the signing dialogue should allow the user to lock the document at the time of signing. The value shall be one of *true*, *false*, and *auto*, as follows: *true* the document should be locked at the time of signing. If the **Ff** entry indicates that **LockDocument** is not a required constraint, the user may choose to override this at the time of signing; otherwise, the document is locked after signing. *false* the document should not be locked after signing. Again, the required flag, **Ff**, determines whether this is a required constraint. *auto* the consuming application decided whether to present the lock user interface for the document and whether to honour the required flag, **Ff**, based on the properties of the document. Default value: *auto*
AppearanceFilter	text string	*(Optional; PDF 2.0)* A text string naming the appearance that shall be used when signing the signature field. interactive PDF processors may choose to maintain a list of named signature appearances. This text string provides authors with a means of specifying which appearance should be used to sign the signature field. If the required bit **AppearanceFilter** in **Ff** is set, the appearance shall be available to sign the document and is used.

A seed value dictionary may also include seed values for private entries belonging to multiple handlers. A given handler shall use only those private entries that are pertinent to itself and ignore any other private entries.

For optional keys that are not present, no constraint shall be placed upon the signature handler for that property when signing.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 238 — 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 *Internet RFC 5280*. The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.
SignaturePolicyOID	ASCII string	*(Optional: PDF 2.0)* The string representation of the **OID** of the signature policy to use when signing.
SignaturePolicyHashValue	string	*(Optional: PDF 2.0)* The computed hash value of the signature policy, computed the same way as hashValue of sigPolicyHash in clause 5.2.9 of CAdES (ETSI EN 319 122-1), according to the hash algorithm specified by **SignaturePolicyHashAlgorithm.**
SignaturePolicyHashAlgorithm	name	*(Optional: PDF 2.0)* The hash function used to compute the value of the **SignaturePolicyHashValue** entry. Entries shall be represented the same way as **SubFilter** values specified in "Table 260 — SubFilter value algorithm support".
SignaturePolicyCommitmentType	Array of ASCII strings	*(Optional: PDF 2.0)* If the **SignaturePolicyOID** is present, this array defines the commitment types that may be used within the signature policy. An empty string may be used to indicate that all commitments defined by the signature policy may be used.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 *Internet RFC 5280*. The key can be any valid 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 *Internet RFC 5280*. 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 *Internet RFC 5280*. The value of the corresponding flag in the **Ff** entry indicates whether this is a required constraint.
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, "Extending PDF". The default value is *Browser*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

If the **SignaturePolicyOID** is absent, the **SignaturePolicyHashValue**, **SignaturePolicyHashAlgorithm** and **SignaturePolicyCommitmentType** fields shall be ignored. If the **SignaturePolicyOID** is present but the **SignaturePolicyCommitmentType** is absent, all commitments defined by the signature policy may be used.

NOTE The above entries allow the creation of a signature-policy-identifier as in CAdES (ETSI 319 122-1). All rules defined in CAdES apply. In particular, CAdES allows the creation of a EPES signature when the signature policy hash is not available, therefore, the absence of the **SignaturePolicyHashValue** does not preclude the creation of a PAdES-EPES signature.

© ISO 2020 – All rights reserved

12.7.6 Form actions

12.7.6.1 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*

12.7.6.2 Submit-form action

Upon invocation of a submit-form action, an interactive PDF 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 240 — Flags for submit-form actions" shows the meanings of the flags; all undefined flag bits shall be reserved and shall be set to 0.

Table 239 — 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.
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 240 — Flags for submit-form actions"). 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 227 — Field flags common to all field types") is set. Fields with no values may also excluded, as dictated by the value of the IncludeNoValueFields flag; see "Table 240 — Flags for submit-form actions".
Flags	integer	*(Optional; inheritable)* A set of flags specifying various characteristics of the action (see "Table 240 — Flags for submit-form actions"). Default value: *0*.
CharSet	string	*(Optional; inheritable; PDF 2.0)* Supported values include: *utf-8*, *utf-16*, *Shift-JIS*, *BigFive*, *GBK*, or *UHC*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 240 — Flags for submit-form actions

Bit position	Name	Meaning
1	Include/Exclude	If clear, the **Fields** array (see "Table 239 — Additional entries specific to a submit-form action") 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 227 — Field flags common to all field types") 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 12.7.8, "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.
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 246 — Entries in the FDF dictionary"). If clear, the incremental updates shall not be included.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Bit position	Name	Meaning
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 media type application/pdf as defined by *Internet RFC 8118*. 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 ECMAScript code that processes it.
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 172 — Additional entries in an annotation dictionary specific to markup annotations") 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 popup 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 239 — Additional entries specific to a submit-form action") together with the Include/Exclude and IncludeNoValueFields flags in the **Flags** entry ("Table 240 — Flags for submit-form actions"). 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 12.7.4.2,

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

"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 226 — Entries common to all field dictionaries" and "Table 227 — Field flags common to all field types") 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:

- HTML Form format (described in the HTML 4.01 Specification).
- Forms Data Format (FDF), which is described in 12.7.8, "Forms data format".
- XFDF, a version of FDF based on XML as defined by ISO 19444-1.
- 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 12.7.4.2, "Field names"), and the value shall be specified by the **V** entry in its field dictionary.

For push-button 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 push-button 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.

12.7.6.3 Reset-form action

Upon invocation of a reset-form action, an interactive PDF 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 226 — Entries common to all field dictionaries"). If no default value is defined for a field, its **V** entry shall be removed. For fields that can have no value (such as push-buttons), the action has no effect. "Table 241 — Additional entries specific to a reset-form action" 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 241 — 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 242 — Flag for reset-form actions"). 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 242 — Flag for reset-form actions"). Default value: *0*.

Table 242 — Flag for reset-form actions

Bit position	Name	Meaning
1	Include/Exclude	If clear, the **Fields** array (see "Table 241 — Additional entries specific to a reset-form action") 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.6.4 Import-data action

Upon invocation of an import-data action, a PDF processor shall import data (see "Table 243 — Additional entries specific to an import-data action") from Forms Data Format (FDF), XFDF (XML-based Forms Data Format according to ISO 19444-1) or any other data format that it supports into the document's interactive form from a specified file.

Table 243 — 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, XFDF or any other data format file from which to import the data.

© ISO 2020 – All rights reserved

12.7.7 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 an ECMAScript *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 PDF processor, 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 31 — Entries in a page object").

12.7.8 Forms data format

12.7.8.1 General

This subclause 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 shall use the MIME media type application/vnd.fdf. On the Microsoft Windows™ and UNIX platforms, FDF files shall have the extension .fdf; on Mac OS, they shall have file type 'FDF'.

12.7.8.2 FDF file structure

12.7.8.2.1 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 85 — FDF file structure"):

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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

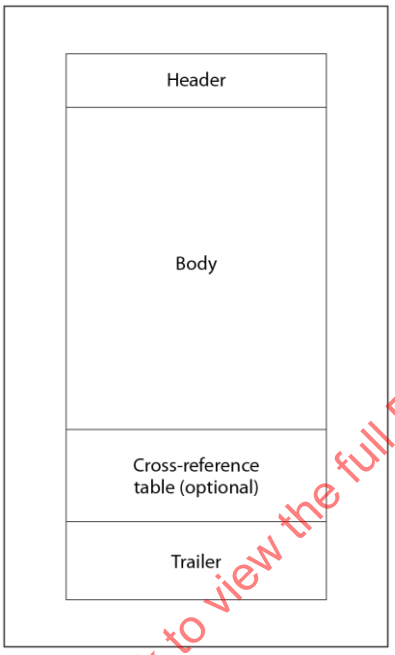


Figure 85 — FDF file structure

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.8.2.2 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 catalog dictionary (see 12.7.8.3, "FDF catalog").

12.7.8.2.3 FDF body

The body of an FDF file shall consist of a sequence of indirect objects representing the file's catalog dictionary (see 12.7.8.3, "FDF catalog") and any additional objects that the catalog dictionary 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 PDF processors. Just as in PDF, objects in FDF can be direct or indirect.

12.7.8.2.4 FDF trailer

The trailer of an FDF file enables an interactive PDF processor 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

© ISO 2020 – All rights reserved

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 shall be an indirect reference to the file's catalog dictionary (see "Table 244 — Entry in the FDF trailer dictionary"). The trailer may optionally contain additional entries for objects that are referenced from within the catalog dictionary.

Table 244 — Entry in the FDF trailer dictionary

Key	Type	Value
Root	dictionary	*(Required; shall be an indirect reference)* The catalog dictionary object for this FDF file (see 12.7.8.3, "FDF catalog").

Thus, the trailer has the overall structure

```
trailer
   <</Root c 0 R
      key2 value2
      …
      keyn valuen
   >>
%%EOF
```

where c is the object number of the file's catalog dictionary.

12.7.8.3 FDF catalog

12.7.8.3.1 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 12.7.8.2.4, "FDF trailer"). As shown in "Table 245 — Entries in the FDF catalog dictionary", the only required entry in the catalog dictionary is **FDF**; its value shall be an *FDF dictionary* ("Table 246 — Entries in the FDF dictionary"), which in turn shall contain references to other objects describing the file's contents. The catalog dictionary may also contain an optional **Version** entry identifying the version of the PDF specification to which this FDF file conforms. "Table 245 — Entries in the FDF catalog dictionary" describes the entries in the FDF catalog dictionary.

Table 245 — 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 12.7.8.2.2, "FDF header"). 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*).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
FDF	dictionary	*(Required)* The FDF dictionary for this file (see "Table 246 — Entries in the FDF dictionary").

Table 246 — 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 12.7.8.3.2, "FDF fields" in 12.7.8.3, "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 12.7.6.2, "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 12.7.8.3.3, "FDF pages") 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 249 — Entries in an FDF field dictionary") or field name that is a string and does not begin with the Unicode prefix ZERO WIDTH NO-BREAK SPACE (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 12.7.8.3.4, "FDF annotation dictionaries" in 12.7.8.3, "FDF catalog"). The array may include annotations of any of the standard types listed in "Table 171 — Annotation types" except **Link, Movie, Widget, PrinterMark, Screen,** and **TrapNet**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 12.7.6.2, "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)* An *ECMAScript dictionary* (see "Table 248 — Entries in the ECMAScript dictionary") defining document-level ECMAScript scripts.

Although deprecated in PDF 2.0, 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 247 — Additional entry in an embedded file stream dictionary for an encrypted FDF file" 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 a file encryption key in order to encrypt a document (revision 4 and earlier)" in 7.6.4.3, "File encryption key algorithm"; the first 5 bytes of the result shall be the file encryption key for the embedded FDF file.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 247 — 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; deprecated in PDF 2.0)* The revision number of the FDF encryption algorithm used to encrypt the file. This value shall be *1*.

The **JavaScript** entry in the FDF dictionary holds an ECMAScript dictionary containing ECMAScript scripts that shall be defined globally at the document level, rather than associated with individual fields. The dictionary may contain scripts defining ECMAScript 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 248 — Entries in the ECMAScript dictionary" shows the contents of this dictionary.

Table 248 — Entries in the ECMAScript dictionary

Key	Type	Value
Before	text string or text stream	*(Optional)* A text string or text stream containing an ECMAScript 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 an ECMAScript 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 an ECMAScript 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").
Doc	Array	*(Optional)* An array defining additional ECMAScript 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, organised 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 12.6.4.17, "ECMAScript actions" these scripts are used to define ECMAScript functions for use by other scripts in the document.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.7.8.3.2 FDF fields

Each field in an FDF file shall be described by an *FDF field dictionary*. "Table 249 — Entries in an FDF field dictionary" 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 226 — Entries common to all field dictionaries", "Table 228 — Additional entries common to all fields containing variable text", "Table 232 — Additional entry specific to a text field", and "Table 234 — Additional entries specific to a

© ISO 2020 – All rights reserved

choice field") or a widget annotation dictionary ("Table 170 — Entries in an appearance dictionary" and "Table 191 — Additional entries specific to a widget annotation"). 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 249 — 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 12.7.4.2, "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 12.7.5, "Field types" for further information.
Ff	integer	*(Optional)* A set of flags specifying various characteristics of the field (see "Table 227 — Field flags common to all field types", "Table 229 — Field flags specific to button fields", "Table 231 — Field flags specific to text fields", and "Table 233 — Field flags specific to choice fields"). 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.
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
AP	dictionary	*(Optional)* An appearance dictionary specifying the appearance of a push-button field (see 12.7.5.2.2, "Push-buttons"). The appearance dictionary's contents are as shown in "Table 170 — Entries in an appearance dictionary", 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 push-button field. This dictionary is similar to an appearance dictionary (see "Table 170 — Entries in an appearance dictionary"), except that the values of the **N, R**, and **D** entries shall all be named page reference dictionaries ("Table 253 — Entries in an FDF named page reference dictionary"). 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 250 — Entries in an icon fit dictionary") 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 (12.7.4.3, "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 in *Adobe XML Architecture, XML Forms Architecture (XFA) Specification, version 3.3*.

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 250 — Entries in an icon fit dictionary" shows the contents of this type of dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 250 — 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*.

© ISO 2020 – All rights reserved

Key	Type	Value
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, centring 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 centre 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*.

12.7.8.3.3 FDF pages

The optional **Pages** field in an FDF dictionary (see "Table 246 — Entries in the FDF dictionary") shall contain an array of FDF page dictionaries (PDF 1.3) describing new pages that shall be added to the target document. "Table 251 — Entries in an FDF page dictionary" shows the contents of this type of dictionary.

Table 251 — Entries in an FDF page dictionary

Key	Type	Value
Templates	array	*(Required)* An array of *FDF template dictionaries* (see "Table 252 — Entries in an FDF template dictionary") 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 252 — Entries in an FDF template dictionary" shows the contents of this type of dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 252 — Entries in an FDF template dictionary

Key	Type	Value
TRef	dictionary	*(Required)* A named page reference dictionary (see "Table 253 — Entries in an FDF named page reference dictionary") that shall specify the location of the template.
Fields	array	*(Optional)* An array of references to **FDF** field dictionaries (see "Table 249 — Entries in an FDF field dictionary") 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. If this flag is *true*, fields with such conflicting names shall be renamed to guarantee their uniqueness. If *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. See the Note in this subclause 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.

Although the **Rename** flag does not define a renaming algorithm, this might be implemented by a PDF processor renaming fields by prepending a page number, a template name, and an ordinal number to the field name.

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 253 — Entries in an FDF named page reference dictionary" shows the contents of this type of dictionary.

Table 253 — 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.

12.7.8.3.4 FDF annotation dictionaries

Each annotation dictionary in an FDF file shall have a **Page** entry (see "Table 254 — Additional entry for annotation dictionaries in an FDF file") that shall indicate the page of the source document to which the annotation is attached.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 254 — 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.

12.7.9 Non-interactive forms

Unlike interactive forms, non-interactive forms do not use widget annotations but are represented with page content. Non-interactive forms are defined by the PrintField attrib0ute (14.8.5.6, "PrintField attributes") for repurposing and accessibility purposes.

12.8 Digital signatures

12.8.1 General

A digital signature (*PDF 1.3*) may be used to verify the integrity of the document's contents using verification information related to a signer. 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, "Extending PDF".

Digital signatures in PDF support four activities:

- the addition of a digital signature to a document,
- the verification of the validity of a signature added to a document,
- the addition of DSS dictionaries and of validation related information (VRI) to allow for later verifications (see 12.8.4.4, "Validation-related information (VRI)"), and
- the addition of document timestamp dictionaries (DTS) to allow for later verifications (see 12.8.5, "Document timestamp (DTS) dictionary").

PDF 2.0 processors should support digital signatures based on the Cryptographic Message Syntax (CMS) and CAdES (ETSI EN 319 122).

Signature information shall be contained in a *signature dictionary*, whose entries are listed in "Table 255 — Entries in a signature dictionary". 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 using an appropriate signature handler. That signature handler shall match with the type of the signature that is intended to be created. It shall compute a *digest* using a cryptographic hash function over the data of the document, and store it in the document.

To verify the signature, an appropriate signature handler is required. That signature handler shall match with the type of the signature that has been created. The signer's certificate shall be determined and verified by the signature handler to match with any of the validation parameters and other

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

conditions. If the verification fails, the signature shall be considered invalid. The digest shall be re-computed and compared with the one stored in the document. Differences between the two indicates that modifications have been made since the document was signed and thus the signature shall be considered invalid.

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, the state of the document can be recreated as it existed at the time of signing.

There are two defined techniques for computing a digital signature of the contents of a PDF file:

- A *byte range digest* shall be computed over a range of bytes in the PDF file, that shall be indicated by the **ByteRange** entry in the signature dictionary. This range should be the entire PDF file, including the signature dictionary but excluding the signature value itself (the **Contents** entry). In case of multiple digital signatures this range shall be the sequence of bytes starting from the "%PDF-" comment at the beginning of the PDF document to the end of the "%%EOF" comment, possibly followed by an optional EOL marker, terminating the incremental update that adds the digital signature dictionary to the document. 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:

- At most one usage rights signature (*PDF 1.5, deprecated in PDF 2.0*). It shall only be applied if no other type of signature is already present. Its signature dictionary shall be referenced from the **UR3** (*PDF 1.6*) entry in the permissions dictionary, whose entries are listed in "Table 263 — Entries in a permissions dictionary", (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. When a usage rights signature is present, it is up to the PDF processor or to the signature handler to process it or not.
- At most one certification signature (also known as author 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 Table 263 — Entries in a permissions dictionary). The signature dictionary shall contain a signature reference dictionary (see "Table 256 — Entries in a signature reference dictionary") that has a **DocMDP** transform method. See 12.8.2.2, "DocMDP" for information on how these signatures shall be created and validated.
- One or more approval signatures (also known as recipient signatures). These shall follow the certification signature if one is present. The signature dictionary of an approval signature shall be the value of a signature field and shall contain a **ByteRange** entry.

NOTE 2 A signature dictionary for a certification or approval signature can also have a signature reference dictionary with a **FieldMDP** transform method; see 12.8.2.4, "FieldMDP".

- Any number of document timestamp signatures, see 12.8.5, "Document timestamp (DTS) dictionary". The timestamp signature dictionary of a document timestamp signature shall be the value of a signature field and shall contain a **ByteRange** entry.

A signature dictionary is used by all of these types of signatures.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 255 — Entries in a signature dictionary

Key	Type	Value
Type	name	*(Optional if Sig; Required if DocTimeStamp)* The type of PDF object that this dictionary describes; if present, shall be *Sig* for a signature dictionary or *DocTimeStamp* for a timestamp signature dictionary. The default value is: *Sig*.
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 PDF processor 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, "Extending PDF".
SubFilter	name	(*Optional*) A name that describes the encoding of the signature value and key information in the signature dictionary. A PDF processor may use any handler that supports this format to validate the signature. (*PDF 1.6*) The following values for public-key cryptographic signatures should be used: *adbe.x509.rsa_sha1*, *adbe.pkcs7.detached*, *adbe.pkcs7.sha1*, *ETSI.CAdES.detached* (PDF 2.0) and *ETSI.RFC3161* (PDF 2.0). (*PDF 2.0*) When the **Type** of this dictionary is *DocTimeStamp*, the **SubFilter** value should be *ETSI.RFC3161* and when the **Type** of this dictionary is *Sig* (possibly by default) any of the values may be used except *ETSI.RFC3161* (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, "Extending PDF"). The prefix "*adbe*" and the prefix "*ETSI*" have been registered by Adobe Systems and ETSI, respectively, and the subfilter names listed above and defined in 12.8.3, "Signature interoperability" and 12.8.5, "Document timestamp (DTS) dictionary" may be used by any developer. The values *adbe.x509.rsa_sha1* and *adbe.pkcs7.sha1* have been deprecated with PDF 2.0. To support backward compatibility, PDF readers should process these values for this key, but PDF writers shall not use this value for this key.
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, a DER-encoded CMS binary data object or a DER-encoded CMS SignedData binary data object. For document timestamp signatures, **Contents** shall be the TimeStampToken as specified in *Internet RFC 3161* as updated by *Internet RFC 5816*. The value of the **messageImprint** field within the TimeStampToken shall be a hash of the bytes of the document indicated by the **ByteRange** and the **ByteRange** shall specify the complete PDF file contents (excepting the **Contents** value). Space for the Contents value shall be allocated before the message digest is computed (see 7.3.4, "String objects").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Cert	byte string or array	*(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 CMS envelope in **Contents**. If **SubFilter** is *ETSI.CAdES.detached* or *ETSI.RFC3161*, this entry shall not be used, and the certificate chain shall be put in the DER-encoded CMS SignedData object 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; shall be direct objects*) 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. If **SubFilter** is *ETSI.CAdES.detached* or *ETSI.RFC3161*, the **ByteRange** shall cover the entire PDF file, including the signature dictionary but excluding the **Contents** value. When a byte range digest is present, all values in the signature dictionary shall be direct objects.
Reference	array	*(Optional; PDF 1.5)* An array of signature reference dictionaries (see "Table 256 — Entries in a signature reference dictionary"). If **SubFilter** is *ETSI.RFC3161*, this entry shall not be used.
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. If **SubFilter** is *ETSI.RFC3161*, this entry shall not be used.
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. If **SubFilter** is *ETSI.RFC3161*, this entry should not be used and should be ignored by a PDF processor.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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. If **SubFilter** is *ETSI.RFC3161*, this entry should not be used and should be ignored by a PDF processor. EXAMPLE 2 A timestamp can be embedded in a CMS binary data object (see 12.8.3.3, "CMS (PKCS #7) signatures").
Location	text string	*(Optional)* The CPU host name or physical location of the signing. If **SubFilter** is *ETSI.RFC3161*, this entry should not be used and should be ignored by a PDF processor.
Reason	text string	*(Optional)* The reason for the signing, such as (I agree...). If **SubFilter** is *ETSI.RFC3161*, this entry should not be used and should be ignored by a PDF processor.
ContactInfo	text string	*(Optional)* Information provided by the signer to enable a recipient to contact the signer to verify the signature. If **SubFilter** is *ETSI.RFC3161*, this entry should not be used and should be ignored by a PDF processor. EXAMPLE 3 A phone number.
R	integer	*(Optional; deprecated in PDF 2.0)* 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. If **SubFilter** is *ETSI.RFC3161*, this **V** value shall be *0* (possibly by default). 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. The use of this dictionary is defined by *Adobe PDF Signature Build Dictionary Specification*, which provides implementation guidelines.
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. If **SubFilter** is *ETSI.RFC3161*, this entry shall not be used.
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*. If **SubFilter** is *ETSI.RFC3161*, this entry shall not be used.

NOTE 3 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Prop_AuthType); key information (**Cert** and portions of **Contents** when the signature value is a CMS object, CMS object, or a document timestamp token); reference (**Reference** and **ByteRange**); and signature value (**Contents** when the signature value is a PKCS #1 object).

Table 256 — 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 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* (*Deprecated in PDF 2.0*) 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 subclauses specified previously for details on the individual transform parameter dictionaries.
Data	(various)	*(Required when **TransformMethod** is FieldMDP, shall be an indirect reference)* 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	*(Required)* A name identifying the algorithm that shall be used when computing the digest if not specified in the certificate. Valid values are *MD5, SHA1 SHA256, SHA384, SHA512* and *RIPEMD160*. NOTE (2020) The use of *MD5* and *SHA1* are deprecated with PDF 2.0. The **DigestMethod** key was also corrected to be required as no default value is defined. Default value for PDF 1.5-1.7: *MD5.*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclauses 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 a certification signature). A document can contain only one signature field that contains a **DocMDP** transform method. It enables the author to specify what changes shall be permitted to be made to 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 "Table 263 — Entries in a permissions dictionary").

A certification signature should have a legal attestation dictionary (see 12.8.7, "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 257 — Entries in the DocMDP transform parameters dictionary") 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 with the exception of subsequent DSS (see 12.8.4.3, "Document Security Store (DSS)") and/or document timestamp (see 12.8.5, "Document timestamp (DTS) dictionary") incremental updates. The values 2 and 3 shall permit modifications that are appropriate for form field or comment workflows as well as subsequent **DSS** and/or document timestamp incremental updates.

12.8.2.2.2 Validating signatures that use the DocMDP transform method

To validate a signature that uses the **DocMDP** transform method, a PDF processor 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 255 — Entries in a signature dictionary") is known to correspond to the state of the document at the time of signing. Therefore, PDF processors 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 257 — 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. Changes to a PDF that are incremental updates which include only the data necessary to add DSS's 12.8.4.3, "Document Security Store (DSS)" and/or document timestamps 12.8.5, "Document timestamp (DTS) dictionary" to the document shall not be considered as changes to the document as defined in the choices below. 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 features described in this subclause are deprecated with PDF 2.0.

The **UR** transform method (*deprecated in PDF 2.0*) 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 "Table 263 — Entries in a permissions dictionary"). The transform parameters dictionary (see "Table 258 — Entries in the UR transform parameters dictionary") 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.

A PDF processor that modifies a PDF, with a UR signature in excess of the rights that are granted by that signature, should remove that signature prior to writing the newly modified PDF.

UR3 (*PDF 1.6, deprecated in PDF 2.0*): The **ByteRange** entry in the signature dictionary (see "Table 255 — Entries in a signature dictionary") shall be present. First, a PDF processor 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 PDF processor 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 258 — 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).
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)* are also permitted (see "Table 263 — Entries in a permissions dictionary"): *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 summarises the comments (markup annotations) in a document.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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)* are also valid: *BarcodePlaintext* — Permits text form field data to be encoded as a plaintext two-dimensional barcode. *Online* — Permits 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 *false*, any possible restriction may be ignored. Default value: *false*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 259 — Entries in the FieldMDP transform parameters dictionary".

Table 259 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 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.
V	name	*(Optional; required for PDF 1.5 and later*) 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 PDF 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 257 — Entries in the DocMDP transform parameters dictionary").
- 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 236 — Entries in a signature field lock dictionary") 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 are 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 12.8.2.2.2, "Validating signatures that use the DocMDP transform method" for details.

12.8.3 Signature interoperability

12.8.3.1 General

It is intended that PDF processors allow interoperability between signature handlers; that is, a PDF file signed with a handler from one vendor should be able to be validated with a handler from a different vendor when they use the same set of validation parameters. PDF 2.0 defines new signature formats for PAdES signatures (PDF Advanced Electronic Signatures) as defined by ETSI EN 319 142-1 and ETSI EN 319 142-2.

If present, the **SubFilter** entry in the signature dictionary shall specify the encoding of the signature

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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, "Extending PDF" 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 and the European Telecommunications Standards Institute (ETSI). PDF 2.0 introduced Elliptic Curve Digital Signature Algorithm (ECDSA) support as defined by *Internet RFC 5480.*

"Table 260 — SubFilter value algorithm support" shows an overview of the **SubFilter** values along with the algorithms that are supported for each **SubFilter**.

Table 260 — SubFilter value algorithm support

SubFilter values	**adbe.pkcs7.detached, ETSI.CAdES.detached or ETSI.RFC3161**	**adbe.pkcs7.sha1**[c]	**adbe.x509.rsa_sha1**[a]
Message Digest	SHA1 (*PDF 1.3*) [d] SHA256 (PDF 1.6) SHA384 (PDF 1.7) SHA512 (PDF 1.7) RIPEMD160 (PDF *1.7*)	SHA1 (*PDF 1.3*)[b] SHA256 (PDF 1.6) SHA384 (PDF 1.7) SHA512 (PDF 1.7) RIPEMD160 (PDF 1.7)	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
ECDSA Algorithm Support (defined by *Internet RFC 5480*)	ANSI X9.62, Elliptic Curve Digital Signature Algorithm (ECDSA) *(PDF 2.0*)	No	No

[a] Despite the appearance of *sha1* in the name of this **SubFilter** value, supported encodings shall not be limited to the SHA-1 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, SHA-1 shall be used to digest the data that is being signed.

[c] The values *adbe.x509.rsa_sha1* and *adbe.pkcs7.sha1* have been deprecated with PDF 2.0.

[d] *SHA1* has been deprecated with PDF 2.0.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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. 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. PKCS #1 signatures are deprecated with PDF 2.0.

© ISO 2020 – All rights reserved

12.8.3.3 CMS (PKCS #7) signatures

12.8.3.3.1 General

When CMS signatures are used, the value of **Contents** shall be a DER-encoded CMS binary data object containing the signature.

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 CMS 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 CMS to the allocated space in the PDF file.

The CMS object shall conform to *Internet RFC 5652*. Different **SubFilter** values may be used and shall be registered in accordance with Annex E, "Extending PDF". **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 CMS SignedData field. No data shall be encapsulated in the CMS SignedData field.
- *adbe.pkcs7.sha1*: The SHA-1 digest of the document's byte range shall be encapsulated in the CMS SignedData field with ContentInfo of type Data. The digest of that SignedData shall be incorporated as the normal CMS digest. The value *adbe.pkcs7.sha1* for the **SubFilter** key has been deprecated with PDF 2.0. To support backward compatibility, PDF readers should process this value for this key, but PDF writers shall not use this value.

At minimum the CMS object shall include the signer's X.509 signing certificate. This certificate shall be used to verify the signature value in **Contents**.

If the signature handler has on-line access, it may place into the CMS object time stamping information and/or revocation information.

NOTE Since PDF 2.0 supports two additional dictionaries, i.e. the DSS and the DTS dictionaries, revocation information can be better placed in a DSS dictionary while time-stamping information can also be placed in a DTS dictionary, in addition to being placed in the CMS object.

In such a case, the CMS object should contain the following:

- Timestamp information as an unsigned attribute (*PDF 1.6*): The timestamp token shall conform to *Internet RFC 3161* as updated by *Internet RFC 5816*, and shall be computed and embedded into the CMS object as described in Appendix A of *Internet RFC 3161* as updated by *Internet RFC 5816*. The specific treatment of this timestamp tokens and its processing is left to the particular signature handlers to define.
- Revocation information as a 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 shall be obtained before the computation of the digital signature. This means that the software used by the signer should 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.
- The signature handler should capture the chain of certificates, including the signer certificate, along with the other certificates in the certificate chain, before signing. The signature handler should validate the certificate's chain before signing. However, if this is not possible, the DSS (see

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.8.4.3, "Document Security Store (DSS)") may be used for adding the information as an incremental update. 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 260 — SubFilter value algorithm support".

- One or more *Internet RFC 5755* 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.

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 of CMS-based signatures

For signatures with a **SubFilter** value other than *ETSI.CAdES.detached*, the following rules apply.

The adbe Revocation Information attribute object identifier is specified as follows using ASN.1 notation:

```
adbe-revocationInfoArchival OBJECT IDENTIFIER::=
                        {adbe(1.2.840.113583) acrobat(1) security(1) 8}
```

The value of the revocation information attribute may include any of the following data types:

- Certificate Revocation Lists (CRLs), described in *Internet RFC 5280*.

NOTE CRLs can be large and therefore require more pre-allocated space in the value of the Contents key.

- Online Certificate Status Protocol (OCSP) Responses, described in *Internet RFC 6960*. These are generally small and should be the data type included in the CMS 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.

The ASN.1 type of adbe's Revocation Information attribute value is RevocationInfoArchival defined as follows using ASN.1 notation:

```
RevocationInfoArchival::= SEQUENCE {
   crl          [0] EXPLICIT SEQUENCE of CRLs, OPTIONAL
   ocsp         [1] EXPLICIT SEQUENCE of OCSPResponse, OPTIONAL
   otherRevInfo [2] EXPLICIT SEQUENCE of OtherRevInfo, OPTIONAL
}
OtherRevInfo::= SEQUENCE {Type OBJECT IDENTIFIER
Value OCTET STRING
}
```

12.8.3.4 CAdES signatures as used in PDF

12.8.3.4.1 General

The Cryptographic Message Syntax (CMS)(see *Internet RFC 5652*) is used by CAdES signatures. The PDF signatures using the **SubFilter** value *ETSI.CAdES.detached* are referred to as *PAdES signatures* and they follow one of two CMS profiles created to be compatible with the corresponding *CAdES profiles* defined in ETSI EN 319 122. Combined with 12.8.4, "Long term validation of signatures" and 12.8.5, "Document timestamp (DTS) dictionary" as described in ETSI EN 319 142-1 (PAdES), compatibility with additional CAdES profiles can be achieved.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.8.3.4.2 Signature dictionary for PAdES signatures

When the **SubFilter** value is *ETSI.CAdES.detached*, the value of **Contents** shall be a DER-encoded CMS SignedData binary data object containing the signature. The signature dictionary shall follow the specification given in 12.7.5.5, "Signature fields" with the following additional restrictions/constraints:

- The **ByteRange** shall cover the entire PDF file, including the signature dictionary but excluding the **Contents** entry.
- The signature dictionary shall not contain a **Cert** entry.
- Either the time of signing may be indicated by the value of the **M** entry in the signature dictionary or the signing-time attribute may be used, but not both.

12.8.3.4.3 Attributes for PAdES signatures

The attributes in the SignedData object used as the value of the signature dictionary **Contents** key shall obey the following rules:

a) content-type: shall be present and shall always have the value "id-data".

b) Signature timestamp: A timestamp from a trusted timestamp server should be present as a unsigned attribute. The timestamp should be applied on the digital signature immediately after the signature is created so the timestamp specifies a time as close as possible to the time at which the document was signed. The rules from clause 5.3 in ETSI EN 319 122-1 shall apply.

c) content timestamp: may be present. If the content timestamp attribute is present, it shall be used in the same way as defined in clause 5.2.8 in ETSI EN 319 122-1.

d) exactly one single SignerInfo attribute shall be present.

e) message-digest: shall be present and shall be used as defined in CMS (*Internet RFC 5652*).

f) signing-certificate or signing-certificate-v2: shall be used as a signed attribute as described in the ESS signing-certificate attribute or as described in the ESS signing-certificate-v2 attribute as defined in clause 5.2.2 in ETSI EN 319 122-1. The details of the signing certificate attribute are defined in *Internet RFC 5035*.

g) signing-time: may be present. If present, it contains a UTC time. If the signing-time attribute is present, the time of signing shall not be indicated by the value of the M entry in the signature dictionary.

h) signer-location, as defined in clause 5.2.5 in ETSI EN 319 122-1, may be present. In such a case, the Location entry in the signature dictionary shall not be present.

i) these attributes shall not be used: counter-signature, content-reference, content-identifier, and content-hints.

j) signer-attributes-v2: may be used and shall follow the definition given in clause 5.2.6.1 of ETSI EN 319 122-1.

k) Unsigned signature attributes not explicitly noted here may be ignored.

12.8.3.4.4 Profiles of ETSI.CAdES.detached

The signatures using the **SubFilter** value *ETSI.CAdES.detached* may follow one of two profiles denoted as PAdES-E-BES (Basic Electronic Signature) and PAdES-E-EPES (Explicit Policy Electronic Signature). These are defined to be compatible with the corresponding profiles defined in ETSI EN 319 122-2 (i.e., CAdES-E-BES and CAdES-E-EPES, respectively). These two CMS profiles are also denoted as PAdES-E-BES (Basic Electronic Signature) and PAdES-E-PES (Explicit Policy Electronic Signature) in ETSI EN 319 142-2. If discrepancies exist between this document and those of ETSI, this document shall prevail.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The following attributes may be present within the CMS SignedData object depending on the profile employed:

For both the PAdES-E-BES and the PAdES-E-PES profiles:

- If a commitment-type-indication attribute is present, a Reason entry shall not be used
- If a commitment-type-indication attribute is not present, a Reason entry may be used

NOTE 2 The commitment-type-indication attribute contains an OID, whereas the entry Reason of the signature dictionary contains a text string which is language dependent.

For the PAdES-E-EPES profile,

- a signature-policy-identifier shall be present as a signed attribute. The rules from clause 5.2.9 in CAdES (ETSI EN 319 122-1) shall apply. The parameters of the signature policy shall take precedence over the seed values defined in 12.7.5.5, "Signature fields". Conforming signature handlers should enforce seed value constraints at signing time, if not overridden, and should enforce signature policies constraints at signing time when possible. During validation conforming signature handlers should not enforce seed values constraints, if present, but shall enforce signature policy constraints.

NOTE 3 It is important not to confuse the EPES defined attribute parameters with the "seed values" defined in clause 12.7.5.5, "Signature fields". While both bear similarities, seed values are workflow constraints for a given document, whereas signature policies represent general endorsement rules agreed upon by the signer and the verifier.

Further CAdES-T compatible capabilities may be provided after the time of the signature using one of the following methods:

- with the addition, by the signer, of a timestamp attribute within its signature when sufficient space has been prepared for it (see ETSI EN 319 122).
- with the addition, by the verifier, of a timestamp token that applies to the PDF document as a whole (see 12.8.5, "Document timestamp (DTS) dictionary").

In both cases, this demonstrates that the signer's signature was created before the UTC time contained in the timestamp token.

All signed attributes shall be checked for proper values by signature validation software.

12.8.3.4.5 Requirements for validation of PAdES signatures

For all profiles of PAdES signatures covered in this document (i.e., those with a **SubFilter** value of *ETSI.CAdES.detached*), when the user opens a signed document or requests verification of these signatures present in the PDF, a PDF processor, as the verifier, shall perform the following steps to verify them:

a) A signature handler shall compare the hash value of the signer's certificate, with the hash value given in the signing-certificate attribute or the signing-certificate-v2 attribute. If the hashes do not match, then the signature is considered invalid. The signature handler shall use the public key contained in the signer's certificate to verify that the document digest found in the signature is correctly signed. If this is not the case, then the signature is considered invalid.

b) The signature handler shall validate the path of certificates used to verify the binding between the subject distinguished name and subject public key as specified in *Internet RFC 5280* clause 6, using a set of validation parameters. The signature may be verified against a time other than the current

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

time if all validation information (e.g. certificates and revocation information) is known to have existed at that time (e.g. using DSS (see 12.8.4.3, "Document Security Store (DSS)"); using document timestamps, (see 12.8.5, "Document timestamp (DTS) dictionary"); or a signature timestamp is present in the signature as an unsigned attribute (see 12.8.3.4.3, "Attributes for PAdES signatures")). Otherwise, the local current time converted into the UTC shall be used.

NOTE The claimed signing time specified by the signature dictionary value with the key **M** is not a trusted indication of the signing time.

c) When the signer's signature is verified against a time indicated in a timestamp token present in a Document timestamp dictionary, then the signature handler shall also validate the path of certificates used to verify the timestamp unit's certificate, as specified in *Internet RFC 5280* clause 6, using a set of validation parameters which may be different from the previous one.

d) The revocation status of the certification path shall be checked as specified in 12.8.3.4.6, "Signature revocation checking model for PAdES signatures".

12.8.3.4.6 Signature revocation checking model for PAdES signatures

A signature handler shall use either, or both, of the following methods to check the revocation status of every certificate belonging to a certification path:

- Certificate Revocation Lists (CRLs) obtained from Certification Authorities (CAs) or CRL Issuers that are identified using a set of validation parameters. Using a CRL, the certificate in question shall be checked against a list of revoked certificates.

NOTE 1 In addition to the certificate issue date and the issuing entities, the CRL specifies revoked certificates and can also specify the reasons for revocation. Each CRL also contains a date at the latest for the next release.

- Online Certificate Status Protocol (OCSP) responses for obtaining the revocation status of a given certificate from trusted network servers that are identified using a set of validation parameters.

For the verification of a signer's signature at a time located before the expiry of the signer's certificate, two cases need to be considered:

1. the current UTC time, or

2. a UTC time in the past, when a timestamp token is present as an unsigned attribute of this signature or of the signature which covers this signature or of the document timestamp which covers this signature.

NOTE 2 For a verification of a signer's signature after the expiry of the signer's certificate, see 12.8.4, "Long term validation of signatures".

12.8.3.4.7 Revocation checking PAdES signatures at the current time

When the signer's signature is verified at the current UTC time, such verification shall be done before the expiry of the signer's certificate. The revocation of each certificate from the certification path of the signer's certificate shall be checked.

When CRLs are being used, a CRL is appropriate only if the current UTC time is included within the time period delimited by the thisUpdate and the nextUpdate fields of the CRL being used.

When OCSP responses are being used, an OCSP response is appropriate only if it has been captured before the expiry of the signer's certificate.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE Such checking is adequate for a data origin authentication service but not for a non-repudiation service, since after a successful validation done at one time, another validation done at a later time will provide a validation failure if one of the certificates from the certification path happens to be subsequently revoked.

12.8.3.4.8 Revocation checking PAdES signatures at a time in the past

When a timestamp token is already present in the CAdES signature as a signature timestamp attribute (it is an unsigned attribute), the signer's signature shall be verified at the UTC time in the past indicated in that token. The revocation status of each certificate from the certification path of the signer's certificate shall be checked at that UTC time, as well as the revocation status of each certificate from the certification path of the timestamping unit's certificate.

NOTE When there is no valid timestamp token present in the CAdES signature, revocation checking at a time in the past is addressed in 12.8.5, "Document timestamp (DTS) dictionary".

When CRLs are being used, a CRL is appropriate only if the two following conditions apply:

- the CRL has been captured before the expiry of the signer's certificate, and
- the time indicated in the nextUpdate field from the CRL is located after the UTC time in the past.

When OCSP responses are being used, an OCSP response is appropriate only if the two following conditions apply:

- the OCSP response has been captured before the expiry of the signer's certificate, and
- the time indicated in the producedAt field from the OCSP response is located after the UTC time in the past.

The following explanations only apply when the certificate of the timestamping unit that has produced the timestamp token has not yet expired at the time of the verification. The scenario when it has expired is addressed in 12.8.5.3, "Subsequent document timestamp dictionaries".

The reason for the revocation of a timestamping certificate and of every CA certificate belonging to the certification path shall be indicated either in the CRLs or in the OCSP responses that are being used.

The revocation status of each certificate of the certification path of the timestamping unit's certificate shall be checked at the current UTC time (i.e. not at a time in the past):

- If no certificate has been revoked, then the UTC time included in the timestamp token can be considered as reliable.
- If a certificate has been revoked for a reason which is either "keyCompromise" or "cACompromise" or if the revocation reason is missing, then the UTC time included in the timestamp token cannot be considered as reliable and the time indicated in the timestamp token cannot be used. The benefits of the timestamp token are then lost and revocation checking should thus be done at the current time (see 12.8.3.4.6, "Signature revocation checking model for PAdES signatures").
- If a certificate has been revoked for any other reason than "keyCompromise" or "cACompromise", then the UTC time included in the timestamp token can be considered as reliable (and is thus usable) only if the time of the revocation of that timestamping unit's certificate occurred after the UTC time included in the timestamp token. Otherwise, revocation checking should be done at the current time (see 12.8.3.4.6, "Signature revocation checking model for PAdES signatures").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

12.8.4 Long term validation of signatures

12.8.4.1 General

Long term validation (LTV) of signatures is achieved by using two types of dictionaries:

- document security store (DSS) dictionaries, and
- document timestamp dictionaries (DTS) (see 12.8.5, "Document timestamp (DTS) dictionary").

12.8.4.2 Introduction to the document security store (DSS)

A PDF signature may not be successfully verified unless its collateral validation components are preserved, e.g., certificates, CRLs, timestamp tokens, revocation lists, and OCSP responses. To facilitate long term signature validation, PDF supports the ability to collect validation information to verify a signature at a later time if it has been verified once as being valid. Some of this information, i.e. certificates, CRLs and OCSP responses, when not already present in the signature, shall be stored in a document security store (DSS), see 12.8.4.3, "Document Security Store (DSS)". This will provide the information needed to verify a signature as this was done when that signature was first verified.

Without an authoritative timestamp token, a signature handler is not able to verify a signature at a date in the past to get the same validation result.

To allow this verification, one of the following is required:

- a signature timestamp attribute (which contains a timestamp token), or
- a timestamp token that applies to the PDF document as a whole (see 12.8.5, "Document timestamp (DTS) dictionary").

When that timestamp token is already present in the CAdES signature (see 12.8.3.4.7, "Revocation checking PAdES signatures at the current time") and if that timestamp token is valid, then the UTC time included in that timestamp token shall be used as the time reference to check the revocation status of the signer's certificate and of all the intermediate CA certificates, up to a trusted root. Afterwards, the DSS dictionary shall be used to collect the certificates, CRLs and OCSP responses that are relevant to validate that signature.

When there is no valid timestamp token present in the CAdES signature, or when such a timestamp token is present but is considered as invalid, the DSS dictionary shall be used to collect the certificates, CRLs and OCSP responses that are relevant to validate that signature and afterwards a timestamp token that applies to the PDF document as a whole shall be used and placed in a Document timestamp dictionary (DTS) as noted in 12.8.5, "Document timestamp (DTS) dictionary").

A timestamp token is itself signed. The case where the certificate of the timestamping unit which has issued that timestamp token has expired is addressed in 12.8.5.3, "Subsequent document timestamp dictionaries".

A timestamp token is itself signed, and so it is possible for the timestamp token's own validation data also to be preserved. As with signatures from the signer(s) of the document, a DSS dictionary may also include signature validation data relating to the timestamp token contained in a signature-timestamp attribute or to document timestamps, see 12.8.5, "Document timestamp (DTS) dictionary").

The DSS supports the ability to add this critical information *after* the signed document has been created and *before* this information is no longer available. At the time of signing, all of the collateral

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

validation components may not be available for various reasons including the inability to connect to remote servers that provide them (e.g. the user is offline) or the inability of the signer to bear the financial or time costs associated with obtaining these components. Also, in many workflows it is the recipient of a signed PDF document who is interested in a long term validation of the signatures and not the signer of the document.

The relevant validation data for each signature may be identified from the security store using a VRI dictionary (see 12.8.4.4, "Validation-related information (VRI)") for optimisation or to remove ambiguity of the validation data used to validate a specific signature.

NOTE A benefit of using a DSS dictionary is that those components which are common to several signatures (e.g. certificates and revocation lists) can be stored in this dictionary once and referenced where ever they are needed. This can greatly reduce the size of a PDF document that contains several signatures.

12.8.4.3 Document Security Store (DSS)

The document security store (DSS), when present, shall be a dictionary that shall be the value of a **DSS** key in the document catalog dictionary (see 7.7.2, "Document catalog dictionary"). This dictionary may contain:

- an array of all certificates used for the signatures, including timestamp signatures, that occur in the document. It shall also hold all the auxiliary certificates required to validate the certificates participating in certificate chain validations.
- an array of all Certificate Revocation Lists (CRL) (see *Internet RFC 5280*) used for some of the signatures, and
- an array of all Certificate Status Protocol (OCSP) responses (see *Internet RFC 6960*) used for some of the signatures.
- a **VRI** key whose value shall be a dictionary containing one **VRI** dictionary (validation-related information) for each signature represented in CMS format.

Any **VRI** dictionaries, if present, shall be located in document incremental update sections. If the signature dictionary to which a **VRI** dictionary applies is itself in an incremental update section, the **DSS/VRI** update shall be done later than the signature update. The inclusion of VRI dictionary entries is optional. All validation material referenced in VRI entries is included in DSS entries too.

“Table 261 — Entries in the document security store (DSS) dictionary” shows the entries in the DSS dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 261 — Entries in the document security store (DSS) dictionary

Key	Type	Value
Type	name	*(Optional)* If present, shall be *DSS* for a document security store dictionary.
VRI	dictionary	*(Optional)* This dictionary contains Signature VRI dictionaries (see 12.8.4.4, "Validation-related information (VRI)"). The key of each entry in this dictionary is the base-16-encoded (uppercase) SHA-1 digest of the signature to which it applies[a] and the value is the Signature VRI dictionary which contains the validation-related information for that signature.

© ISO 2020 – All rights reserved

Key	Type	Value
Certs	array	*(Optional)* An array of indirect reference to streams, each containing one DER-encoded X.509 certificate (see *Internet RFC 5280*). This array contains certificates that may be used in the validation of any signatures in the document.
OCSPs	array	*(Optional)* An array of indirect references to streams, each containing a DER-encoded Online Certificate Status Protocol (OCSP) response (see *Internet RFC 6960*). This array contains **OCSPs** that may be used in the validation of the signatures in the document.
CRLs	array	*(Optional)* An array of indirect references to streams, each containing a DER-encoded Certificate Revocation List (CRL) (see *Internet RFC 5280*). This array contains **CRLs** that may be used in the validation of the signatures in the document.
[a] For a document signature or document timestamp signatures, the bytes that are hashed are those of the complete hexadecimal string, including zero padding, in the **Contents** entry of the associated signature dictionary, containing the signature's DER-encoded binary data object (e.g. CMS or CAdES objects).		

For the signatures of **CRLs** and **OCSP** responses, the bytes that are hashed are the respective signature object represented as a BER-encoded OCTET STRING encoded with primitive encoding.

12.8.4.4 Validation-related information (VRI)

A signature VRI dictionary shall contain validation-related information (VRI) for one signature in the document that a given signature handler or PDF processor has used to successfully validate the given signature. A signature VRI dictionary shall reference:

- a selection of certificates (Cert) from the certificates list (Certs) in the DSS dictionary applicable to this signature;
- a selection of **CRLs** from the **CRL** list in the DSS dictionary applicable to this signature, if any;
- a selection of **OCSP responses** from the **OCSP** response list in the DSS dictionary applicable to this signature.

If any of the **Cert, CRL** or **OCSP** arrays is empty that entry in the dictionary shall be omitted.

A VRI dictionary may also contain the time at which the data placed in the VRI dictionary was created or/and a timestamp token which contains the UTC time at which the VRI dictionary was created.

A signature VRI dictionary shall not be used to record the information used in an unsuccessful validation attempt. “Table 262 — Entries in the signature validation-related information (VRI) dictionary” shows the entries in the VRI dictionary.

Table 262 — Entries in the signature validation-related information (VRI) dictionary

Key	Type	Value
Type	name	*(Optional)* If present, shall be *VRI* for a validation-related information dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Cert	array	*(Optional)* An array of (indirect references to) streams, each containing one DER-encoded X.509 certificate (see *Internet RFC 5280*). This array contains certificates that were used in the validation of this signature.
CRL	array	*(Required, if a CRL is present)* An array of indirect references to streams that are all **CRLs** used to determine the validity of the certificates in the chains related to this signature. Each stream shall reference a **CRL** that is an entry in the **CRLs** array in the DSS dictionary.
OCSP	array	*(Required, if an OCSP is present)* An array of indirect references to streams that are all **OCSPs** used to determine the validity of the certificates in the chains related to this signature. Each stream shall reference an OCSP that is an entry in the **OCSPs** array in the DSS dictionary.
TU	date	*(Optional)* The date/time at which this signature VRI dictionary was created. **TU** shall be a date string as defined in 7.9.4, "Dates".
TS	stream	*(Optional)* A stream containing the DER-encoded timestamp (see *Internet RFC 3161* as updated by *Internet RFC 5816*) that contains the date/time at which this signature **VRI** dictionary was created. NOTE 1 The date/time contained in the timestamp token can be used for audit purposes. NOTE 2 The hash value to be contained in the timestamp token is left undefined. For PKCS #7 signatures the datum that is hashed and included in the messageImprint field of the DER encoded timestamp stored in the TS entry (see *Internet RFC 3161* as updated by *Internet RFC 5816*) is the encryptedDigest field in the signature's PKC S#7 object (as defined in *Internet RFC 2315*).

EXAMPLE 1 DSS dictionary (and associated objects)

```
100 0 obj
      <<
         /Type /Catalog
         /DSS 101 0 R
         … other stuff here …
      >>
endobj

101 0 obj
      <<
         /VRI 102 0 R
         /OCSPs [103 0 R]
         /CRLs [104 0 R]
         /Certs [105 0 R 106 0 R]
      >>
endobj

102 0 obj
      <<
         /4B783B9A6D0D69E4E881BFDF080835E896735416  <</OCSP [103 0 R] /CRL [104 0 R]>>
      >>
endobj

103 0 obj
      <<
         /Length …                    %whatever the length of the stream is
      >>
stream
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
… OCSP data goes here …
endstream

104 0 obj
      <<
         /Length …                    %whatever the length of the stream is
      >>
stream
… %CRL data goes here …
endstream

105 0 obj
      <<
         /Length …                    %whatever the length of the stream is
      >>
stream
… Certificate data goes here …
endstream

106 0 obj
      <<
         /Length …                    %whatever the length of the stream is
      >>
stream
… Certificate data goes here …
endstream
```

EXAMPLE 2 *DSS* sample with Two Signatures

```
101 0 obj
      <<
         /VRI 102 0 R
         /OCSPs [103 0 R 107 0 R]
         /CRLs [104 0 R]
         /Certs [105 0 R 106 0 R]
      >>
endobj

102 0 obj
      <<
         /4B783B9A6D0D69E4E881BFDF080835E896735416     <</OCSP [103 0 R] /CRL [104 0 R]>>
         /123456789ABCDEF987654321FEDCBA1234567890     <</OCSP [107 0 R]>>

      >>

107 0 obj
      <<
         /Length …                    %whatever the length of the stream is
      >>
stream
… OCSP data goes here …
endstream
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.8.4.5 Usage of the DSS VRI

The validation data contained in the DSS dictionary and those embedded in the signature itself may be used by another party later relying on the signature. The applicability of the validation data in the signature **VRI** dictionary is subject to external conditions such as a set of validation parameters. In the presence of *DSS* in a PDF document the preferred order of the search for validation data should be as follows:

© ISO 2020 – All rights reserved

a) Validation data referenced in the **VRI** dictionary

b) Validation data in *DSS*

c) Validation data embedded in the signature.

12.8.5 Document timestamp (DTS) dictionary

12.8.5.1 General

A document timestamp dictionary establishes the exact contents of the complete PDF file at the time indicated in the timestamp token.

12.8.5.2 Initial document timestamp dictionary

The timestamp token shall be an *Internet RFC 3161* TimeStampToken, as updated by *Internet RFC 5816*, obtained from a trusted timestamp authority. A Document Timestamp dictionary is a standard signature dictionary as described in "Table 255 — Entries in a signature dictionary".

The **ByteRange** key shall cover the entire PDF file, including the signature dictionary but excluding the Contents value. The hash value computed over that byte range shall be sent to a timestamping authority and the TimeStampToken which is received shall be placed in the Contents key.

The existence of one or more document timestamps shall be determined by examining signature fields (see 12.7.5.5, "Signature fields"). A document level timestamp is treated as a digital signature in most respects. It normally would not have any visual appearance within the document content.

NOTE When a document level timestamp is validated, using the same procedures as for other signatures using its own set of validation parameters, one can determine that the contents of the document have not been changed since a given past date.

12.8.5.3 Subsequent document timestamp dictionaries

A timestamp token present either in a signature timestamp attribute or in a document timestamp dictionary may expire due to the expiry of its certificate or the cryptographic strength of some of their algorithms may not be resistant any longer to some new cryptographic attack.

It is thus necessary to apply a new timestamp token before the expiry of the certificate and/or before a cryptographic attack may succeed but it is also necessary to be able to demonstrate that the certificates related to the certification path of the timestamp authority certificate were not revoked for a reason which is either "keyCompromise" or "cACompromise" at the time when the new timestamp token has been applied to the document.

The certificates, CRLs or OCSP responses used to demonstrate that the certificates related to the certification path of the previous timestamp token was not revoked for a reason which is either "keyCompromise" or "cACompromise" just before the time the new timestamp token has been requested shall be included into the DSS dictionary.

Then the new timestamp token shall be placed into a new document timestamp dictionary which will protect the whole structure.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

The process may be repeated several times as long as there is a need to reverify the signatures included in the document. “Figure 86 — Illustration of a PDF document with repeated LTV” illustrates a PDF document with repeated LTV.

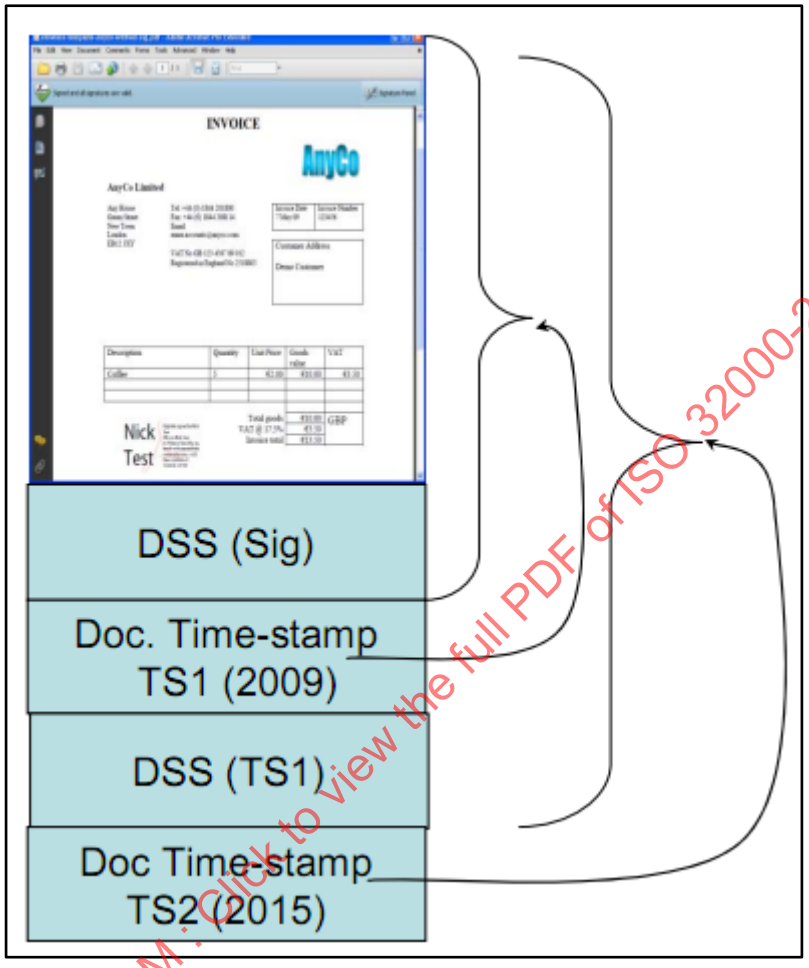


Figure 86 — Illustration of a PDF document with repeated LTV

When evaluating the **DocMDP** restrictions (see 12.8.2.2, "DocMDP") the presence of a document timestamp and/or **DSS** information shall be ignored.

EXAMPLE Document timestamp

```
1 0 obj
      <<
         /Type /Catalog /Pages 2 0 R
         /AcroForm 5 0 R
      >>
endobj

2 0 obj
      <<
         /Kids [3 0 R]
         /Count 1
         /Type /Pages
      >>
endobj

3 0 obj
      <<
         /Type /Page
         /Parent 2 0 R
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            /MediaBox [0 0 612 792]
            /Annots 4 0 R
         … other keys go here …
         >>
endobj

4 0 obj
         <<
            /Type /Annot /Subtype /Widget
            /Rect [0 0 0 0]
            /F 4 /P 3 0 R
            /FT /Sig /T (Sig)
            /V 6 0 R
         >>
endobj

5 0 obj
         <<
            /Fields [4 0 R]
            /SigFlags 3
         >>
endobj

6 0 obj
         <<
            /Type /DocTimeStamp
            /Filter /Adobe.PPKLite
            /SubFilter /ETSI.RFC3161
            /Contents <00000>                          %values go here inside of <>
            /ByteRange [0 0 0 0]                       %values go here inside of []
         >>
endobj
```

12.8.6 Permissions

The **Perms** entry in the document catalog dictionary (see "Table 29 — Entries in the catalog dictionary") shall specify a permissions dictionary (*PDF 1.5*). Each entry in this dictionary (see "Table 263 — Entries in a permissions dictionary" 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 — Standard security handler user access permissions") 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 263 — Entries in a permissions dictionary

Key	Type	Value
DocMDP	dictionary	*(Optional)* An indirect reference to a signature dictionary (see "Table 255 — Entries in a signature dictionary"). This dictionary shall contain a **Reference** entry that shall be a signature reference dictionary (see "Table 255 — Entries in a signature dictionary") that has a **DocMDP** transform method (see 12.8.2.2, "DocMDP") and corresponding transform parameters. If this entry is present, PDF processors shall enforce the permissions specified by the **P** entry 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; deprecated in PDF 2.0)* A signature dictionary that may be used to specify and validate additional capabilities (usage rights) granted for this document; that is, the enabling of features of a PDF processor that are not available by default. 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 and recognized by the PDF processor, then the PDF processor shall allow the specified permissions for the document, in addition to the default permissions. NOTE For example, a PDF processor may not permit saving documents by default. The signature can be used to validate that the additional permissions placed within the PDF document have been granted by the authority or agent that did the signing.

12.8.7 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.

PDF provides a mechanism 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, PDF writers should also create a legal attestation dictionary, whose entries are shown in "Table 264 — Entries in a legal attestation dictionary". This dictionary shall be the value of the **Legal** entry in the document catalog dictionary (see "Table 29 — Entries in the catalog dictionary"). 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 264 — Entries in a legal attestation dictionary

Key	Type	Value
JavaScriptActions	integer	*(Optional)* The number of ECMAScript actions found in the document (see 12.6.4.17, "ECMAScript actions").
LaunchActions	integer	*(Optional)* The number of launch actions found in the document (see 12.6.4.6, "Launch actions").
URIActions	integer	*(Optional)* The number of URI actions found in the document (see 12.6.4.8, "URI actions").
MovieActions	integer	*(Optional; deprecated in PDF 2.0)* The number of movie actions found in the document (see 12.6.4.10, "Movie actions").
SoundActions	integer	*(Optional; deprecated in PDF 2.0)* The number of sound actions found in the document (see 12.6.4.9, "Sound actions").
HideAnnotationActions	integer	*(Optional)* The number of hide actions found in the document (see 12.6.4.11, "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; deprecated in PDF 2.0)* 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 57 — Entries in a graphics state parameter dictionary").
DevDepGS_HT	integer	*(Optional)* The number of references to the graphics state parameter **HT** found in the document (see "Table 57 — Entries in a graphics state parameter dictionary").
DevDepGS_TR	integer	*(Optional)* The number of references to the graphics state parameter **TR** found in the document (see "Table 57 — Entries in a graphics state parameter dictionary").
DevDepGS_UCR	integer	*(Optional)* The number of references to the graphics state parameter **UCR** found in the document (see "Table 57 — Entries in a graphics state parameter dictionary").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
DevDepGS_BG	integer	*(Optional)* The number of references to the graphics state parameter **BG** found in the document (see "Table 57 — Entries in a graphics state parameter dictionary").
DevDepGS_FL	integer	*(Optional)* The number of references to the graphics state parameter **FL** found in the document (see "Table 57 — Entries in a graphics state parameter dictionary").
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

12.9.1 General

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 interactive PDF processors 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 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

$$\sum_{i=0}^{n-2} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

Beginning with PDF 1.6, such information may be stored in a measure dictionary (see "Table 266 — Entries in a measure dictionary"). 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 31 — Entries in a page object") shall specify an array of viewport dictionaries, whose entries shall be as shown in "Table 265 — Entries in a viewport dictionary". Viewports allow different

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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.

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 265 — 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 normalised 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 267 — Additional entries in a rectilinear measure dictionary") of the measurement dictionary specified by **Measure**.
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 266 — Entries in a measure dictionary") that specifies the scale and units that shall apply to measurements taken on the contents within the viewport.
PtData	dictionary	*(Optional; PDF 2.0)* A point data dictionary (see "Table 272 — Entries in a point data dictionary") that shall specify the extended geospatial data that applies to this viewport.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

A measure dictionary shall specify an alternative 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 266 — Entries in a measure dictionary" 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. *RL* 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). PDF 2.0 defines a *geospatial*

© ISO 2020 – All rights reserved

coordinate system specified by the value *GEO* for the **Subtype** entry. Other subtypes may be used, providing the flexibility to measure using other types of coordinate systems.

When the value of the **Subtype** entry is *GEO*, the dictionary shall define the relationship between points or regions in the two dimensional PDF object space and points or regions with respect to an underlying model of the earth (or, potentially, other ellipsoid objects).

Table 266 — 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. *RL* for a rectilinear coordinate system *GEO* *(PDF 2.0)* for a geospatial coordinate system Default value: *RL*

"Table 267 — Additional entries in a rectilinear measure dictionary" 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 268 — Entries in a number format dictionary"). 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.

When the subtype of a measurement dictionary is *GEO*, additional entries are defined. "Table 269 — Additional entries in a geospatial measure dictionary" lists and describes these additional entries in a geospatial measure dictionary.

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 can 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).

NOTE 3 A number format dictionary specifying feet needs to precede one specifying inches.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 267 — 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 recognised 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 265 — Entries in a viewport dictionary").
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 265 — Entries in a viewport dictionary"). Default value: the first coordinate pair (lower-left corner) of the rectangle specified by the viewport's **BBox** entry.
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 268 — 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 recognised 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 267 — Additional entries in a rectilinear measure dictionary") 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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*.
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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.9.2 Algorithm: Use of a number format array to create a formatted text string

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:

a) The entry in the rectilinear measure dictionary (see "Table 267 — Additional entries in a rectilinear measure dictionary") 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.

© ISO 2020 – All rights reserved

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 non-zero 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 non-zero 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 non-zero 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 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 Geospatial features

12.10.1 General

PDF is a common delivery mechanism for map and satellite imagery data. In PDF 2.0, a geospatial coordinate system is introduced ("Table 266 — Entries in a measure dictionary") along with a number of PDF constructs, as explained in this clause, to support geospatially registered content.

12.10.2 Geospatial measure dictionary

When the subtype of a measurement dictionary ("Table 266 — Entries in a measure dictionary") is *GEO*, additional entries are defined through a geospatial measure dictionary.

A geospatial measure dictionary, "Table 269 — Additional entries in a geospatial measure dictionary", contains a description of the earth-based coordinate system associated with the PDF object, and corresponding arrays of points in that coordinate system and the local object coordinate system. It may

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

contain a bounding polygon (the **Bounds** entry), which defines the region of the PDF object for which the geographic associations are valid. It may also contain a choice of default units (the **PDU** entry) for user displays of positions, distances and areas. An optional display coordinate system (the **DCS** entry) allows a document to be authored to display values in a coordinate system other than that associated with the source data. For example, a map may be created in a state planar coordinate system based on a 1927 datum, but it is possible to display its latitude and longitude values in the WGS84 datum corresponding to values reported by a GPS device.

The entries of a geospatial measure dictionary are shown in "Table 269 — Additional entries in a geospatial measure dictionary".

Table 269 — Additional entries in a geospatial measure dictionary

Key	Type	Description
Bounds	array	*(Optional; PDF 2.0)* An array of numbers that shall be taken pairwise to define a series of points that describes the bounds of an area for which geospatial transformations are valid. For maps, this bounding polygon is known as a neatline. These numbers are expressed relative to a unit square that describes the **BBox** associated with a Viewport or form XObject, or the bounds of an image XObject. If not present, the default values shall define a rectangle describing the full unit square, with values of [0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0]. NOTE 1 The polygon description need not be explicitly closed by repeating the first point values as a final point.
GCS	dictionary	*(Required; PDF 2.0)* A geographic or projected coordinate system dictionary. See "Table 270 — Entries in a geographic coordinate system dictionary" and "Table 271 — Entries in a projected coordinate system dictionary".
DCS	dictionary	*(Optional; PDF 2.0)* A projected or geographic coordinate system that shall be used for the display of position values, such as latitude and longitude. See "Table 270 — Entries in a geographic coordinate system dictionary" and "Table 271 — Entries in a projected coordinate system dictionary". Formatting the displayed representation of these values is controlled by the interactive PDF processor.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Description
PDU	array	*(Optional; PDF 2.0) Preferred Display Units*. An array of three names that identify in order a linear display unit, an area display unit, and an angular display unit. The following are valid linear display units: *M* a metre *KM* a kilometre *FT* an international foot *USFT* a U.S. Survey foot *MI* an international mile *NM* an international nautical mile The following are valid area display units: *SQM* a square metre *HA* a hectare (10,000 square metres) *SQKM* a square kilometre *SQFT* a square foot (US Survey) *A* an acre *SQMI* a square mile (international) The following are valid angular display units: *DEG* a degree *GRD* a grad (1/400 of a circle, or 0.9 degrees)
GPTS	array	*(Required; PDF 2.0)* An array of numbers that shall be taken pairwise, defining points in geographic space as degrees of latitude and longitude, respectively when defining a geographic coordinate system. These values shall be based on the geographic coordinate system described in the **GCS** dictionary. When defining a projected coordinate system, this array contains values in a planar projected coordinate space as eastings and northings. For **Geospatial3D**, when Geospatial feature information is present (requirement type **Geospatial3D**) in a 3D annotation, the **GPTS** array is required to hold 3D point coordinates as triples rather than pairwise where the third value of each tripe is an elevation value. NOTE 2 Any projected coordinate system includes an underlying geographic coordinate system.
LPTS	array	*(Optional; PDF 2.0)* An array of numbers that shall be taken pairwise to define points in a 2D unit square. The unit square is mapped to the rectangular bounds of the Viewport, image XObject, or forms XObject that contains the measure dictionary. This array shall contain the same number of number pairs as the **GPTS** array; each number pair is the unit square object position corresponding to the geospatial position in the **GPTS** array. For **Geospatial3D**, when Geospatial feature information is present in a 3D annotation (requirement type **Geospatial3D**), the **LPTS** array is required to hold 3D point coordinates as triples corresponding to the **GPTS** array in the 3D annotation view world coordinate space.
PCSM	array	*(Optional; PDF 2.0)* A 12-element transformation matrix of real numbers, defining the transformation from XObject position coordinates to projected coordinate system. If **GCS** is a geographic coordinate system dictionary then **PCSM** should be ignored and **GTPS** used instead. If **PCSM** is present, it has priority over **GPTS**, and **GPTS** values may be ignored. This priority provides backward compatibility. NOTE 3 PCSM is an acronym for "Projected Coordinate System Matrix".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.10.3 Geographic coordinate system dictionary

A geographic coordinate system (GEOGCS) specifies an ellipsoidal object in geographic coordinates: angular units of latitude and longitude. The geographic coordinate system shall be described in either or both of two well-established standards: as a numeric EPSG reference code, or as a Well Known Text (WKT) string, which contains a description of algorithms and parameters needed for transformations.

"Table 270 — Entries in a geographic coordinate system dictionary" lists the entries in a geographic coordinate system dictionary. A geographic coordinate system dictionary may be a value of the **GCS** or the **DCS** entry of a geospatial measure dictionary. (See "Table 269 — Additional entries in a geospatial measure dictionary".)

Table 270 — Entries in a geographic coordinate system dictionary

Key	Type	Description
Type	name	*(Required; PDF 2.0)* The type of PDF object that this dictionary describes. If present, shall be *GEOGCS* for a geographic coordinate system dictionary.
EPSG	integer	*(Optional; PDF 2.0)* An **EPSG** reference code specifying the geographic coordinate system. Shall not be present if **WKT** is present.
WKT	ASCII string	*(Optional; PDF 2.0)* A string of Well Known Text describing the geographic coordinate system. Shall not be present if **EPSG** is present.

Either an **EPSG** code or a **WKT** string shall be present in a geographic coordinate system dictionary.

The **EPSG** reference codes are described in a database available through http://www.epsg.org as administered by the International Association of Oil and Gas Producers (OGP). The **WKT** (Well Known Text) format is specified in ISO 19162.

12.10.4 Projected coordinate system dictionary

A projected coordinate system (PROJCS), which includes an embedded *GEOGCS*, specifies the algorithms and associated parameters used to transform points between geographic coordinates and a two-dimensional (projected) coordinate system. Any transformation between a three-dimensional curved geographic coordinate system and a two-dimensional coordinate system introduces distortions. For small areas, this distortion may be small enough to allow direct mapping between geographic coordinates and PDF object coordinates without requiring the use of a projected coordinate system.

The projected coordinate system shall be described in either or both of two well-established standards: as a numeric **EPSG** reference code, or as a Well KnownText (WKT) string, which contains a description of algorithms and parameters needed for transformations.

"Table 271 — Entries in a projected coordinate system dictionary" lists the entries in a projected coordinate system dictionary. A projected coordinate system dictionary may be a value of the **GCS** or the **DCS** entry of a geospatial measure dictionary, "Table 269 — Additional entries in a geospatial measure dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 271 — Entries in a projected coordinate system dictionary

Key	Type	Description
Type	name	(*Required; PDF 2.0*) The type of PDF object that this dictionary describes; shall be *PROJCS* for a projected coordinate system
EPSG	integer	(*Optional; PDF 2.0*) An **EPSG** reference code specifying the projected coordinate system.
WKT	ASCII string	(*Optional; PDF 2.0*) A string of Well Known Text describing the projected coordinate system.

Either an **EPSG** code or a **WKT** string shall be required in the projected coordinate system dictionary.

EXAMPLE 1 A **WKT** describing a geographic coordinate system

An example of **WKT** description of a geographic coordinate system, formatted for readability. The **EPSG** code equivalent to the GCS_North_American_1983 geographic coordinate system is 4269.

```
GEOGCS["GCS_North_American_1983",
      DATUM[
         "D_North_American_1983",
         SPHEROID["GRS_1980",6378137.0,298.257222101]
      ],
      PRIMEM["Greenwich",0.0],
      UNIT["Degree",0.0174532925199433]
]
```

EXAMPLE 2 A **WKT** describing a projected coordinate system

An example of **WKT** description of a projected coordinate system, formatted for readability. The **EPSG** code equivalent to the North_American_Albers_Equal_Area_Conic projected coordinate system is 102008.

```
PROJCS["North_America_Albers_Equal_Area_Conic",
      GEOGCS["GCS_North_American_1983",
         DATUM["D_North_American_1983",
            SPHEROID["GRS_1980",6378137.0, 298.257222101]
         ],
         PRIMEM["Greenwich",0.0],
         UNIT["Degree",0.0174532925199433]
      ],
      PROJECTION["Albers"],
         PARAMETER["False_Easting",0.0],
         PARAMETER["False_Northing",0.0],
         PARAMETER["Central_Meridian",-96.0],
         PARAMETER["Standard_Parallel_1",20.0],
         PARAMETER["Standard_Parallel_2",60.0]
         PARAMETER["Latitude_Of_Origin",40.0],
         UNIT["Meter",1.0]
      ]
```

12.10.5 Point data dictionary

Any 2D object (Viewport, image XObject, or form XObject) that contains a measure dictionary ("Table 266 — Entries in a measure dictionary") of subtype *GEO* can optionally include a **PtData** entry. The value of a **PtData** entry is a point data dictionary or an array of point data dictionaries of extended data associated with points in the 2D space. "Table 272 — Entries in a point data dictionary" lists the entries of a point data dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 272 — Entries in a point data dictionary

Key	Type	Description
Type	name	*(Required; PDF 2.0)* The type of PDF object that this dictionary describes; shall be *PtData* for a point data dictionary.
Subtype	name	(*Required; PDF 2.0*) Shall be *Cloud*.
Names	array	*(Required; PDF 2.0)* An array of names that identify the internal data elements of the individual point arrays in the **XPTS** array. There are three predefined names: *LAT* latitude in degrees. The **XPTS** value is a number type. *LON* longitude in degrees. The **XPTS** value is a number type. *ALT* altitude in metres. The **XPTS** value is a number type. NOTE These names are, in effect, column headers for the array of **XPTS** values.
XPTS	array	*(Required; PDF 2.0)* An array of arrays of values. The number of members in each interior array shall correspond to the size of the **Names** array; each member in the interior arrays is of a type defined by the corresponding name in the **Names** array. The **XPTS** array is a collection of tuples without any guaranteed ordering or relationship from point to point.

The names *LAT*, *LON*, and *ALT* are predefined, and shall be used to associate altitude information with latitude and longitude positions.

12.11 Document requirements

12.11.1 General

A PDF processor that supports document requirements shall evaluate them before execution of any ECMAScripts.

The **Requirements** entry in the document catalog (see 7.7.2, "Document catalog dictionary") shall be an array of requirement dictionaries, whose entries are shown in "Table 273 — Entries common to all requirement dictionaries".

Table 273 — 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. See "Table 276 — Entries in a requirement handler dictionary" for valid values.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Description
V	name or dictionary	*(Optional; PDF 2.0)* The minimum version level of support needed to satisfy the requirement. See 12.11.4, "Requirement versions". If this entry is absent, determining if the requirement is satisfied shall be done without regard to version number. Unless otherwise mentioned in the entries in "Table 276 — Entries in a requirement handler dictionary", the value shall represent the PDF version.
RH	dictionary or array	*(Optional)* An alternative requirement handler dictionary or an array of such dictionaries. Each dictionary identifies a requirement handler that shall be disabled (not invoked) if the interactive PDF processor can check the requirement specified in the **S** entry (whether or not it can satisfy that requirement). See 12.11.5, "Requirement handlers". Default value: an empty array.
Penalty	integer	*(Optional; PDF 2.0)* An integer value that shall be between 0 and 100 (inclusive) that represents the penalty value to be applied when this requirement cannot be met by a PDF processor. Default value is *100*.

There are two additional keys that may appear in a requirements dictionary that are specific to certain types of requirements (i.e., value of the **S** key). These are described in "Table 274 — Entries for specific types of requirements".

Table 274 — Entries for specific types of requirements

Key	Type	Description
Encrypt	dictionary	*(Required, if the **S** key has the value* Encryption*; PDF 2.0)* An encryption dictionary ("Table 20 — Entries common to all encryption dictionaries") that defines all of the relevant aspects of the encryption method needed to process the document.
DigSig	dictionary	*(Optional, but only used when the **S** key has the value of* DigSig**,** DigSigValidation or DigSigMDP*; PDF 2.0)* A signature dictionary ("Table 255 — Entries in a signature dictionary") that defines all of the relevant aspects that are needed in order to process the digital signature requirements.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

12.11.2 Requirement types

The **S** entry in a requirement dictionary identifies ("Table 273 — Entries common to all requirement dictionaries") a feature of the PDF language or a capability that may be present in a PDF processor. Such entries enable the document to identify feature(s) of PDF beyond those commonly expected, such as 2D graphics rendering, and are required for correct handling in accordance with this document. In addition, although not required for viewing, a document may also use requirement values that stipulate required features of interactive PDF processors such as the ability to interact with or modify the document.

"Table 275 — Requirement types" lists requirement types that have been defined through PDF 2.0.

© ISO 2020 – All rights reserved

Table 275 — Requirement types

Type	Description
OCInteract	Requires an interactive PDF processor to be able to display the list of optional content groups (OCGs) in the **Order** array as described in "Table 99 — Entries in an optional content configuration dictionary". In addition, requires that an interactive PDF processor support the **SetOCGState** action (see 12.6.4.13, "Set-OCG-state actions"). Additional information about OCGs can be found in 8.11.2, "Optional content groups" and 8.11.4.4, "Usage and usage application dictionaries".
OCAutoStates	Requires an interactive PDF processor to implement the various **Usage** values that can be present as the value of the **AS** key in an OCD as described in "Table 99 — Entries in an optional content configuration dictionary" and 8.11.4.4, "Usage and usage application dictionaries".
AcroFormInteract	Requires support for user interaction with forms (see 12.7, "Forms") defined as interactive form dictionaries including updating field appearances when values change. In addition, support for Trigger Actions (12.6.3, "Trigger events") is required. NOTE 1 This requirement does not cover presentation of a form's static appearance. That presentation uses annotation appearances (12.5.5, "Appearance streams"), which all PDF processors are assumed to support.
Navigation	Requires support for the presentation and handling of basic navigational elements including link annotations (12.5.6.5, "Link annotations") and outlines (12.3.3, "Document outline"). In addition, support shall be provided for **GoTo**, **GoToR** and **URI** actions (12.6.4, "Action types") in any of these elements or as a document, page or annotation trigger events (12.6.3, "Trigger events").
Markup	Requires support for the creation, modification and deletion of markup annotations (12.5.6.2, "Markup annotations") including text annotations. In addition, any time the visual appearance of the annotation changes, the appearance stream shall be updated.
3DMarkup	Requires support for the creation, modification and deletion of text notes and markup annotations on 3D objects (13.6.7.3.6, "3D comment note"). In addition, any time where the visual appearance of the annotation changes, the appearance stream shall be updated.
Multimedia	Requires support for multimedia (**Screen**) annotations (12.5.6.18, "Screen annotations"). See also 13.2, "Multimedia". The support that is required is for the general multimedia framework, not for an external player for any specific type of multimedia content. Negotiation of the choice of an external player is handled by the must honour (**MH**) and best efforts (**BE**) mechanism (13.2.2, "Viability") that is defined as part of the multimedia framework (13.2, "Multimedia").
U3D	Requires support for 3D data streams conforming to the U3D specification. This shall apply to the use of U3D in either 3D (13.6.3, "3D streams") or RichMedia annotations (13.7.2.2, "RichMediaSettings dictionary"). This also includes support for associated ECMAScripts. If a **V** key is present in its **Requirements** dictionary, it shall represent the version of U3D and not the PDF version.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Type	Description
PRC	Requires support for 3D data streams conforming to the PRC specification. This shall apply to the use of PRC in either 3D (13.6.3, "3D streams") or RichMedia annotations (13.7.2.2, "RichMediaSettings dictionary"). This also includes support for associated ECMAScripts. If a **V** key is present in its **Requirements** dictionary, it shall represent the version of PRC and not the PDF version.
Action	Requires support for actions in general (12.6, "Actions"), other than **GoTo** and **URI** actions (which are subsumed under the Navigation and Attachment requirements), **SetOCGState** (subsumed under **OCInteract**) and ECMAScript actions (which are separately declared with the **EnableJavaScripts** requirement).
EnableJavaScripts	Requires support for execution of ECMAScripts appearing in ECMAScript actions and in the ECMAScript name tree for document-level ECMAScripts. NOTE 2 ECMAScripts contained in 3D & RichMedia annotations are handled by their respective requirements.
Attachment	Requires support for displaying (to the user) the list of file attachments (see 7.11.4, "Embedded file streams") and enabling users to extract any existing attachments. In addition, support is provided for **GoToE** actions (12.6.4, "Action types") when located in any navigational element or trigger event. NOTE 3 The list of file attachments is taken from the **EmbeddedFiles** names tree (see 7.7.4, "Name dictionary") and any FileAttachment annotation (see 12.5.6.15, "File attachment annotations").
AttachmentEditing	In addition to the requirements of the **Attachment** value, support for adding new attachments into the **EmbeddedFiles** names tree (see 7.7.4, "Name dictionary"), deleting existing ones as well as modification of attachment attributes (e.g., name & description) are also required.
Collection	Requires support for displaying the embedded files referenced from the document's collection dictionary (12.3.5, "Collections") along with any associated metadata. Also requires that the user can extract or otherwise view the contents of each item in the collection. (PDF 2.0) For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **Collection** requirement should not be specified for the unencrypted wrapper document. NOTE 4 Although the unencrypted wrapper document is a collection, the intent of the wrapper is to enable PDF processors that are unable to decrypt the embedded encrypted payload document to present the content of the unencrypted wrapper document to assist users in understanding the cryptographic requirements of the encrypted payload document. Specifying the **Collection** requirement on the wrapper could discourage PDF processors incapable of displaying collections from presenting the unencrypted wrapper content.
CollectionEditing	In addition to the requirements of the **Collection** value, support for adding to the collection (12.3.5, "Collections"), deleting existing items as well as modification of collection item attributes and metadata, are also required.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Type	Description
DigSigValidation	Requires support for the validation of digital signatures (both document and certifying) that have been applied to the PDF including the handling of supplied revocation information. See 12.8, "Digital signatures", and 12.8.3.4.5, "Requirements for validation of PAdES signatures". This does not require the support for 12.8.2.2.2, "Validating signatures that use the DocMDP transform method" which is a separate requirement: **DigSigMDP**.
DigSig	In addition to the validation requirements of **DigSigValidation**, this specifies the requirements to support the application of a digital signature to a document (also known as signing). See 12.8, "Digital signatures".
DigSigMDP	In addition to the requirements of **DigSig** and **DigSigValidation**, this also requires support for modification detection analysis to determine if only allowable modifications have been made. See 12.8.2.2.2, "Validating signatures that use the DocMDP transform method".
RichMedia	Requires support for playing rich media annotations as specified in 13.7.2, "RichMedia annotations".
Geospatial2D	Requires support for processing provided geospatial information in the page content and associated resources. See 12.10, "Geospatial features".
Geospatial3D	Requires support for processing provided geospatial information in any 3D annotations. See 12.10, "Geospatial features". This type requires provision within the 3D Annotation, and also applies 3D requirements to Geospatial information.
DPartInteract	Requires support for the display of the DParts hierarchy and its use for navigation of the document parts. See 14.12, "Document parts".
SeparationSimulation	Requires support for simulation separations as described in 10.8, "Rendering for separations" and 10.8.3, "Separation simulation". NOTE 5 This is sometimes referred to as "Overprint Preview".
Transitions	Requires support for transitions/presentations (12.4.4, "Presentations") as well as transition actions (12.6.4.14, "Rendition actions")
Encryption	Requires support for the specific set of encryption parameters that are specified by the encryption dictionary provided as the value of the **Encrypt** key in the requirement dictionary (see "Table 273 — Entries common to all requirement dictionaries" and "Table 274 — Entries for specific types of requirements"). (PDF 2.0) For unencrypted wrapper documents for an encrypted payload document (see 7.6.7, "Unencrypted wrapper document") the **Encryption** requirement should not be specified for the unencrypted wrapper document. NOTE 6 The intent of the wrapper is to enable PDF processors that are unable to decrypt the embedded encrypted payload document to present the content of the unencrypted wrapper document to assist users in understanding the cryptographic requirements of the encrypted payload document. Specifying the **Encryption** requirement on the wrapper could discourage PDF processors incapable of decrypting the embedded encrypted payload from presenting the unencrypted wrapper content.

Additional requirement types, including ones identifying vendor-specific features, may be registered according to the rules described in Annex E, "Extending PDF".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.11.3 Requirement penalty values

Each **Requirements** dictionary shall contain an **S** key whose value should be chosen from a list of those defined in "Table 275 — Requirement types" but only for those features that are deemed critical for the successful rendering of the document. The values of the **Penalty** key is the penalty number expressed as an integer in the range [0, 100]. The presence of this requirements dictionary indicates that the associated feature is used or needed by the document.

A value of zero for the **Penalty** key would indicate that although the document uses this feature the need is optional. A value of 100 indicates that this document will not produce the author's intent unless the PDF processor can fully support this feature. Values between 0 and 100 are available to weight the value of this feature among other features in the same document requirements array as well as when contributing to the total penalty points to weigh against other documents in the choosing process if alternatives are available.

In the situation where the penalty values are being used to evaluate the presentation of the base PDF document, and there exist no other alternates, if the penalty value exceeds 100 then the PDF processor should not attempt to display or process the document.

12.11.4 Requirement versions

A requirement dictionary may include a **V** entry (*PDF 2.0*) that specifies a version number for a specific technology related to the requirement in question. It might be a PDF version number, an XFA version number (for those requirement related to XFA), a version of U3D or PRC, or a vendor-specific extension level.

Some PDF and XFA features have evolved over successive PDF versions. A PDF file may contain uses of a feature or an embedded data stream that can only be successfully interpreted by a PDF processor supporting a specific version of PDF or higher. This constraint may be indicated by including a **V** entry in the requirement dictionary identifying the PDF version.

The value of **V** shall be one of the following:

- A name that specifies a version number, represented as two or more decimal integers separated by a period. The number of decimals places is determined by the technology in question for which the version applies. For example, PDF and XFA versions are always two digits; others are always three.

NOTE This is specified as a name, not as a number, in order to avoid any ambiguities caused by inexact internal representation of decimal fractions. No non-numerals, except for the decimal point, can be present.

- An extensions dictionary that specifies a vendor-specific extension to a PDF version (see 7.12, "Extensions dictionary"). It shall only be used when describing a version of PDF for which a simple version (e.g., 1.7) is not sufficient and has no meaning for any other requirement. The extensions dictionary shall contain exactly one entry, whose key shall correspond to the registered prefix for the vendor that has defined this requirement type.

If the **V** entry is not present, determining if the requirement is satisfied shall be done without regard to version number.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

12.11.5 Requirement handlers

An alternative requirement handler is an alternative means for determining if the requirements associated with a document's features are satisfied. Traditionally, requirements handling has been accomplished with an ECMAScript segment in the document-level ECMAScripts stored in the document's name dictionary.

To create PDF documents that are compatible with processors that support PDF 1.7 and PDF 2.0, both the document requirements feature and the ECMAScript alternative requirements handler should be present in the document.

A document using the document requirements feature can specify that an ECMAScript function is to be disabled by including an **RH** entry whose value is an alternative requirement handler dictionary (or an array of such dictionaries), each of which identifies a handler that shall be disabled. "Table 276 — Entries in a requirement handler dictionary" describes the entries in an alternative requirement handler dictionary.

A requirement handler is a program that verifies certain requirements are satisfied. "Table 276 — Entries in a requirement handler dictionary" describes the entries in a requirement handler dictionary.

Table 276 — 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 ECMAScript requirement handlers) and *NoOp*. A value of *NoOp* allows older PDF processors to ignore unrecognised 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 ECMAScript action stored in the document name dictionary (see 7.7.4, "Name dictionary"). If the PDF processor understands the parent document requirements dictionary and can verify the requirement specified in that dictionary, it shall disable execution of the requirement handler identified in this dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

If an alternative requirement handler dictionary has an **S** entry with an unrecognised type, it shall be ignored.

12.11.6 Requirements processing

Document requirements can be presented for an individual document as the value of the **Requirements** key in that document's catalog. Document requirements shall be evaluated before execution of any document ECMAScripts. If requirements cannot be met, as determined by the computation of the penalty value as described in 12.11.3, "Requirement penalty values", then the processing of the document shall not continue.

© ISO 2020 – All rights reserved

For PDFs that are acceptable for processing, all subsequent processing shall occur without regard for the outcome of the requirements computation.

NOTE 1 That is, there is no formal connection between the requirement type and the operation of the associated feature(s).

If the reader encounters an unsupported feature (whether or not that feature was declared as a requirement), it shall take the normal fallback actions.

NOTE 2 The most common fallback is to do nothing — that is, to ignore the use of the unsupported feature. Other common fallbacks can include presenting a static annotation appearance in place of a dynamic annotation.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

13 Multimedia features

13.1 General

This clause describes those features of PDF that support embedding and playing multimedia content. It contains the following subclauses:

- 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 deprecated features superseded by 13.7, "

- Rich media".
- 13.6, "3D Artwork" describes the capability of embedding three-dimensional graphics in a document, introduced in PDF 1.6.

13.7, "

- Rich media" describes rich media annotations providing a common framework for video, audio, animations and other multimedia presentations.

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 interactive PDF processor 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 can 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 interactive PDF processors. Authors may control how old interactive PDF processors 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 summarises 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 12.6.4.14, "Rendition actions") shall be used to begin the playing of

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclauses have references to corresponding elements in the Synchronized Multimedia Integration Language (SMIL 3.0) standard.

13.2.2 Viability

When playing multimedia content, the interactive PDF processor 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 293 — Entries in a media screen parameters dictionary") allows the playback volume to be set by means of the **V** entry in its **MH** and **BE** dictionaries (see "Table 294 — Entries in a media screen parameters MH/BE dictionary").

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 interactive PDF processors. If an entry that is unrecognised by the viewer is in the **MH** dictionary, the object shall be considered non-viable. If an unrecognised 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 unrecognised key or an unrecognised value for a recognised key.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 can be dependent on one another. That is, an interactive PDF processor or player can 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 subclauses:

- Rendition ("Table 278 — Entries in a rendition MH/BE dictionary")
- Media clip data ("Table 287 — Entries in a media clip data MH/BE dictionary")
- Media clip section ("Table 289 — Entries in a media clip section MH/BE dictionary")
- Media play parameters ("Table 291 — Entries in a media play parameters MH/BE dictionary")
- Media screen parameters ("Table 294 — Entries in a media screen parameters MH/BE dictionary")

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 interactive PDF processor should play the first viable media rendition it encounters in the tree (see 13.2.2, "Viability").

NOTE 1 "Table 277 — Entries common to all rendition dictionaries" 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 32 — Entries in the name dictionary"). ECMAScript actions (see 12.6.4.17, "ECMAScript 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 interactive PDF processor needs to 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). As a

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

consequence it is recommended that a document 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 278 — Entries in a rendition MH/BE dictionary"), 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 279 — Entries in a media criteria dictionary").

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 unrecognised by the interactive PDF processor, 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 277 — 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 is non-viable if the interactive PDF processor does not recognise 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 ECMAScript actions.
MH	dictionary	(*Optional*) A dictionary whose entries (see "Table 278 — Entries in a rendition MH/BE dictionary") shall be honoured for the rendition to be considered viable.
BE	dictionary	(*Optional*) A dictionary whose entries (see "Table 278 — Entries in a rendition MH/BE dictionary") shall only be honoured in a "best effort" sense.

Table 278 — Entries in a rendition MH/BE dictionary

Key	Type	Value
C	dictionary	(*Optional*) A media criteria dictionary (see "Table 279 — Entries in a media criteria dictionary"). 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 a **MH** or a **BE** dictionary. The only exception is that if an entry in a **BE** dictionary is unrecognised by the interactive PDF processor, it shall not affect the viability of the object.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 279 — 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.
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 280 — Entries in a minimum bit depth dictionary") 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 281 — Entries in a minimum screen size dictionary") 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 interactive PDF processor 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 dictionary (see "Table 29 — Entries in the catalog dictionary"). If this entry is present and non-empty, the version of multimedia constructs fully supported by the interactive PDF processor shall be within the specified range in order for this object to be viable.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 interactive PDF processor 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 280 — 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 specifying the minimum screen depth (in bits) of the monitor for the rendition to be viable.
M	integer	(*Optional*) A monitor specifier (see "Table 304 — Monitor specifier values") that specifies which monitor the value of **V** should be tested against. If the value is unrecognised, the object shall not be viable. Default value: *0*.

Table 281 — 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 304 — Monitor specifier values") that specifies which monitor the value of **V** should be tested against. If the value is unrecognised, the object shall be not viable. Default value: *0*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

13.2.3.2 Media renditions

"Table 282 — Additional entries in a media rendition dictionary" 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 290 — Entries in a media play parameters dictionary") shall contain a **PL** entry, whose referenced media players

© ISO 2020 – All rights reserved

dictionary (see 13.2.7.2, "Media players dictionary") has a non-empty **MU** array or a non-empty **A** array.

Table 282 — 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 290 — Entries in a media play parameters dictionary") 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 293 — Entries in a media screen parameters dictionary") 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 283 — Additional entries specific to a selector rendition dictionary"). 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 can be used, for example, to specify that a large video clip need be used on high-bandwidth machines and a smaller clip need be used on low-bandwidth machines.

Table 283 — 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 valid.

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

© ISO 2020 – All rights reserved

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 for media clip dictionaries are listed in "Table 284 — Entries common to all media clip dictionaries".

Table 284 — 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 is non-viable if the interactive PDF processor does not recognise 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 285 — Additional entries in a media clip data dictionary".

NOTE 1 Media clip data dictionaries can reference a URL to a streaming video presentation or a movie embedded in the PDF file.

Table 285 — 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; shall not be present 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*.
P	dictionary	(*Optional*) A media permissions dictionary (see "Table 286 — Entries in a media permissions dictionary") 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 alternative text descriptions for the media clip data in case it cannot be played; see 14.9.2.4, "Multi-language text arrays"
PL	dictionary	(*Optional*) A media players dictionary (see 13.2.7.2, "Media players dictionary") that identifies, among other things, players that are valid and not valid for playing the media. If the media players dictionary is non-viable, the media clip data shall be non-viable.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
MH	dictionary	(*Optional*) A dictionary whose entries (see "Table 287 — Entries in a media clip data MH/BE dictionary") shall be honoured for the media clip data to be considered viable.
BE	dictionary	(*Optional*) A dictionary whose entries (see "Table 289 — Entries in a media clip section MH/BE dictionary") should only be honoured in a "best effort" sense.

The media clip data object is non-viable if the object referenced by the **D** entry does not contain a **Type** entry, the **Type** entry is unrecognised, 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 interactive PDF processor, and the form XObject shall be rendered as if it were any other data type.

The **F** and **D** entries in the media play parameters dictionary (see "Table 290 — Entries in a media play parameters dictionary") 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 an interactive PDF processor 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 286 — Entries in a media permissions dictionary") specifying the manner in which the data referenced by the media may be used by an interactive PDF processor. These permissions allow authors control over how their data are exposed to operations that could allow it to be copied. If the dictionary contains unrecognised entries or entries with unrecognised values, it is considered non-viable, and the interactive PDF processor shall not play the media.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 286 — 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.

© ISO 2020 – All rights reserved

Key	Type	Value
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 — Standard security handler user access permissions") 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 — Standard security handler user access permissions") are set, or both. *(TEMPALWAYS)* Always allowed. Default value: *(TEMPNEVER).* An unrecognised value shall be treated as *(TEMPNEVER).*

An unrecognised value shall be treated as *(TEMPNEVER).*

The **BU** entry in the media clip data **MH** and **BE** dictionaries (see "Table 287 — Entries in a media clip data MH/BE dictionary") 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 2 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 (i.e., the target host exists).
- Absolute URls 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 3 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 catalog dictionary (see 7.7.2, "Document catalog dictionary").
- 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 287 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

13.2.4.3 Media clip section

A *media clip section dictionary* (see "Table 288 — Additional entries in a media clip section dictionary") 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 288 — 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 alternative 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 289 — Entries in a media clip section MH/BE dictionary") shall be honoured for the media clip section to be considered viable.
BE	dictionary	(*Optional*) A dictionary whose entries (see "Table 289 — Entries in a media clip section MH/BE dictionary") 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 289 — Entries in a media clip section MH/BE dictionary") 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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 valid.
- 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 cannot be honoured 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 289 — 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 290 — 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 valid and not valid 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 291 — Entries in a media play parameters MH/BE dictionary") shall be honoured for the media play parameters to be considered viable.
BE	dictionary	(*Optional*) A dictionary whose entries (see "Table 291 — Entries in a media play parameters MH/BE dictionary") shall only be honoured in a "best effort" sense.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 291 — 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 not be permitted. Default value: *100*.
C	boolean	(*Optional*) A flag specifying whether to display a player-specific controller user interface when playing. EXAMPLE 1 play/pause/stop controls. Default value: *false*
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. *0* The 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 minimised. 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 unrecognised 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 unrecognised value, the object shall be considered non-viable.
D	dictionary	(*Optional*) A media duration dictionary (see "Table 292 — Entries in a media duration dictionary"). 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 2 The first frame is displayed. Relevant only for media that may be paused. Default value: *true*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
RC	number	(*Optional*) Specifies the number of iterations of the duration **D** to repeat. Negative values shall be invalid; non-integral values shall be permitted. NOTE 6 Similar to SMIL's *repeatCount* attribute. Zero means repeat forever. Default value: *1.0*.

The value of the **D** entry is a *media duration dictionary*, whose entries are shown in "Table 292 — Entries in a media duration dictionary". 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 292 — 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 is non-viable if the interactive PDF processor does not recognise 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 300 — Entries in a timespan dictionary"). A negative duration shall not be permitted.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

13.2.6 Media screen parameters

13.2.6.1 General

A *media screen parameters dictionary* (see "Table 293 — Entries in a media screen parameters dictionary") shall specify where a media object should be played. It shall contain **MH** and **BE** dictionaries (see "Table 294 — Entries in a media screen parameters MH/BE dictionary"), 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.

Document-based security attacks are possible if windows containing arbitrary media content can be displayed without indicating to the user that the window is only hosting a media object. This recommendation can 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 293 — 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 294 — Entries in a media screen parameters MH/BE dictionary") shall be honoured for the media screen parameters to be considered viable.
BE	dictionary	(*Optional*) A dictionary whose entries (see "Table 294 — Entries in a media screen parameters MH/BE dictionary") should be honoured.

Table 294 — 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. Valid values are: *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*. Unrecognised value in **MH**: object is non-viable; in **BE**: treat as default value.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	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 interactive PDF processor 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 304 — Monitor specifier values") 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). Unrecognised 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 295 — Entries in a floating window parameters dictionary") 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 295 — Entries in a floating window parameters dictionary". 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.

Unrecognised 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 295 — 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. Valid values are: *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 "Table 294 — Entries in a media screen parameters MH/BE dictionary") 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**. Valid values are: *0* Upper-left corner *1* Upper centre *2* Upper-right corner *3* Centre left *4* Centre *5* Centre right *6* Lower-left corner *7* Lower centre *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). Valid values are: *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*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
R	integer	(*Optional*) Specifies whether the floating window may be resized by a user. Valid values are: *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*.
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 interactive PDF processor may provide default text.

13.2.6.2 Media offset dictionary

A *media offset dictionary* ("Table 296 — Entries common to all media offset dictionaries") 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 296 — 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 297 — Additional entries in a media offset time dictionary") *F* A media offset frame dictionary (see "Table 298 — Additional entries in a media offset frame dictionary") *M* A media offset marker dictionary (see "Table 299 — Additional entries in a media offset marker dictionary") The rendition is non-viable if the interactive PDF processor does not recognise the value of this entry.

Table 297 — Additional entries in a media offset time dictionary

Key	Type	Value
T	dictionary	(*Required*) A timespan dictionary (see "Table 300 — Entries in a timespan dictionary") that shall specify a temporal offset into a media object. Negative timespans are prohibited in this context. The media offset time dictionary is non-viable if its timespan dictionary is non-viable.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 298 — 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 out of range.

Table 299 — 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 300 — Entries in a timespan dictionary".

Table 300 — 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 is non-viable if the interactive PDF processor does not recognise the value of this entry.
V	number	(*Required*) The number of seconds in the timespan. Non-integral values shall be allowed. Negative values shall not be used. 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 subclause defines several dictionary types that are referenced by the previous subclauses.

13.2.7.2 Media players dictionary

13.2.7.2.1 General

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 (see “Table 301 — Entries in a media players dictionary”) references *media player info*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

dictionaries (see 13.2.7.3, "Media player info dictionary") that shall provide specific information about each player.

Table 301 — 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 302 — Entries in a media player info dictionary") 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 302 — Entries in a media player info dictionary") 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 302 — Entries in a media player info dictionary") 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 **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 unrecognised 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.

13.2.7.2.2 Algorithm: Media Player

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:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 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 302 — Entries in a media player info dictionary") shall associate information with a particular version or range of versions of a player. The **PID** entry shall provide information about the player, as described in 13.2.7.4, "Software identifier dictionary".

Table 302 — 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 honoured 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 honoured 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 303 — Entries in a software identifier dictionary". An interactive PDF processor uses this information to determine whether a given media player may be used in a given situation. If the dictionary contains keys that are unrecognised by the interactive PDF processor, it shall be considered to be partially recognised. The interactive PDF processor may or may not decide to treat the software identifier as viable, depending on the context in which it is used.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 303 — 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 13.2.7.4.3, "Software URIs").
L	array	(*Optional*) The lower bound of the range of software versions that this software identifier dictionary specifies (see 13.2.7.4.4, "Version arrays"). 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 13.2.7.4.4, "Version arrays"). 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 13.2.7.4.4, "Version arrays"). 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 13.2.7.4.4, "Version arrays"). 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 3.0's *systemOperatingSystem* attribute. There may not be multiple copies of the same identifier in the array. An empty array represents all operating systems. Default value: an empty array.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

13.2.7.4.2 Algorithm: software identifier

The following procedure shall be used to determine whether a piece of software is considered to match a software identifier dictionary:

a) The software name shall match the name specified by the **U** entry (see 13.2.7.4.3, "Software URIs").

b) The software version shall be within the interval specified by the **L**, **H**, **LI**, and **HI** entries (see 13.2.7.4.4, "Version arrays").

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.

13.2.7.4.3 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 it is not recognised by the interactive PDF processor, the software shall be considered a non-match. The syntax of URIs of this scheme is

"vnd.adobe.swname:" *software_name*

© ISO 2020 – All rights reserved

where *software_name* shall be reg_name as defined in *Internet RFC 3986*. Also, *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, "Extending PDF").

EXAMPLE The URI for Adobe Acrobat is

```
vnd.adobe.swname:ADBE_Acrobat.
```

13.2.7.4.4 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

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.

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].

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 is 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 304 — Monitor specifier values".

Table 304 — 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Value	Description
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

The features described in this subclause are deprecated with PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

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 188 — Additional entries specific to a sound annotation" and "Table 212 — Additional entries specific to a sound action") 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 — Entries common to all stream dictionaries". 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 305 — Additional entries specific to a sound object" shows the additional dictionary entries specific to a sound object.

Table 305 — 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
E	name	(*Optional*) The encoding format for the sample data: *Raw* Unspecified or unsigned values in the range 0 to 2B - 1 *Signed* Twos-complement values *muLaw* m-law–encoded samples *ALaw* A-law–encoded samples Default value: *Raw*.
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 — Entries common to all stream dictionaries" 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 1) preceding the corresponding sample for the right (channel 2).

To maximise the portability of PDF documents containing embedded sounds, interactive PDF processors 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 subclause are deprecated with PDF 2.0. They are superseded by the general multimedia framework described in 13.2, "Multimedia".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

PDF may 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 306 — Entries in a movie dictionary") that shall describe the static characteristics of the movie and a *movie activation dictionary* ("Table 307 — Entries in a movie activation dictionary") that shall specify how it shall be presented.

Table 306 — Entries in a movie dictionary

Key	Type	Value
F	file specification	(*Required*) A file specification identifying a self-describing movie file. 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 307 — Entries in a movie activation dictionary

Key	Type	Value
Start	(various)	(*Optional*) The starting time shall be nominally a non-negative 64-bit integer, specified as follows: • If it is representable as a 32 bit integer, it shall be specified as such. • If it is not representable as a 32 bit 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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*.
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 interactive PDF processor 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 306 — Entries in a movie dictionary").
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] centres the window on the screen. Default value: *[0.5 0.5]*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

13.5 Alternate presentations

The features described in this clause are deprecated with PDF 2.0.

Beginning with PDF 1.4, a PDF document may contain *alternate presentations*, which specify alternative ways in which the document may be viewed. The optional **AlternatePresentations** entry (PDF 1.4) in a document's name dictionary (see "Table 32 — Entries in the name dictionary") contains a name tree that maps name strings to the alternate presentations available for the document.

Since PDF processors are not required to support alternate presentations, authors of documents containing alternate presentations should define the files such that something useful and meaningful

© ISO 2020 – All rights reserved

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 12.6.4.17, "ECMAScript actions") initiated by user action on an interactive form element (see 12.7, "Forms").

"Table 308 — Entries in a slideshow dictionary" shows the entries in a slideshow dictionary.

Table 308 — 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. 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 1 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 can 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 2 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 PDF processors 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
         …
      >>
endobj

3 0 obj                                                %The name dictionary
      <</AlternatePresentations 4 0 R>>
endobj
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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)
      >>
endobj

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*) PDF documents may specify how embedded 3D artwork shall be rendered, coloured, lit, and cross-sectioned, without the use of embedded ECMAScript. They may also specify state information that shall be applied to individual nodes (3D graphics objects or collections thereof) in the 3D artwork, such as visibility, opacity, position, or orientation.
- Pages containing 3D artwork may be printed.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- Users may pan, zoom and rotate 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 programmatic intervention.
- ECMAScripts and other software may programmatically manipulate objects in the artwork, creating dynamic presentations in which objects move, spin, appear, and disappear. See ISO/DIS 21757-1.
- (*PDF 1.7*) The activation of 3D artwork can trigger the display of additional user interface items in an interactive PDF processor. 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 subclauses 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. (see13.6.2 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. PDF 2.0 extends PDF to support the ISO 14739-1 *Product Representation Compact (PRC)* file format.
- 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 a way to include 3D artwork in PDF documents. Rich media annotations are another method (see 0 "

Rich media"). "Table 309 — Additional entries specific to a 3D annotation" shows the entries specific to a 3D annotation dictionary. "Table 166 — Entries common to all annotation dictionaries" 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 166 — Entries common to all annotation dictionaries") that has a normal appearance (the **N** entry in "Table 170 — Entries in an appearance dictionary"). This appearance may be used by applications that do not support 3D annotations and by all applications for the initial display of the annotation.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 309 — 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 311 — Entries in a 3D stream dictionary"): • 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 310 — Entries in a 3D activation dictionary") 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 an ECMAScript animation. Interactive PDF processors 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*.
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**.
3DU	dictionary	(*Optional; PDF 2.0*) A 3D units dictionary that specifies the units definitions for the 3D data associated with this annotation. See "Table 325 — Entries in a 3D units dictionary".
GEO	dictionary	*(Optional; PDF 2.0)* For **Geospatial3D** requirement type, a geospatial information section may be present as an attribute within a 3D Annotation. There are further conditions placed on the **GPTS** and **LPTS** arrays within the geo-reference coordinate tables to include 3D point values. See 12.10.2, "Geospatial measure dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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, interactive PDF processors 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 centre 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 13.6.3, "3D streams". 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 13.6.4, "3D views", 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 311 — Entries in a 3D stream dictionary").

The **3DA** entry shall be an activation dictionary (see "Table 310 — Entries in a 3D activation dictionary") 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. PDF writers need to 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") can 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 310 — 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. At any one time, only a single page shall be considered open in an interactive PDF processor, even though more than one page might be visible, depending on the page layout. Default value: *XA*. 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 interactive PDF processors, *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*. In non-interactive PDF processors, 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. At any one time, only a single page shall be considered open in an interactive PDF processor, even though more than one page might be visible, depending on the page layout. Default value: *PI*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
DIS	name	(*Optional*) A name specifying the state of the artwork instance upon deactivation of the annotation. Valid values are: *U* uninstantiated *I* instantiated *L* live Default value: *U*. NOTE 1 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 PDF processors.
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 2 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*
Style	name	(*Optional; PDF 2.0*) The value shall be either *Embedded* or *Windowed*. An interactive PDF processor shall support both *Embedded* and *Windowed*. Default value: *Embedded*
Window	dictionary	(*Optional; PDF 2.0*) A RichMediaWindow Dictionary that describes the size and position of the floating user interface window when the value for **Style** is set to *Windowed*. See "Table 339 — Entries in a RichMediaWindow dictionary" for a detailed description.
Transparent	boolean	(*Optional; PDF 2.0*) A flag that indicates whether the interactive PDF processor shall display the underlying page content through the transparent areas of the rich media content (where the alpha value is less than 1.0). Default value: *false*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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.

© ISO 2020 – All rights reserved

- *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 PDF processors that have ECMAScript 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 can 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 can 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 can 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.

The **Style** and **Window** entries describe the mechanism that an interactive PDF processor shall use to present the 3D content in a floating window.

The **Transparent** entry provides a flag for specifying whether the background of a 3D view shall be rendered using transparency. If **Transparent** has the value of *true*, then some area of the page may be visible through the background if the resulting alpha value at that location is less than 1.0. See also the **Transparent** entry of "Table 310 — Entries in a 3D activation dictionary".

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 311 — Entries in a 3D stream dictionary", 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 customised behaviours or presentations.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 311 — 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 following are the only recognised values: *U3D* which specifies the Universal 3D file format. *PRC* which specifies the PRC file format. (*PDF 2.0*)
VA	array	(*Optional*) An array of *3D view dictionaries*, each of which specifies a named preset view of this 3D artwork (see 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 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 ECMAScript.
OnInstantiate	stream	(*Optional*) An ECMAScript 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 interactive PDF processors 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*.
ColorSpace	name or array	(*Optional, PDF 2.0*) The RGB colour space in which the 3D artwork's colour values are encoded. Valid values are the name *DeviceRGB*, an array specifying a valid **CalRGB** color space (see 8.6.5.3 "CalRGB colour spaces"), or an array specifying a valid RGB-based **ICCBased** color space (see 8.6.5.5 "ICCBased colour spaces"). If this key is not present, the colour space for the 3D artwork colour values are considered undefined and a PDF processor may choose any appropriate RGB-based colour space, such as sRGB.

The **Subtype** entry specifies the format of the 3D stream data. PDF processors 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE PDF processors need to follow the approach of falling back to the normal appearance with regard to entries in other dictionaries that can 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 interactive PDF processors 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 an ECMAScript script that shall be executed by applications that support ECMAScript 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 interactive PDF processors 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 interactive PDF processor.

“Table 312 — Entries in an 3D animation style dictionary” shows the entries in an animation style dictionary.

Table 312 — 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 313 — Animation styles" for valid values. If an animation style is encountered other than those described in "Table 313 — Animation styles", 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*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TM	number	(*Optional*) A positive number specifying the time multiplier that shall 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 313 — Animation styles") 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 interactive PDF processor.
- t_0 is the time when the interactive PDF processor 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 313 — Animation styles

Style	Description
None	Keyframe animations shall not be driven directly by the interactive PDF processor. This value shall be used by documents that are intended to drive animations through an alternative means, such as ECMAScript. 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. $t = 0.5(r_1 - r_0)\big(1 - cos\big(m(t_a - t_0)\big)\big) + r_0$ $p = 2 * pi/m$

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 314 — Entries in a 3D reference dictionary"):

- 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 314 — Entries in a 3D reference dictionary"), 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 314 — 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 87 — Default view of artwork" through "Figure 89 — Shared artwork (annotations 2 & 3) modified" 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
201 0 obj                                %3D reference dictionary
      <</Type /3DRef
         /3DD 200 0 R                    %Reference to the actual 3D artwork.
      >>
endobj
```

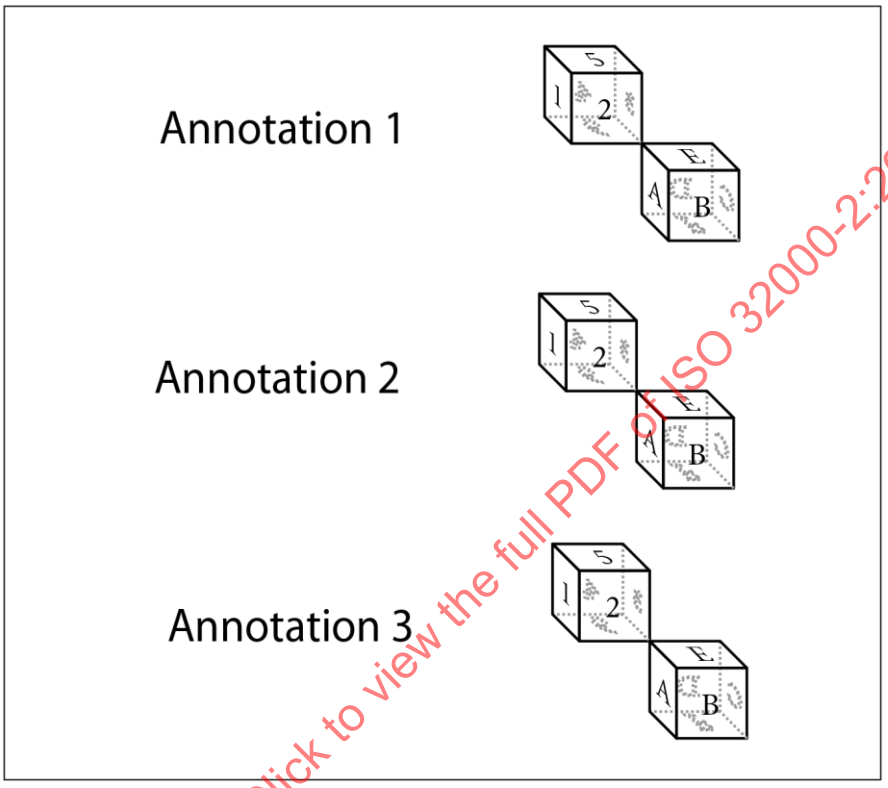


Figure 87 — Default view of artwork

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

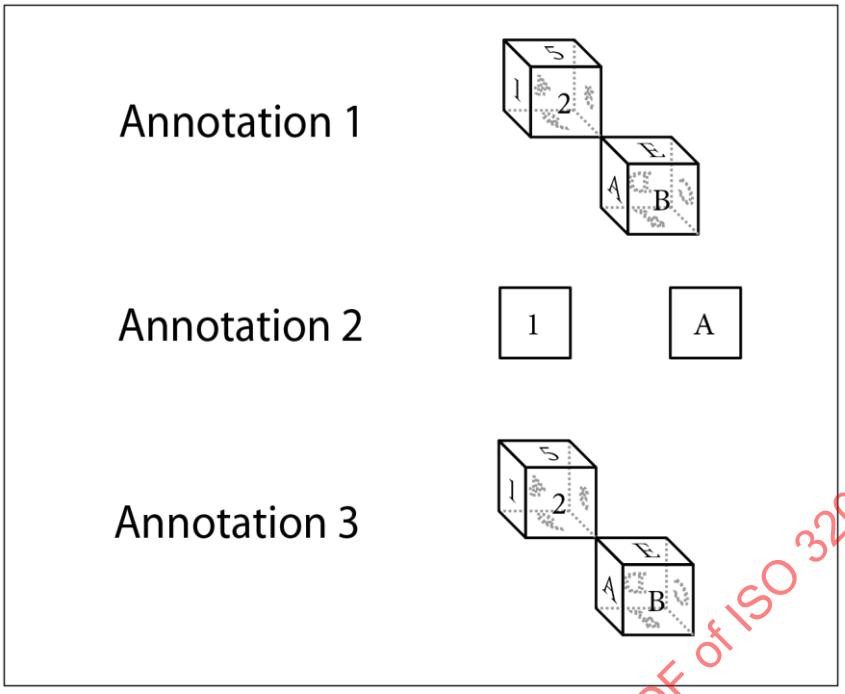


Figure 88 — Annotation 2 rotated

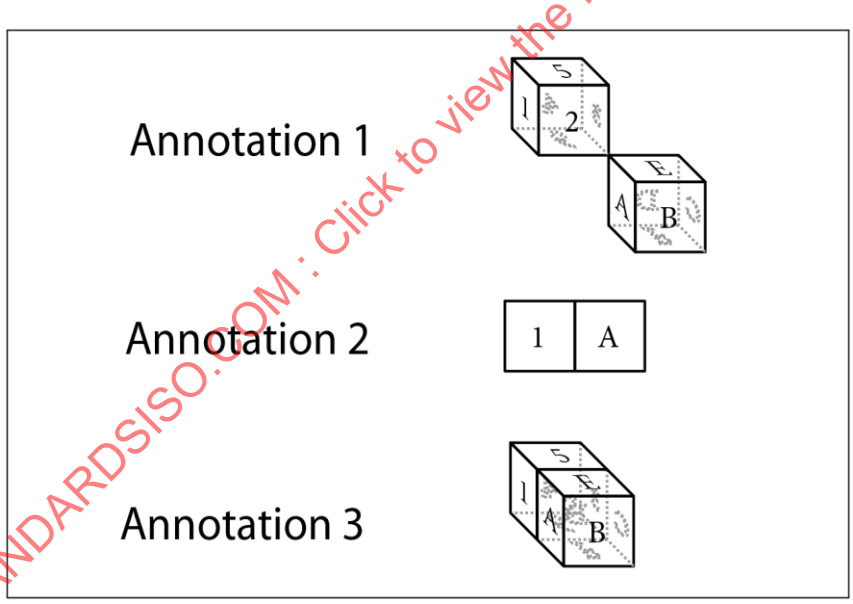


Figure 89 — Shared artwork (annotations 2 & 3) modified

"Figure 87 — Default view of artwork" shows the same initial view of the artwork in all three annotations. "Figure 88 — Annotation 2 rotated" shows the results of rotating the view of the artwork within Annotation 2. "Figure 89 — Shared artwork (annotations 2 & 3) modified" 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"

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 ECMAScript. 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 can 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 315 — Entries in a 3D view dictionary".

Table 315 — 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 12.6.4.16, "Go-To-3D-View actions"). If omitted, the external name **XN** shall be used.
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 in the 3D stream object shall specify the matrix. NOTE 1 There is no corresponding **MS** field value for the PRC file format, that would correspond to a 3D stream object of type PRC. *M* is the only valid entry for 3D stream objects of type PRC (or it can be omitted). If omitted, the view specified in the 3D artwork shall be used.
MA	array	(*Optional; PDF 2.0*) An array of 3D measurement/markup dictionaries, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. See "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes" for the definition of a 3D measurement dictionary.
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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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. PDF writers should specify only a single text string, not an array, for this entry. NOTE 2 Do not confuse View Nodes with nodes. A View Node is a parameter in the 3D artwork that specifies a view; a node is a PDF dictionary that specifies 3D graphic objects or collections thereof.
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 centre of orbit for this view; see discussion following "Table 315 — Entries in a 3D view dictionary". If this entry is not present, the interactive PDF processor shall determine the centre 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 *(capital letter 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 that shall 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 an interactive PDF processor. Default value: *false*

For any view, the PDF 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:

- It can 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 is required to be associated with a specific view (not with the camera position defined by the 3D view dictionary). The interactive PDF processor shall draw it only when the user navigates using the 3D view, and shall not draw it when the user happens to navigate to the same orientation by manual means.
- The interactive PDF processor shall draw it only if the user has not invoked any actions that alter the artwork-to-world matrix.

NOTE 2 Failure to abide by these restrictions could result in misalignment of the overlay with the rendered 3D graphics.

The **CO** entry specifies the distance from the camera to the *centre of orbit* for the 3D view, which is the point around which the camera shall rotate when performing an orbit-style navigation. "Figure 90 — Rotation around the centre of orbit" illustrates camera positioning when orbiting around the centre of orbit.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

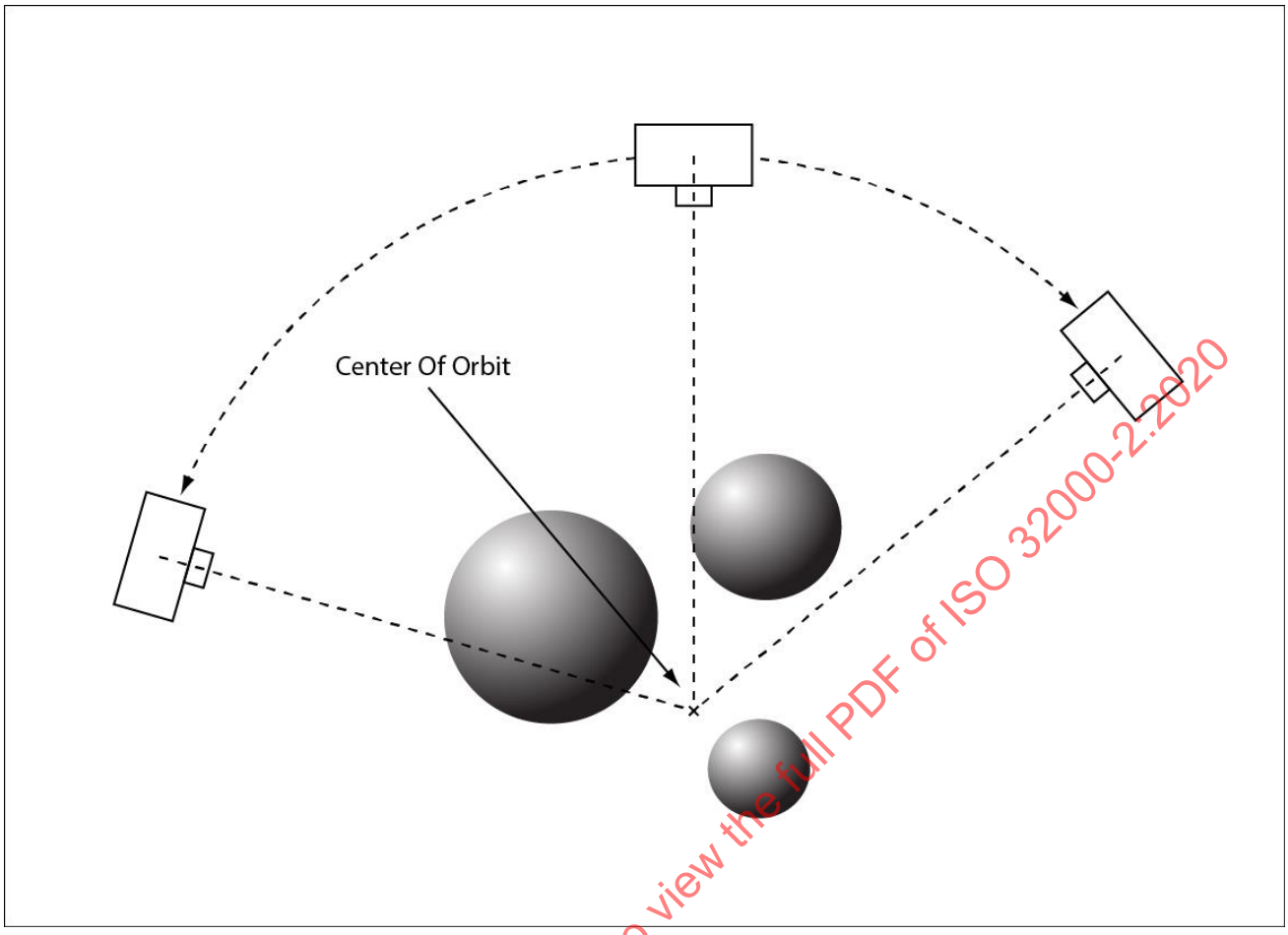


Figure 90 — Rotation around the centre of orbit

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 ECMAScript. See 13.6.4.7, "3D node dictionaries"

13.6.4.2 Projection dictionaries

A *projection dictionary* (see "Table 316 — Entries in a projection dictionary") 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

PDF 1.6 introduces *near/far clipping*. This type of clipping defines a *near plane* and a *far plane* (as shown in "Figure 91 — Perspective projection of 3D artwork onto the near plane"). 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 in "Table 316 — Entries in a projection dictionary".

Table 316 — 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	(*Required if **Subtype** is P*; *meaningful only if the value of* **CS** *is XNF*) 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 centred 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 allowed 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 centred along the *z* axis in the camera coordinate system (see "Figure 91 — Perspective projection of 3D artwork onto the near plane"). The cone intersects with the near plane, forming a circular area on the near plane. "Figure 92 — Objects projected onto the near clipping plane, as seen from the position of the camera" shows this circle and graphics from the position of the camera.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

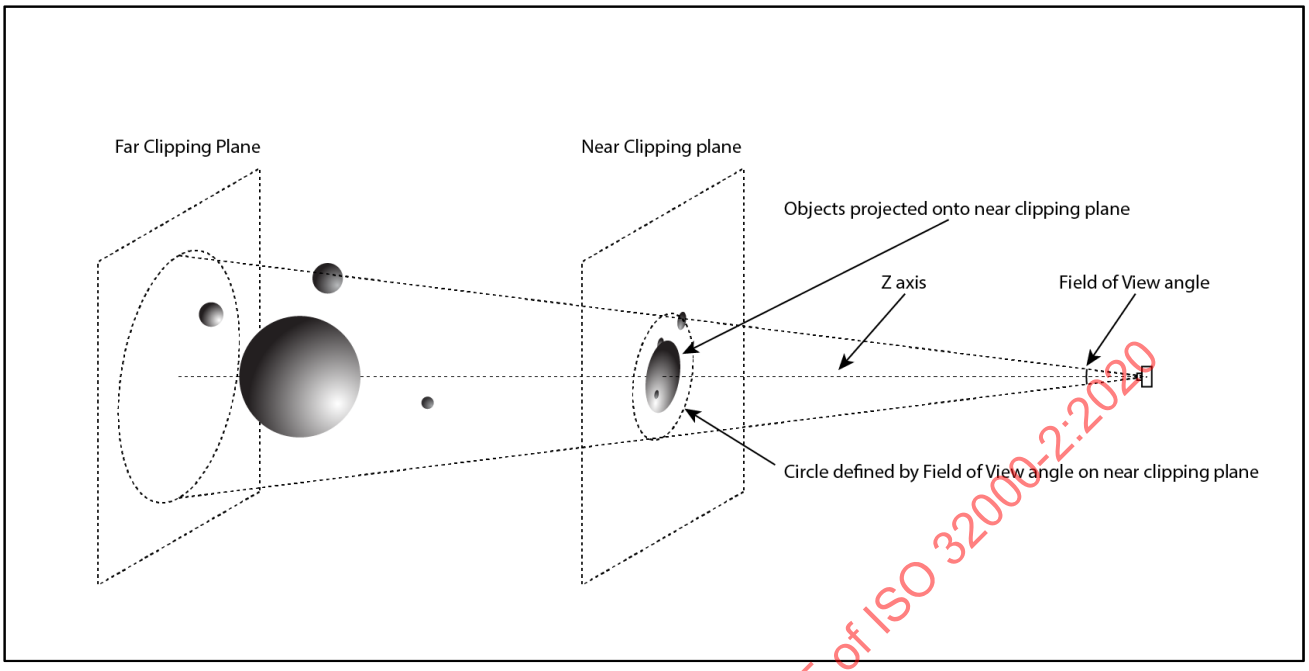


Figure 91 — Perspective projection of 3D artwork onto the near plane

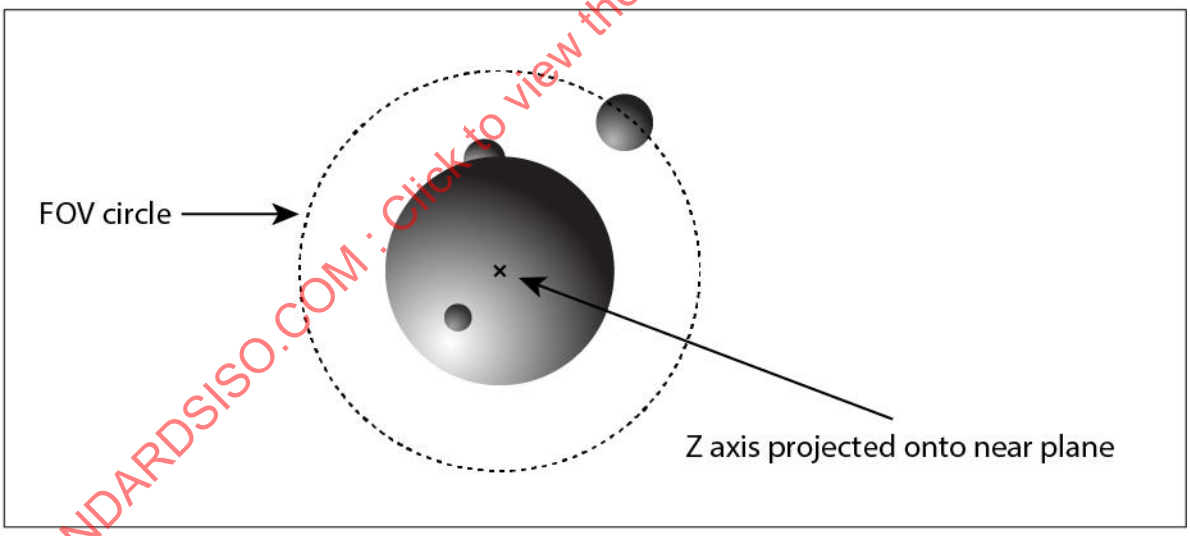


Figure 92 — Objects projected onto the near clipping plane, as seen from the position of the camera

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 93 — Positioning and scaling the near plane onto the annotation's 3D view box"). 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 94 — 3D annotation positioned on the page" 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.

© ISO 2020 – All rights reserved

ISTANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

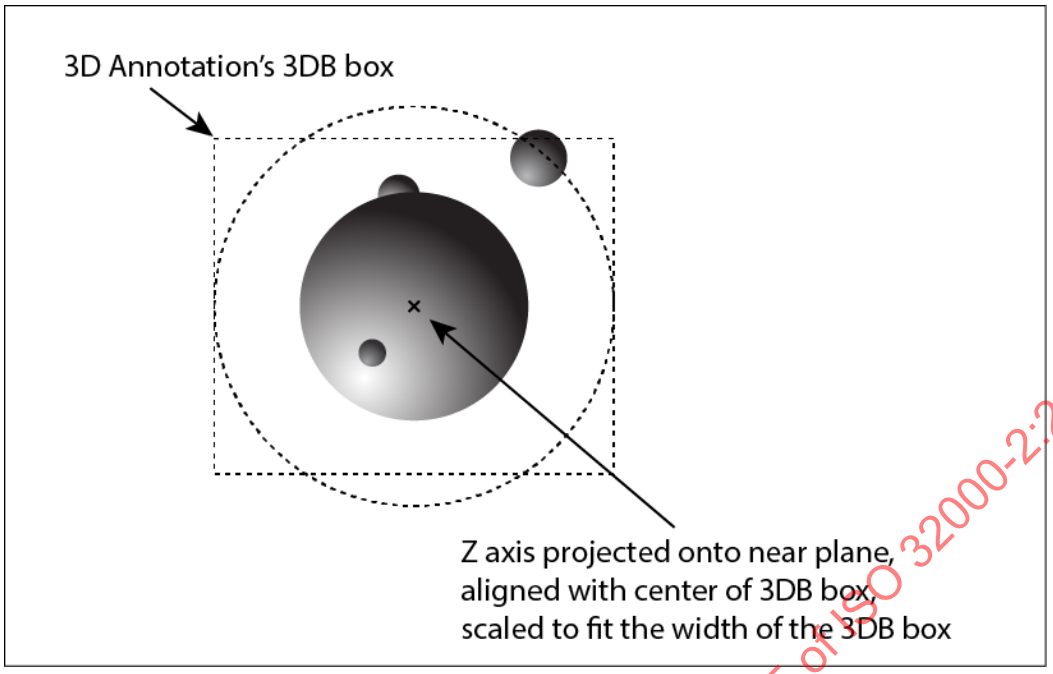


Figure 93 — Positioning and scaling the near plane onto the annotation's 3D view box

Figure 94 — 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 317 — Entries in a 3D background dictionary" 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").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE An annotation's normal appearance needs to 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 317 — 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 309 — Additional entries specific to a 3D annotation"). Default value: *false*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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 can be filled with opaque colours, they can be stroked as a "wireframe", or the artwork can be rendered with special lighting effects.

NOTE 2 A render mode dictionary enables document authors to customise the rendered appearance of 3D artwork to suit the needs of the intended consumer, without reauthoring the artwork. For interactive PDF processors concerned strictly with geometry, complex artwork rendered using the **Wireframe** or **Points** style can have much better performance without the added overhead of texturing and lighting effects. Artwork in a document intended for print can 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 318 — Entries in a render mode dictionary" shows the entries in a render mode dictionary.

Table 318 — 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 319 — Render modes" for specific values. If an unrecognised value is encountered, then this render mode dictionary shall be ignored.

© ISO 2020 – All rights reserved

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 colour that shall 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 be *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 192 — Entries in an appearance characteristics dictionary"). 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 is 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 319 — Render modes" describes the render modes that may be specified in a render mode dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 319 — Render modes

Mode	Description
Solid	Displays textured and lit geometric shapes. The **AC** entry shall be ignored.
SolidWireframe	Displays textured and lit geometric shapes 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 with an added level of transparency. The **AC** entry shall be ignored.
TransparentWireframe	Displays textured and lit geometric shapes 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Mode	Description
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 319 — Render modes", 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 320 — Entries in a 3D lighting scheme dictionary" shows the entries in a 3D lighting scheme dictionary.

Table 320 — 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 321 — 3D lighting scheme styles").

"Table 321 — 3D lighting scheme styles" describes the supported lighting schemes. With the exception of the Artwork lighting style, all the lights specified in "Table 321 — 3D lighting scheme styles" 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 321 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Scheme	Description		
None	No lights shall be used. That is, lighting specified in the 3D artwork shall be ignored.		
White	Three blue-gray 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-gray 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>
Hard	Three gray 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>

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Scheme	Description		
Cube	Six gray 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 gray 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>	

If a lighting scheme style is encountered other than those described in "Table 321 — 3D lighting scheme styles" the lighting scheme dictionary containing that entry shall be ignored.

NOTE 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 interactive PDF processor to ignore unrecognised 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 interactive PDF processors 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 322 — Entries in a 3D cross section dictionary" shows the entries in a 3D cross section dictionary.

Table 322 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
C	array	(*Optional*) A three element array specifying the centre 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; meaningful only if **IV** is true*) 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. Default value: *[/DeviceRGB 0 1 0]* representing the colour green.
ST	boolean	(*Optional; PDF 2.0*) *Show Transparent*. A flag indicating that the portion of a model on the cut side of a section plane shall be viewed using a transparent render mode. If *true*, the clipped portion of the model shall be drawn as transparent regardless of the render mode of the non-cut portion of the model. The recommend appearance uses graphical settings identical or similar to those used by the existing Transparent 3DRenderMode in "Table 319 — Render modes" already supported by the 13.6.4.4, "3D render mode dictionaries". If *false*, the clipped portion of the model shall not be drawn. Default value: *false*.
SC	boolean	(*Optional; PDF 2.0*) *Section Capping*. A flag indicating whether 3D Nodes and the section plane shall be analysed and rendered as if cutting through a solid object. If *true*, an algorithm is applied with the result that additional temporary geometry may be created to represent a section cap. The section cap shall be visible within the area enclosed by the intersection of the 3D Node 3D geometry with the section plane. If the flag is *false*, no additional temporary geometry shall be created to represent a section cap. Default value: *false*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The ***C*** entry specifies the centre of the cutting plane. This implies that the plane passes through the centre 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 centre. The orientation may be determined by a two-step process:

a) 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**.

b) 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]
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
         /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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

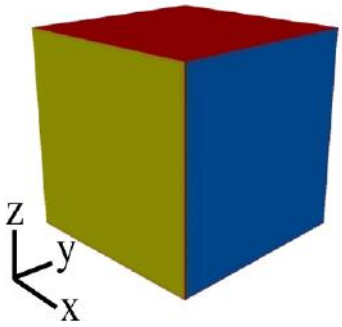


Figure 95 — Rendering of the 3D artwork using View0 (no cross section)

"Figure 95 — Rendering of the 3D artwork using View0 (no cross section)" through "Figure 99 — Rendering of the 3D artwork using View4" use world coordinates whose origin is the centre 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.

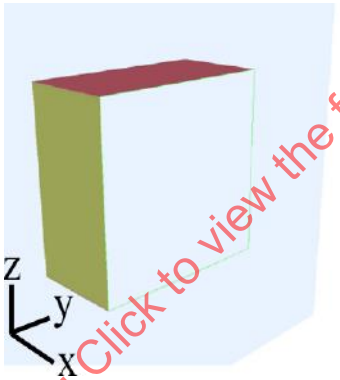


Figure 96 — Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)

"Figure 96 — Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)" 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]

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

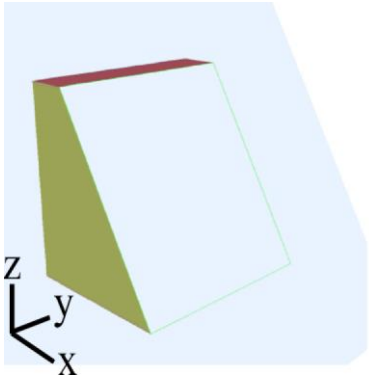


Figure 97 — Rendering of the 3D artwork using View2 (cross section rotated around the y axis by -30 degrees)

"Figure 97 — Rendering of the 3D artwork using View2 (cross section rotated around the y axis by -30 degrees)" 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 96 — Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)") 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]

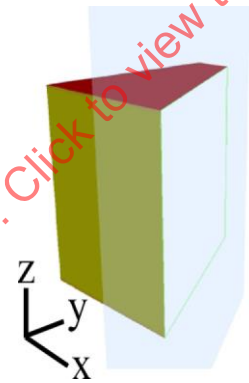


Figure 98 — Rendering of the 3D artwork using View3 (cross section rotated around the z axis by 30 degrees)

"Figure 98 — Rendering of the 3D artwork using View3 (cross section rotated around the z axis by 30 degrees)" 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 96 — Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)") 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]

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

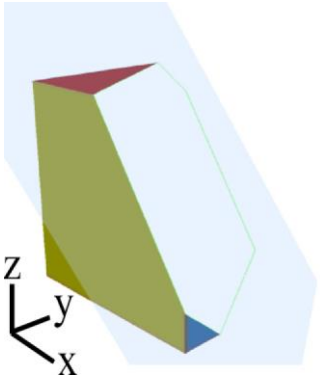


Figure 99 — Rendering of the 3D artwork using View4

(cross section rotated around the *y* axis by -30 degrees and around the *z* axis by 30 degrees)

"Figure 99 — Rendering of the 3D artwork using View4" 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 96 — Rendering of the 3D artwork using View1 (cross section perpendicular to the x axis)") 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 323 — Entries in a 3D node dictionary" shows the entries in a 3D node dictionary.

Table 323 — 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
N	text string	(*Required*) The name of the node being described by the node dictionary. All names in the node dictionary shall be unique. Interpretation of this entry shall depend upon the 3D format specified in the **Subtype** entry in "Table 311 — Entries in a 3D stream dictionary" as described below: *U3D* If the **Subtype** of the corresponding 3D Stream is *U3D*, this entry shall correspond to the field Node block name, specified in the Universal 3D file format. *PRC* (*PDF 2.0*) If the **Subtype** of the corresponding 3D Stream is *PRC*, this entry shall be the Unique Identifier (UUID) as specified in ISO 14739-1. NOTE 1 When comparing this entry to node names for a particular convention (such as Universal 3D or PRC), interactive PDF processors will need to translate between the PDF text encoding used by PDF and the character encoding specified in the 3D stream. NOTE 2 The description of the value of the **N** key was clarified in this document (2020).
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 is 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").
Instance	dictionary	(*Required if data are present; PDF 2.0*) An indirect object reference to a **RichMediaInstance** dictionary (see "Table 343 — Entries in a RichMediaInstance dictionary") that is also referenced by an entry in the Instances array.
Data	text string or stream	(*Optional; PDF 2.0*) A text string or stream that contains state data to be passed to the instance when the view is triggered. See the extended description of the **Data** key in "Table 345 — Entries in a View Params dictionary".
RM	dictionary	(*Optional; PDF 2.0*) A render mode dictionary that shall specify the render mode and related properties for this node. If omitted, the render mode specified in the 3D view is used, and if that is not present, the render mode of the 3D artwork is used. The render mode dictionary shall be identical to that used by the 3D view dictionary. See "Table 318 — Entries in a render mode dictionary".

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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

rendered in the same way as a non-visible node, having a separate value for the visibility of a node allows interactive PDF processors 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.

The **Instance** and **Data** entries provide a reference to an instance array dictionary within the **Instances** array. The content of **Data** shall be passed to and from the appropriate media run time engine by the interactive PDF processor in a rich media context.

The **RM** provides the facility of specifying a change in render mode for each node. In PDF 1.7, render mode could be specified globally or per view. See 13.6.4, "3D views".

The **RM** entry to the 3D node dictionary is applicable both for the 3D node structure within a rich media context (see 13.7.2, "RichMedia annotations") and within the existing 3D annotations structure (see 13.6.2, "3D Annotations"). The **Instances** array and additional **Data** entries are applicable only to 3D nodes within a rich media context.

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
      …
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
   >>
endobj

6 0 obj                              %View2, using the alternative set of node params
   <<
      /Type /3DView
      /XN (View2)
      /NA 4 0 R
      …
   >>
endobj
```

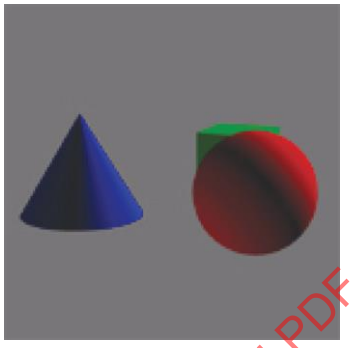


Figure 100 — Rendering of the 3D artwork using View1 (all shapes visible and opaque)

"Figure 100 — Rendering of the 3D artwork using View1 (all shapes visible and opaque)" 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).

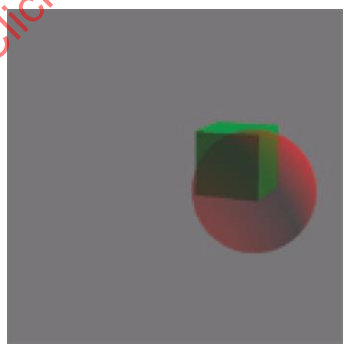


Figure 101 — Rendering of the 3D artwork using View2 (the cone is hidden and the sphere is semi-transparent)

"Figure 101 — Rendering of the 3D artwork using View2 (the cone is hidden and the sphere is semi-transparent)" shows a view with a node array that specifies the same three nodes used in "Figure 100 — Rendering of the 3D artwork using View1 (all shapes visible and opaque)". 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)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 subclause 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:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ tx & ty & 1 \end{bmatrix} = [a \quad b \quad c \quad d \quad tx \quad ty]$$

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:

$$\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ tx & ty & tz & 1 \end{bmatrix} = [a \quad b \quad c \quad d \quad e \quad f \quad g \quad h \quad i \quad tx \quad ty \quad tz]$$

3D coordinate transformations shall be expressed as matrix transformations:

$$[x' \quad y' \quad z' \quad 1] = [x \quad y \quad z \quad 1] \times \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

Carrying out the multiplication has the following results:

$$x' = a \times x + d \times y + g \times z + tx$$
$$y' = b \times x + e \times y + h \times + ty$$
$$z' = c \times x + f \times y + i \times z + tz$$

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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:

Artwork coordinates shall be transformed to world coordinates:

$$[x_w\ y_w\ z_w\ 1] = [x_a\ y_a\ z_a\ 1] \times \mathrm{aw}$$

World coordinates shall be transformed to camera coordinates:

$$[x_c\ y_c\ z_c\ 1] = [x_w\ y_w\ z_w\ 1] \times (\mathrm{cw}^{-1})$$

The first two steps can be expressed as a single equation, as follows:

$$[x_c\ y_c\ z_c\ 1] = [x_a\ y_a\ z_a\ 1] \times (\mathrm{aw} \times \mathrm{cw}^{-1})$$

Finally, the camera coordinates shall be projected into two dimensions, eliminating the *z* coordinate, then scaled and positioned to make the scene contents fit within the annotation's target coordinate system.

NOTE The above paragraph was clarified in this document (2020).

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 173 — Additional entries in markup annotation dictionaries specific to external data") that specifies the 3D annotation and view. "Table 324 — Entries in an external data dictionary used to markup 3D annotations" describes the entries in an external data dictionary used to markup 3D annotations.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 324 — 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; 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*) 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. NOTE This is strictly a checksum, and is not used for security purposes.

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 interactive PDF processors 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 interactive PDF processor, but a warning shall be issued to the user.

EXAMPLE The following example shows how markup annotations can be associated with particular views. This example was corrected in this document (2020).

```
2 0 obj                                  %3D stream data with two named views
      <<
         /Type /3D
         /Subtype /U3D
         /VA [4 0 R 5 0 R]
         …
      >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 /ExData
         /Subtype /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 /ExData
               /Subtype /Markup3D
               /3DA 3 0 R
               /3DV 5 0 R
            >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            …
        >>
endobj

9 0 obj                                 %Stamp comment on CommentView2
        <<
            /Type /Annot
            /Subtype /Stamp
            /ExData
               <<
                  /Type /ExData
                  /Subtype /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.

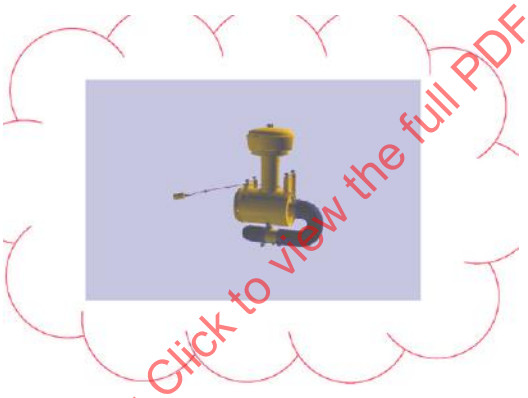


Figure 102 — 3D artwork set to its default view

"Figure 102 — 3D artwork set to its default view" shows the default view, which has no markup annotations.

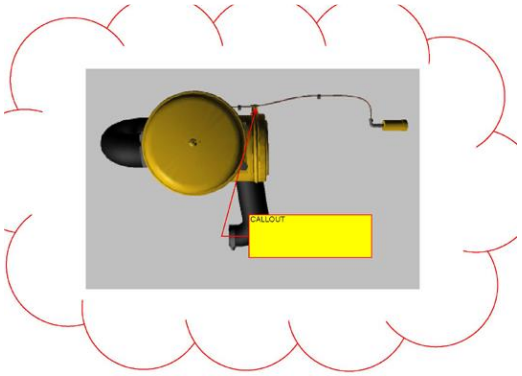


Figure 103 — 3D artwork set to CommentView1

"Figure 103 — 3D artwork set to CommentView1" shows another view to which a markup annotation is applied.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

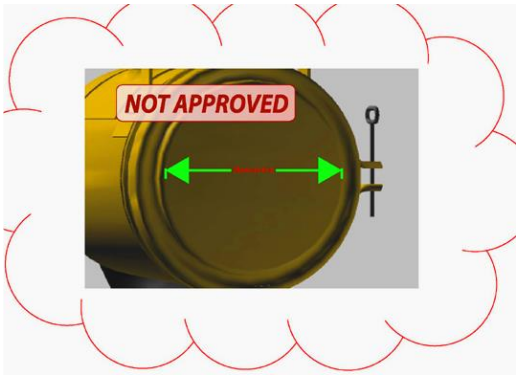


Figure 104 — 3D artwork set to CommentView2

"Figure 104 — 3D artwork set to CommentView2" 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)

13.6.7 Persistence of 3D measurements and markups

13.6.7.1 General

Beginning with PDF 2.0, it is possible to add *3D measurement data* to an instance of a 3D artwork. These measurement data are stored in a *3D measurement/markup dictionary*. 3D measurements are associated with 3D views, and each 3D view is able to contain zero or more 3D measurement dictionaries.

After a measurement is associated with view, it shall be visible only when that view is selected or active in an interactive PDF processor. As different views are displayed, the measurements associated with that view shall be made visible and previously displayed measurements become invisible.

3D measurements are either simple 3D markups used to add information to the geometric data shown in a view, or have an associated comment, in which case they have all the associated functionality of a comment. A 3D measurement is able to be promoted or demoted to or from comment status.

There are three key aspects to defining persistent 3D measurements:

- A mechanism to define the units associated with the geometric data being measured. A **3DU** entry in "Table 309 — Additional entries specific to a 3D annotation", has as its value a *3D units dictionary*, which stores the units data for this 3D annotation. (See "Table 325 — Entries in a 3D units dictionary" for more information.)
- The association between 3D measurements and 3D view is realised through the **MA** entry in a 3D view dictionary (see "Table 315 — Entries in a 3D view dictionary"). The value of this entry is an array of *3D measurement/markup dictionaries*, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. For more information about 3D measurement dictionaries, see 13.6.7.3, "3D measurement/markup dictionary".
- When a 3D measurement is promoted to a comment, a *projection annotation* ("see 12.5.6.24, "Projection annotations") is created to manage the comment and its appearance in the comments list. An indirect reference to this projection annotation is placed in the 3D measurement dictionary. (See 13.6.7.3, "3D measurement/markup dictionary".)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

13.6.7.2 The 3D units dictionary

The data associated with a 3D artwork annotation may be defined in an *arbitrary 3D coordinate system*. For viewing purposes, the interactive PDF processor shall define a camera to map these coordinates onto a view surface. (3D coordinate systems are discussed in 13.6.5, "Coordinate systems for 3D".) For measurement purposes, distances are computed in this arbitrary coordinate system and assigned physical meaning by entries in the 3D units dictionary. These sets of optional units shall be defined:

- *Creation time units*: Units known at the time the 3D artwork annotation is created.
- *User override units*: Units defined by the user after the annotation was created.
- *Display units*: Units that the user would like used when displaying distances for all newly created measurements.

The first two definitions assign physical meaning to measured distances, and the third defines how the distances are presented. When a 3D annotation is created, the application may have information from external sources that allows it to determine the units of the data being imported. Later the user may want to either override that definition or control what units data are displayed in.

In addition to defining the units, a scaling operation shall be defined that maps one set of units to another. For the creation time and user override units, the mapping states that "m model data units = n real units". For the display units, the mapping states that "m model units = n display units". In most cases, the m and n scale values shall be 1.0 because most data are defined in some well-known units system, such as meters or inches.

The entries in the *3D units dictionary* establish these mappings for each set of units (creation time, user override, and display units). The 3D Units dictionary is referenced in "Table 309 — Additional entries specific to a 3D annotation" as the value of the **3DU** entry.

Table 325 — Entries in a 3D units dictionary

Key	Type	Value
TSm	number	(*Optional; PDF 2.0*) The creation time units m scale value. If omitted, **TSm** defaults to 1.0; if included, **TU** shall exist.
TSn	number	(*Optional; PDF 2.0*) The creation time units n scale value. If omitted, **TSn** defaults to 1.0; if included, **TU** shall exist.
TU	text string	(*Optional; PDF 2.0*) The creation time units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. NOTE 1 It is recommended that the label use a universally recognised abbreviation.
USm	number	(*Optional; PDF 2.0*) The user defined units m scale value. If omitted, **USm** defaults to 1.0; if included, **UU** shall exist.
USn	number	(*Optional; PDF 2.0*) The user defined units n scale value. If omitted, **USn** defaults to 1.0; if included, **UU** shall exist.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
UU	text string	(*Optional; PDF 2.0*) The user override units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. NOTE 2 It is recommended that the label use a universally recognised abbreviation.
DSm	number	(*Optional; PDF 2.0*) The display units m scale value. If omitted, **DSm** defaults to 1.0; if included, **DU** shall exist.
DSn	number	(*Optional; PDF 2.0*) The display units n scale value. If omitted, **DSn** defaults to 1.0; if included, **DU** shall exist.
DU	text string	(*Optional; PDF 2.0*) The display units value. A text string specifying a label for displaying the units represented by this dictionary in a user interface. NOTE 3 It is recommended that the label use a universally recognised abbreviation.

The following algorithm shall be used to map model space distances for display in 3D measurements for each of the three sets of units.

Starting from these default values:

n = 1.0 // a number

m = 1.0 // a number

Units = "Model Units" // a text string

In the algorithm that follows, an entry is defined if it is included in the 3D Units dictionary.

Creation time units – The following is the process creation time units definition:

If **TU** is defined, then Units = TU

If **TSm** is defined, then m = TSm

If **TSn** is defined, then n = TSn

If either **TSm** or **TSn** is defined, **TU** should be included in the 3D Units dictionary; if **TU** is not defined in this case, the unit specification is undefined and shall be ignored.

User override units – The following is the process user override units definition:

If **UU** is defined, then Units = UU

If **USm** is defined, then m = USm

If **USn** is defined, then n = USn

If either **USm** or **USn** is defined, **UU** should be included in the 3D Units dictionary; if **UU** is not defined in this case, the unit specification is undefined and is ignored.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Display units – The following is the display units definition:

If **DU** is defined, then Units = DU

If **DSm** is defined, then m = m × DSm

If **DSn** is defined, then n = n × DSn

If either **DSm** or **DSn** is defined, **DU** should be included in the 3D Units dictionary; if **DU** is not defined in this case, the unit specification is undefined and is ignored.

Finally, if X is a model space distance and Y is the displayed value, the functional relationship between X and Y is given by the equation Y Units = (m/n) × X.

13.6.7.3 3D measurement/markup dictionary

13.6.7.3.1 General

The **MA** entry of "Table 315 — Entries in a 3D view dictionary" contains an array of *3D measurement dictionaries*, where each dictionary represents an instance of a 3D measurement to be displayed in the context of this view. Each 3D measurement dictionary describes the type of the 3D measurement through the **Subtype** entry, as well as provides a name for the measurement as it may appear in the user interface of an colour PDF processor.

The entries of the 3D Measurement/Markup dictionary are described in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes".

Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes; if present, it shall be *3DMeasure* for a 3D measurement dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Subtype	name	(*Required; PDF 2.0*) A name specifying the measurement type for this measurement which shall be chosen from the following: *LD3* A *linear dimension measurement* is used to denote the distance between two arbitrary points on a 3D model. See 13.6.7.3.2, "3D linear dimension measurement" for a listing of additional entries in this dictionary. *PD3* A *perpendicular dimension measurement* is used to denote the perpendicular distance between two geometric entities (normally two lines or a point and a line). See "Table 328 — Additional entries in a 3D measurement/markup dictionary for a 3D perpendicular dimension measurement" for a listing of additional entries in this dictionary. *AD3* An *angular dimension measurement* is used to denote the angle between two linear entities. See "Table 329 — Additional entries in a 3D measurement/markup dictionary for a 3D angular dimension measurement" for a listing of additional entries in this dictionary. *RD3* A *radial dimension measurement* is used to define the radius or diameter of a circular 3D entity. See "Table 330 — Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement" for a listing of additional entries in this dictionary. *3DC* A *3D comment note* lets users connect a comment to a specific piece of geometry in the 3D model. See "Table 331 — Additional entries in a 3D measurement/markup dictionary for a 3D comment note" for a listing of additional entries in this dictionary.
TRL	text string	(*Optional; PDF 2.0*) A name string that may be associated with a measurement markup. If omitted, an interactive PDF processor may create one. Interactive PDF processors that do not provide a user interface for such elements should ignore this field.

The **TRL** field contains the current name for the measurement markup. Each measurement markup is assigned a name ("Measurement 1", "3D Comment 22", … and so on) when it is created. These names may be shown by an interactive PDF processor so that users can see what measurements are associated with what views. Users should be allowed to override the initially defined names.

13.6.7.3.2 3D linear dimension measurement

A *3D linear measurement* is a markup showing the distance between two arbitrary points on a 3D model. "Figure 105 — 3D linear measurement" shows an example.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

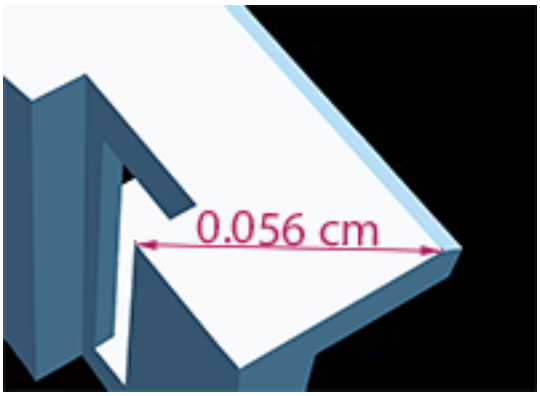


Figure 105 — 3D linear measurement

As shown, a *3D linear measurement* consists of two filled circles, called *anchor points*, one at each of the two positions being measured, and a line with an arrowhead on each end connecting the two anchor points (referred to as the measure line). This line is then labelled with a value representing the distance between the two anchor points.

In addition to the entries in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes", "Table 327 — Additional entries in a 3D measurement/markup dictionary for a 3D linear dimension measurement" lists additional entries for a 3D measurement dictionary with a **Subtype** value of *LD3* for 3D linear measurement.

Table 327 — Additional entries in a 3D measurement/markup dictionary for a 3D linear dimension measurement

Key	Type	Value
AP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup will lie.
A1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the first anchor point in world space. NOTE 1 It is assumed that this is a position on the 3D model associated with this view.
N1	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 1 (**A1**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
A2	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the second anchor point in world space. NOTE 2 It is assumed that this is a position on the 3D model associated with this view.
N2	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 2 (**A2**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
TP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the text anchor point for the measurement value string.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TY	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.
TS	number	(*Optional; PDF 2.0*) A number representing the measurement text string height defined in points in the default user space. The measurement text shall be zoom invariant. The default is 12 points.
C	array	(*Optional; PDF 2.0*) An array of three numbers in the range 0.0 to 1.0 that represent the *DeviceRGB* colour of the measurement markup. The default value is the array *[1 1 1]*, representing the colour white.
V	number	(*Required; PDF 2.0*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; PDF 2.0*) A text string, called the units string, which represents the units for the measurement.
P	integer	(*Optional; PDF 2.0*) The number of decimal digits, which represents the precision, shown for the measurement value (**V**). The default is 3, if **P** is not specified.
UT	text string	(*Optional; PDF 2.0*) A string defined by the user that is appended to the end of the measurement value string. If omitted, no string is appended.
S	dictionary	(*Optional; PDF 2.0*) A comment reference is an indirect reference to a projection annotation that may be associated with this 3D measurement. See 13.6.7.4, "3D measurements and projection annotations".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

"Figure 106 — Parameters for a 3D linear measurement" depicts some of the parameters for a 3D linear measurement.

The value text should be drawn on the annotation plane (**AP**) where the horizontal text path is defined by the vector from **A1** to **A2** with the text up direction defined by the vector **TY**. The lower left corner of the text box is positioned at the text anchor point (**TP**).

NOTE 1 A PDF viewer can choose to display the text aligned to the viewing plane, if it makes more sense in a particular use such as printing a legend.

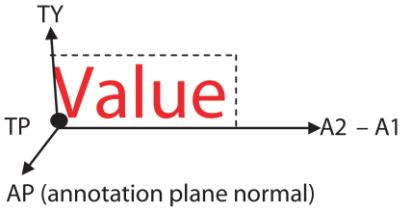


Figure 106 — Parameters for a 3D linear measurement

© ISO 2020 – All rights reserved

The actual Y-axis shall be formed by taking the cross product of **AP** and (**A2** - **A1**). The vector **TY** shall be used only to determine the orientation of this Y-axis.

If the text position **TP** is outside the area between **A1** and **A2**, an extension line collinear to the measure line connecting **TP** to the closest anchor point is generated.

There are three parts to the text string displayed with the measurement, a numeric value (**V**), a units string (**U**), and an optional user string (**UT**). The display of the numeric value field number is also controlled by the precision value (**P**), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

13.6.7.3.3 3D perpendicular dimension measurement

A *perpendicular measurement* is used to denote the perpendicular distance between two geometric entities (normally two lines or a point and a line) as illustrated here.

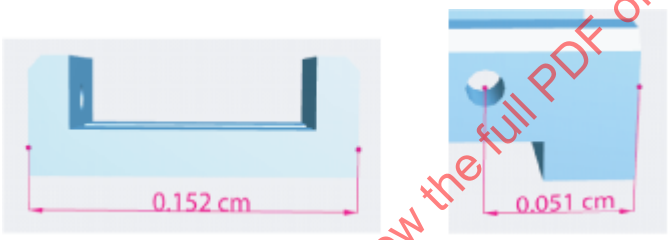


Figure 107 — Perpendicular measurement

"Figure 107 — Perpendicular measurement" shows a perpendicular measurement markup consists of two filled circles at the anchor points, two parallel extension lines (referred to as leader lines) starting at the anchor points and extending away from the anchor points. There is also a labelled line with arrowheads on both sides (referred to as the measure line) indicating that the distance shown is the perpendicular distance between the two parallel lines.

In addition to the entries in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes", the following entries are defined for a **I** with a **Subtype** value of *PD3* for 3D perpendicular measurement.

Table 328 — Additional entries in a 3D measurement/markup dictionary for a 3D perpendicular dimension measurement

Key	Type	Value
AP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup will lie.
A1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the first anchor point in world space. NOTE 1 It is assumed that this is a position on the 3D model associated with this view.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
N1	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 1 (**A1**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
A2	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the second anchor point in world space. NOTE 2 It is assumed that this is a position on the 3D model associated with this view.
N2	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 2 (**A2**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
D1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the direction vector for leader lines associated with the anchor points (**A1** and **A2**).
TP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TY	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.
TS	number	(*Optional; PDF 2.0*) A number representing the measurement text string height defined in points in the default user space. The measurement text shall be zoom invariant. The default is 12 points.
C	array	(*Optional; PDF 2.0*) An array of three numbers in the range 0.0 to 1.0, representing the **DeviceRGB** colour of the measurement markup. The default value is the array *[1 1 1]*, representing the colour white.
V	number	(*Required; PDF 2.0*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; PDF 2.0*) A string, called the units string, representing the units for the measurement.
P	integer	(*Optional; PDF 2.0*) The number of decimal digits, which represents the precision, shown for the measurement value (**V**). The default is *3*, if **P** is not specified.
UT	text string	(*Optional; PDF 2.0*) A string defined by the user that is appended to the end of the measurement value string. If omitted, no string is appended.
S	dictionary	(*Optional; PDF 2.0*) A comment reference is an indirect reference to a projection annotation that may be associated with this 3D measurement. See 13.6.7.4, "3D measurements and projection annotations".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

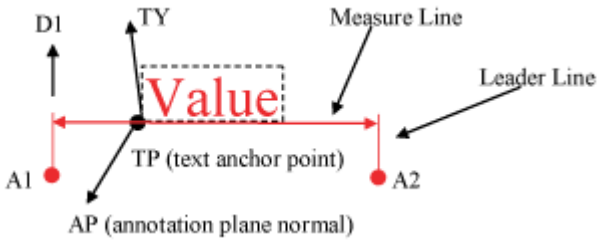


Figure 108 — Parameters associated with the perpendicular dimension

"Figure 108 — Parameters associated with the perpendicular dimension" illustrates the parameters associated with the perpendicular dimension and shows the measure markup and parameters. All the markup items are drawn on the annotation plane (as defined by **AP**). The text layout is defined in a similar manner as for linear dimensions. The lower-left corner of the text box is positioned at the text anchor point (**TP**), and the text's X-axis is aligned with the measure line. The text will flow in the direction defined by a vector from **A1** to **A2**. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the **TY** parameter.

In addition to controlling text position, the text anchor point (**TP**) also controls the lengths of the leader lines and the placement of the measure line. Because the leader lines are parallel and the measure line is required to be perpendicular to both leader lines, the intersection of the leader lines and the measure line is easily computed.

If the text position **TP** is outside the area between **A1** and **A2**, an extension line collinear to the measure line connecting **TP** to the closest anchor point is generated.

There are three parts to the text string displayed with the measurement: a numeric value (**V**), a units string (**U**), and an optional user string (**UT**). The display of the numeric value field number is also controlled by the precision value (**P**), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

13.6.7.3.4 3D angular dimension measurement

An angular measurement is used to denote the angle between two linear entities, as shown in "Figure 109 — 3D Angular Dimension Measurement".

Figure 109 — 3D Angular Dimension Measurement

"Figure 109 — 3D Angular Dimension Measurement" shows an angular measurement markup consisting of two anchor points, one located on each of the two linear entities whose angle is being measured. Connected to each anchor point is an extension line that is collinear with the edge it

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

measures. A labelled arc, with an arrowhead at each end, connects the two extension lines, making it clear which angle is being measured.

In addition to the entries in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes", "Table 329 — Additional entries in a 3D measurement/markup dictionary for a 3D angular dimension measurement" lists entries defined for a 3D measurement dictionary with a **Subtype** value of *AD3* for 3D angular measurement.

Table 329 — Additional entries in a 3D measurement/markup dictionary for a 3D angular dimension measurement

Key	Type	Value
AP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup lies.
A1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the first anchor point in world space. NOTE 1 It is assumed that this is a position on the 3D model associated with this view.
D1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the direction vector for the leader line associated with the first anchor point (**A1**).
N1	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 1 (**A1**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
A2	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the second anchor point in world space. NOTE 2 It is assumed that this is a position on the 3D model associated with this view.
D2	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the direction vector for the leader line associated with the second anchor point (**A2**).
N2	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 2 (**A2**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
TP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TX	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the horizontal direction vector for the text string presenting the measurement value string.
TY	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the up direction vector, called the text Y direction, for the text string presenting the measurement value string.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TS	number	(*Optional; PDF 2.0*) A number representing the measurement text string height defined in points in the default user space. The measurement text shall be zoom invariant. The default is 12 points.
C	array	(*Optional; PDF 2.0*) An array of three numbers in the range 0.0 to 1.0, representing the DeviceRGB colour of the measurement markup. The default value is the array *[1 1 1]*, representing the colour white.
V	number	(*Required; PDF 2.0*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
P	integer	(*Optional; PDF 2.0*) The number of decimal digits, which represents the precision, shown for the measurement value (**V**). The default is *3*, if **P** is not specified.
UT	text string	(*Optional; PDF 2.0*) A text string defined by the user that is appended to the end of measurement value string. If omitted, no string is appended.
DR	boolean	(*Optional; PDF 2.0*) A flag that indicates whether degrees or radians are shown in angular measurements. If **DR** is *true*, angular measurements are shown in degrees. The default is *true*.
S	dictionary	(*Optional; PDF 2.0*) A comment reference is an indirect reference to a projection annotation that may be associated with this 3D measurement. See 13.6.7.4, "3D measurements and projection annotations".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The key geometric parameters for an angular dimension are shown in "Figure 110 — Geometric parameters for an angular dimension".

Figure 110 — Geometric parameters for an angular dimension

The angle is defined by the measurement value **V** (30 in "Figure 110 — Geometric parameters for an angular dimension") and is the angle between the leader direction vectors (**D1** and **D2**). The angular measurement markup is generated by first computing the centre point of the angle, cp in this figure.

The text position (**TP**) controls the position of the measurement text, the placement of the angle arc, and the length and direction of the extension lines. The extension lines are drawn from the anchor point to a point at a distance ||TP-cp|| from the centre point cp along the associated direction vector,

© ISO 2020 – All rights reserved

which is the intersection of the angle arc and the extension line. The angle arc centre is at the centre point cp (and its radius is ||TP - cp||) and is drawn between the two extensions lines. The markup text is displayed (based on the text orientation parameters) with the lower-left corner of the text string starting at the text position (**TP**).

The text layout is defined in a similar manner as for other dimensions. The lower-left corner of the text box is positioned at the text anchor point (**TP**), and the text's X-axis is defined by the vector **TX**. Note that the vector **TX** is expected to be orthogonal with the annotation plane normal. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the **TY** parameter.

The measurement value is interpreted as either being in degrees or radians as defined by the (**DR**) value, and the appropriate label string is created.

There are three parts to the text string displayed with the measurement: a numeric value (**V**), a degrees or radians string (**U**), and an optional user string (**UT**). The display of the numeric value field number is also controlled by the precision value (**P**), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the degrees or radians string and user text as appropriate. This process is viewer dependent.

There are some special cases:

- Parallel direction vectors (**D1** and **D2**) are invalid, and no markup is generated.
- If the text position **TP** is outside the cone of the angle, an extension line is added to connect the text with the angle arc.

13.6.7.3.5 3D radial dimension

The radial measurement is used to define the radius or diameter of a circular 3D entity. "Figure 111 — 3D Radial Dimension" illustrates two examples of a radial dimension.

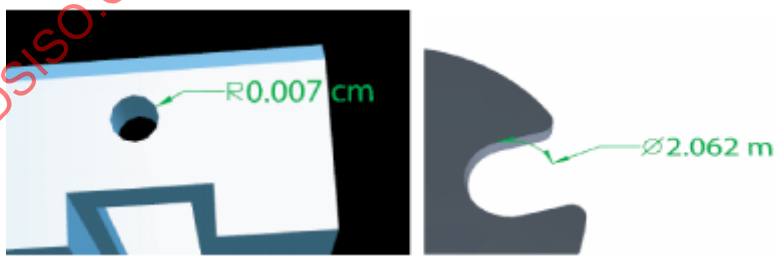


Figure 111 — 3D Radial Dimension

As shown on the left, the basic markup for a radial dimension consists of an arrow pointing to a circle or arc that is connected to a leader line and text label that defines the radius or diameter. If the arrow is positioned such that it is off the underlying arc, as in the figure on the right, an extension arc is generated that clarifies which arc is being measured.

For radius measurements, the measure value is preceded by an "R" in the measure string. For diameter values, the measure value is preceded by the diameter symbol (⌀).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

In addition to the entries in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes", "Table 330 — Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement" lists entries defined for a 3D measurement dictionary with a **Subtype** value of *RD3* for 3D radial measurement.

Table 330 — Additional entries in a 3D measurement/markup dictionary for a 3D radial dimension measurement

Key	Type	Value
AP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the 3D annotation plane on which the measurement markup lies.
A1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the first anchor point, called the circle centre point, in world space. NOTE 1 It is assumed that this is a position on the 3D model associated with this view.
A2	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the second anchor point, which is a point on the arc, in world space. NOTE 2 It is assumed that this is a position on the 3D model associated with this view.
N2	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 2 (**A2**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.
A3	array	(*Required for arcs only; PDF 2.0*) A three-element array of numbers that defines the start of the arc being measured. This entry is required if the geometry being measured is an arc. See "Figure 113 — Defining a radial dimension for an arc" for a visual representation of **A3**.
A4	array	(*Required for arcs only; PDF 2.0*) A three-element array of numbers that defines the end of the arc being measured. This entry is required if the geometry being measured is an arc. See "Figure 113 — Defining a radial dimension for an arc" for a visual representation of **A4**.
TP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TX	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the horizontal direction vector for the text string presenting the measurement value string.
TY	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the up direction vector for the text string presenting the measurement value string.
EL	number	(*Optional; PDF 2.0*) The length of the extension line in points. The default is 60 points.
TS	number	(*Optional; PDF 2.0*) A number representing the measurement text string height defined in points in the default user space. The measurement text shall be zoom invariant. The default is 12 points.
C	array	(*Optional; PDF 2.0*) An array of three numbers in the range 0.0 to 1.0, representing the **DeviceRGB** colour of the measurement markup. The default value is the array *[1 1 1]*, representing the colour white.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
V	number	(*Required; PDF 2.0*) A numeric value representing a measurement value. This value is converted to a text string and displayed as part of the measurement text string.
U	text string	(*Required; PDF 2.0*) A string representing the units for the measurement.
P	integer	(*Optional; PDF 2.0*) The number of decimal digits, which represents the precision, shown for the measurement value (**V**). The default is *3*, if **P** is not specified.
UT	text string	(*Optional; PDF 2.0*) A string defined by the user that is appended to the end of measurement value string. If omitted, no string is appended.
SC	boolean	(*Optional; PDF 2.0*) A flag that indicates whether the underlying circle, or arc, is shown or not. If *true*, the underlying circle associated with a radial dimension is displayed. The circle or arc shall be redrawn in the markup colour (**C**). The default is *false*, not to show the circle.
R	boolean	(*Optional; PDF 2.0*) A flag that indicates whether the measurement value is for a radius or a diameter. If *true*, the measurement value associated with a radial measurement represents a radius, as opposed to a diameter value. The default is *true*, the measurement value represents a radius.
S	dictionary	(*Optional; PDF 2.0*) A comment reference is an indirect reference to a projection annotation that may be associated with this 3D measurement. See 13.6.7.4, "3D measurements and projection annotations".

The parameters for defining the radial measurement for a circle are shown in "Figure 112 — Defining the radial measurement for a circle".

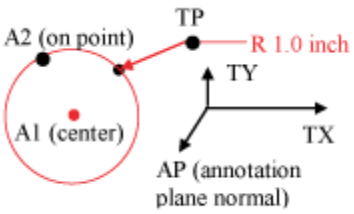


Figure 112 — Defining the radial measurement for a circle

The circle being measured is defined by two points lying on the annotation plane, a circle centre **A1**, and a point on the circle **A2**. For radial measurement, the text position (**TP**) controls the text string position, the arrow line orientation, and the extension line. The arrow line is drawn from the text position (**TP**) to the intersection point between the circle and a line from text position to the circle centre. The extension line will be drawn from the text position (**TP**) in the direction of the text string's X-axis (**TX**). The length of the extension line is defined by the **EL** parameter. The left centre of measure value string will begin at the end of the extension line. Note that the vector **TX** is expected to be orthogonal with the annotation plane normal. The text's up direction is defined as the cross product of the annotation plane normal and the text X-axis, in the direction defined by the **TY** parameter.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

The parameters for defining a radial dimension for an arc are very similar, as shown in "Figure 113 — Defining a radial dimension for an arc".

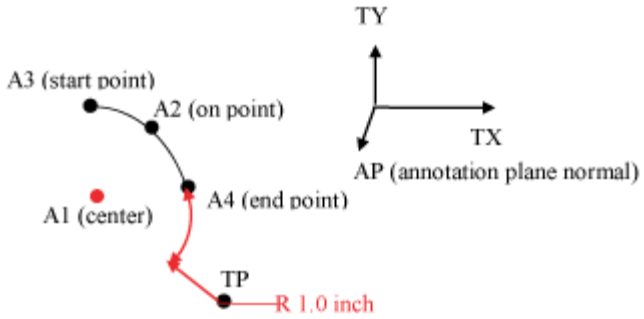


Figure 113 — Defining a radial dimension for an arc

There are three parts to the text string displayed with the measurement: a numeric value (**V**), a units string (**U**), and an optional user string (**UT**). The display of the numeric value field number is also controlled by the precision value (**P**), which indicates how many digits to display to the right of the decimal point. The viewer should convert the numeric value to a string and combine it with the units string and user text as appropriate. This process is viewer dependent.

13.6.7.3.6 3D comment note

3D comment notes let users connect a comment to a specific piece of geometry in the 3D model. The markup consists of a leader line that connects the model to a text box placed in the 3D scene. The text box is rendered so that the text is always facing the user.

Commenting functionality is specified by creating a projection annotation that represents the 3D measurement/ markup within the commenting system. See 13.6.7.4, "3D measurements and projection annotations" for additional details.

In addition to the entries in "Table 326 — Entries in a 3D measurement/markup dictionary common to all markup subtypes", "Table 331 — Additional entries in a 3D measurement/markup dictionary for a 3D comment note" lists entries defined for a 3D measurement dictionary with a **Subtype** value of **3DC** for 3D comment notes.

Table 331 — Additional entries in a 3D measurement/markup dictionary for a 3D comment note

Key	Type	Value
A1	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the model space position of the first anchor point in world space. NOTE It is assumed that this is a position on the 3D model associated with this view.
N1	text string	(*Optional; PDF 2.0*) The name of the part (or model tree node) associated with anchor point 1 (**A1**). The part name is used to verify that the part exists and is visible. If not, the measurement is not displayed. If omitted, no validation occurs.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TP	array	(*Required; PDF 2.0*) A three-element array of numbers specifying the text anchor point for the measurement value string.
TB	array	(*Optional; PDF 2.0*) A two-element integer array defining the *x* and *y* size of the text box to contain the user text string (**UT**). The *x* value is defined as a number of characters shown in the *x* direction (the top row); the *y* value is the number of rows of text. The interactive PDF processor is free to flow the text in the text box as appropriate. If the entire string does not fit, it may be truncated. The interactive PDF processor may also choose any font in which to render the content, though a monospaced font is recommended.
TS	number	(*Optional; PDF 2.0*) The text size – a number representing the measurement text string height defined in points in the default user space. The measurement text shall be zoom invariant. Default value: *12*.
C	array	(*Optional; PDF 2.0*) An array of three numbers in the range 0.0 to 1.0, representing the **DeviceRGB** colour of the measurement markup. The default value is the array *[1 1 1]*, representing the colour white.
UT	text string	(*Optional; PDF 2.0*) The text string displayed as the contents of the 3D comment note.
S	dictionary	(*Optional; PDF 2.0*) A comment reference is an indirect reference to a projection annotation that may be associated with this 3D measurement. See "Table 341 — Entries in a RichMediaContent dictionary".

The first anchor point (**A1**) defines the connection to the model. The model space text position **TP** field defines placement of the corner of the text box that lies closest to anchor point 1 (**A1**) in the current view. The user string field contains the text to be fitted into the text box.

13.6.7.4 3D measurements and projection annotations

Associations between a 3D measurement and a projection annotation are created by defining an indirect reference to a dictionary in both the projection annotation and the 3D measurement.

- Projection annotation: When the projection annotation (see 12.5.6.24, "Projection annotations") is created, the **ExData** entry (an external data dictionary) is defined and used to manage the association between the 3D measurement and the comment. "Table 332 — Entries in the external data dictionary of a projection annotation" describes the entries in the external data dictionary of the projection annotation that are used for referencing a 3D measurement.
- 3D measurement: The value of the **S** entry in the 3D measurement dictionary is an indirect reference to a projection annotation dictionary of the associated comment.

Table 332 — Entries in the external data dictionary of a projection annotation

Key	Type	Value
Type	name	(*Required; PDF 2.0*) The type of PDF object that this dictionary describes; if present, shall be *ExData* for an external data dictionary.
Subtype	name	(*Required; PDF 2.0*) The type of external data that this dictionary describes; shall be *3DM* for an association to a 3D measurement.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
M3DREF	dictionary	(*Required; PDF 2.0*) An indirect reference to a 3D measurement dictionary for which this projection annotation is a comment. See 12.5.6.24, "Projection annotations".

13.7 Rich media

13.7.1 General

PDF 2.0 introduces *rich media annotations* providing a common framework for video, audio, animations and other multimedia presentations. For the allowed instances of rich media see "Table 342 — Entries in a RichMediaConfiguration dictionary". The framework includes general controls most of which are commonly needed to stage, present and control an animation, video or other interactive presentation. Such features as: when to activate and deactivate the presentation, supplying scripts that control the presentation, setting viewing parameters such as lighting and camera angles (e.g., for 3D interactive presentations), setting animation parameters such as speed and looping behaviour, supplying assets including the base video, audio and animations files as well as any necessary auxiliary files (e.g., scripts), determining how the rich media annotation relates to the page display (e.g., within a fixed rectangle on the page, or having its own window), and details determining configuration of toolbars, navigation and other controls.

Not all of these common framework features are appropriate for all rich media types and selected media types have additional features defined specific to that type. The details for the framework and specific controls are given in the following subclauses.

13.7.2 RichMedia annotations

13.7.2.1 General

The annotation subtype *RichMedia* shares many low-level structural similarities with the 3D Artwork defined in 13.6, "3D Artwork". At the top level, the *rich media annotation* has two primary custom structures. The *RichMediaSettings dictionary* is unique to each annotation, while the *RichMediaContent dictionary* can be shared across rich-media annotations.

The rich media annotation may contain any of the entries of an annotation dictionary (see 12.5.2, "Annotation dictionaries"). "Table 333 — Additional entries specific to a RichMedia annotation" shows the additional annotation entries specific to this type of annotation.

Table 333 — Additional entries specific to a RichMedia annotation

Key	Type	Value
Subtype	name	(*Required; PDF 2.0*) The type of annotation that the dictionary describes. The name shall be *RichMedia* for a rich media annotation.
RichMediaContent	dictionary	(*Required; PDF 2.0*) A RichMediaContent dictionary that stores the rich media artwork and information as to how it should be configured and viewed. See "Table 341 — Entries in a RichMediaContent dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
RichMediaSettings	dictionary	(*Optional; PDF 2.0*) A RichMediaSettings dictionary that stores conditions and responses that determine when the annotation should be activated and deactivated by an interactive PDF processor and the initial state of artwork in those states. See "Table 334 — Entries in a RichMediaSettings dictionary". Default value: If no RichMediaSettings dictionary is present, the first configuration is loaded.

In addition to these entries, a RichMedia annotation shall provide an appearance stream in its **AP** entry (see "Table 166 — Entries common to all annotation dictionaries ") that has a normal appearance (the **N** entry in "Table 170 — Entries in an appearance dictionary"). This appearance may be used by applications that do not support RichMedia annotations and by all applications for the initial display of the annotation

As with 3D annotations defined in 13.6.2, "3D Annotations", RichMedia annotations can be in one of two states:

- *Inactive*: (the default initial state) The annotation displays the normal appearance of the annotation.

NOTE Typically, the normal appearance is a 2D bitmap portraying a rendering of the default view of the artwork within the **RichMedia** annotation.

- *Active*: The annotation displays a rendering of the artwork. This rendering is specified by the annotation's RichMediaSettings entry. (See "Table 334 — Entries in a RichMediaSettings dictionary".)

The subsequent clauses describe the structural components of a rich media annotation, beginning with the RichMediaSettings dictionary and its subcomponents, followed by the RichMediaContent dictionary, and its subcomponents. See Example 3 for a detailed and comprehensive example of a rich media annotation.

13.7.2.2 RichMediaSettings dictionary

13.7.2.2.1 General

The *RichMediaSettings dictionary* has a purpose similar to that of the 3D activation dictionary ("Table 310 — Entries in a 3D activation dictionary") with the 3D Annotation described in 13.6.2, "3D Annotations". The RichMediaSettings dictionary stores the conditions and responses that occur in response to certain events, such as activation and deactivation of the annotation, and contains two dictionaries. "Table 334 — Entries in a RichMediaSettings dictionary" gives the details of the content of this dictionary.

Table 334 — Entries in a RichMediaSettings dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, shall be *RichMediaSettings*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Activation	dictionary	(*Optional; PDF 2.0*) A RichMediaActivation dictionary (see "Table 335 — Entries in a RichMediaActivation dictionary") that specifies the style of presentation, default script behavior, default view information, and animation style when the annotation is activated.
Deactivation	dictionary	(*Optional; PDF 2.0*) A RichMediaDeactivation dictionary (see "Table 336 — Entries in a RichMediaDeactivation dictionary") that specifies the condition and type of unloading (restart or pause) that occurs during deactivation.

13.7.2.2.2 RichMediaActivation dictionary

The *RichMediaActivation dictionary* specifies the style of presentation, default script behavior, default view information, and animation style when the annotation is activated. "Table 335 — Entries in a RichMediaActivation dictionary" details the contents of the dictionary.

The activation dictionary includes a *script array*. Each element contains an indirect object reference to file specification objects that are also referenced by the **Assets** name tree of the RichMediaContent dictionary. When the annotation is activated, each script in the array shall be executed in order in a common environment per annotation by the rich media run time ECMAScript interpreter.

There may be an optional reference to a view in the scene's **Views** array (see "Table 341 — Entries in a RichMediaContent dictionary") that acts as the default view for the annotation. If not specified, the first view in the views array shall be used. If there are no views specified, default values shall be used for all camera and display parameters.

Table 335 — Entries in a RichMediaActivation dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, shall be *RichMediaActivation*.
Condition	name	(*Optional; PDF 2.0*) A name that shall specify the circumstances under which the annotation shall be activated. The following values are valid: *XA* the annotation is explicitly activated by a user action or script. *PO* the annotation is activated as soon as the page that contains the annotation receives focus as the current page. *PV* the annotation is activated as soon as any part of the page that contains the annotation becomes visible. NOTE One example is in a multiple-page presentation. Only one page is the current page although several are visible. Default value: *XA*.
Animation	dictionary	(*Optional; PDF 2.0*) A *RichMediaAnimation dictionary* (see "Table 337 — Entries in a RichMediaAnimation dictionary") that describes the preferred method that interactive PDF processor should use to drive keyframe animations present in this artwork.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
View	dictionary	(*Optional; PDF 2.0*) An indirect object reference to a 3D view dictionary (see "Table 315 — Entries in a 3D view dictionary" that shall also be referenced by the **Views** array within the annotation's RichMediaContent dictionary (see "Table 341 — Entries in a RichMediaContent dictionary"). Default value: The first element in the **Views** array of the annotation specified in the RichMediaContent dictionary. If a Views array does not exist, default values for the components of a 3D view dictionary (see "Table 344 — Additional entries in a 3D view dictionary") are used.
Configuration	dictionary	(*Optional; PDF 2.0*) An indirect object reference to a RichMediaConfiguration dictionary (see "Table 342 — Entries in a RichMediaConfiguration dictionary") that shall also be referenced by the **Configurations** array in the RichMediaContent dictionary (see "Table 341 — Entries in a RichMediaContent dictionary"). Default value: The first element within the **Configurations** array specified in the RichMediaContent dictionary.
Presentation	dictionary	(*Optional; PDF 2.0*) A RichMediaPresentation dictionary (see "Table 338 — Entries in a RichMediaPresentation dictionary") that contains information as to how the annotation and user interface elements will be visually laid out and drawn.

13.7.2.2.3 RichMediaDeactivation dictionary

The second component of the RichMediaSettings dictionary is the *RichMediaDeactivation dictionary*, which specifies the condition that causes deactivation of the annotation.

Table 336 — Entries in a RichMediaDeactivation dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, shall be *RichMediaDeactivation*.
Condition	name	(*Optional; PDF 2.0*) A name specifying the circumstances under which the annotation shall be deactivated. The following values are valid: *XD* the annotation is explicitly deactivated by a user action or script. *PC* the annotation is deactivated as soon as the page that contains the annotation loses focus as the current page. *PI* the annotation is deactivated as soon as the entire page that contains the annotation is no longer visible. Default value: *XD*.

13.7.2.2.4 RichMediaAnimation dictionary

A *RichMediaAnimation dictionary* specifies the preferred method that interactive PDF processors should use to apply timeline scaling to keyframe animations. It can also specify that keyframe animations be played repeatedly. The **Animation** entry of the RichMediaActivation dictionary ("Table 335 — Entries in a RichMediaActivation dictionary") can specify a RichMediaAnimation dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

A *keyframe animation* can be provided as part of 3D model content. "Table 337 — Entries in a RichMediaAnimation dictionary" shows the entries in a RichMediaAnimation dictionary.

NOTE Keyframe animation is an interactive feature that is highly dependent on the behaviour and controls provided by the interactive PDF processor.

Table 337 — Entries in a RichMediaAnimation dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, the type shall be *RichMediaAnimation*.
Subtype	name	(*Optional; PDF 2.0*) The animation style described by this dictionary. Valid values are *None*, *Linear*, and *Oscillating*. See "Table 313 — Animation styles" for descriptions of these animation styles. If an animation style is encountered other than those described, an animation style of *None* shall be used. Default value: *None*
PlayCount	integer	(*Optional; PDF 2.0*) An integer specifying the play count for this animation style. A nonnegative integer shall represent the number of times the animation is played. A negative integer shall indicate that the animation is infinitely repeated. This value shall be ignored for an animation subtype of type *None*. Default value: *-1*
Speed	number	(*Optional; PDF 2.0*) A positive number specifying the speed that shall be used when running the animation. A value greater than one shall shorten the time it takes to play the animation, or effectively speed up the animation. NOTE This allows authors to change the desired speed of animations without re-authoring the content. This value shall be ignored for an animation subtype of type *None*. Default value: *1*

EXAMPLE A RichMediaSettings dictionary

```
20 0 obj                                    %RichMediaSettings dictionary
    <</Type /RichMediaSettings
      /Activation
      <</Type /RichMediaActivation          %RichMediaActivation Dictionary
        /Condition /XA
        /Configuration 14 0 R               %Reference to element in Configurations array
        /View 19 0 R                        %Reference to element in Views array
        /Animation                          %RichMediaAnimation dictionary
        <</Type /RichMediaAnimation
           /Subtype /Linear
           /Speed 1
           /PlayCount -1
        >>
        /Presentation 21 0 R
        /Scripts [32 0 R 33 0 R]            %References to scripts in Resources name tree
      >>
      /Deactivation                         %RichMediaDeactivation dictionary
      <</Type /RichMediaDeactivation
        /Condition /XD
      >>
  >>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
endobj
```

13.7.2.2.5 RichMediaPresentation dictionary

The *RichMediaPresentation dictionary* contains information about how the annotation and user interface elements should be visually laid out and drawn by an interactive PDF processor. The visibility of the Toolbar is specified, and it will only take effect if the **RichMediaConfiguration** subtype is *3D*. (See "Table 342 — Entries in a RichMediaConfiguration dictionary")

User interface items such as a navigation panes and toolbars can be displayed or hidden by default. Such user interface elements may only be implemented by those interactive PDF processors where it is appropriate to do so.

NOTE The navigation pane is a user interface item that would be used to display the hierarchical relationship of entities within the artwork. (See 13.6.2, "3D Annotations".)

The **Style** of the annotation can be presented *Embedded* within the PDF page or separately *Windowed*. If a *Windowed* state is chosen, the default dimensions and position of the window may be given. "Table 338 — Entries in a RichMediaPresentation dictionary" contains a description of each element.

Table 338 — Entries in a RichMediaPresentation dictionary

Key	Type	Value
Type	name	*(Optional; PDF 2.0)* The type of PDF object that this dictionary describes. If present, shall be *RichMediaPresentation*.
Style	name	*(Optional; PDF 2.0)* The style of presentation of the rich media. The value can be *Embedded* or *Windowed*. A interactive PDF processor shall support the state *Embedded*. Default value: *Embedded*
Window	dictionary	(*Optional; PDF 2.0*) A RichMediaWindow Dictionary that describes the size and position of the floating user interface window when the value for **Style** is set to *Windowed*. See "Table 339 — Entries in a RichMediaWindow dictionary" for a detailed description.
Transparent	boolean	(*Optional; PDF 2.0*) A flag that indicates whether the page content shall be displayed through the transparent areas of the rich media content (where the alpha value is less than 1.0). If *true*, the rich media artwork shall be composited over the page content using an alpha channel. If *false*, the rich media artwork shall be drawn over an opaque background prior to composition over the page content. Default value: *false*
NavigationPane	boolean	(*Optional; PDF 2.0*) A flag that indicates the default behavior of a navigation pane user interface element. If *true*, the navigation pane should be visible when the content is initially activated. If *false*, the navigation pane should not displayed by default. Default value: *false*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Toolbar	boolean	(*Optional; PDF 2.0*) A flag that indicates the default behavior of an interactive toolbar associated with this annotation. If *true*, a toolbar should be displayed when the annotation is activated and given focus. If *false*, a toolbar should not displayed by default. The toolbar should be positioned in proximity to the annotation. Default value: *true* for content of Subtype 3D, and *false* otherwise.
PassContextClick	boolean	(*Optional; PDF 2.0*) A flag that indicates whether a contextual click on the rich media annotation is passed to the media player run time or shall be handled natively by the interactive PDF processor. NOTE A context click is usually generated by a mouse right-click although it can be invoked by other means. This can include, but is not limited to, an explicit context menu keyboard key or the combination of a mouse click and a keyboard modifier key. If *false*, the interactive PDF processor shall handle the context click. If *true*, the interactive PDF processor shall allow the media runtime to provide any user interface it wishes. Default value: *false*

13.7.2.2.6 RichMediaWindow dictionary

The *RichMediaWindow dictionary* stores the dimensions and position of the floating window that an interactive PDF processor may present. It is used only if **Style** (see "Table 339 — Entries in a RichMediaWindow dictionary") is set to *Windowed*. The window described is a non-printing user interface element. An interactive PDF processor that supports *Windowed* styles shall constrain the window extent to be within the presentation area of the interactive PDF processor.

All coordinates shall be expressed in default user space units although do not rotate or scale the window to match the orientation and magnification of the page. The expected behavior of the interactive PDF processor is similar to how it presents an annotation when the flags NoZoom and NoRotate are set to true. (See 12.5.3, "Annotation flags".) "Table 339 — Entries in a RichMediaWindow dictionary" details the contents of this dictionary.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 339 — Entries in a RichMediaWindow dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, shall be *RichMediaWindow*.
Width	dictionary	(*Optional; PDF 2.0*) A dictionary with keys **Default**, **Max**, and **Min** describing values for the width of the Window in default user space units. Default values: **Default** shall be *288*, **Max** shall be *576*, and **Min** shall be *72*

© ISO 2020 – All rights reserved

Key	Type	Value
Height	dictionary	(*Optional; PDF 2.0*) A dictionary with keys **Default**, **Max**, and **Min** describing values for the width of the Window in default user space units. Default values: *Default* shall be *216*, *Max* shall be *432*, and *Min* shall be *72*
Position	dictionary	(*Optional; PDF 2.0*) A RichMediaPosition dictionary describing the position of the *RichMediaWindow*. See "Table 340 — Entries in a RichMediaPosition dictionary" for a detailed description.

The position of the window in the interactive PDF processor presentation area is described by the *RichMediaPosition* dictionary. The position of the window shall remain fixed, regardless of the page translation. "Table 340 — Entries in a RichMediaPosition dictionary" details the contents of this dictionary.

Table 340 — Entries in a RichMediaPosition dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, it shall be *RichMediaPosition*.
HAlign	name	(*Optional; PDF 2.0*) Describes the horizontal alignment. Valid values are *Near*, *Center*, and *Far*. Default value: *Far*
VAlign	name	(*Optional; PDF 2.0*) Describes the vertical alignment. Valid values are *Near*, *Center*, and *Far*. Default value: *Near*
HOffset	number	(*Optional; PDF 2.0*) The offset from the alignment point specified by the **HAlign** key. A positive value for **HOffset**, when **HAlign** is either *Near* or *Center*, shall offset the position towards the *Far* direction. A positive value for **HOffset**, when **HAlign** is *Far*, shall offset the position towards the *Near* direction. Default value: *18*
VOffset	number	(*Optional; PDF 2.0*) The offset from the alignment point specified by the **VAlign** key. A positive value for **VOffset**, when **VAlign** is either *Near* or *Center*, shall offset the position towards the *Far* direction. A positive value for **VOffset**, when **VAlign** is *Far*, shall offset the position towards the *Near* direction. Default value: *18*

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

The descriptions of the values of the **HAlign** and **VAlign** entries depend on the primary logical content order as defined by the value of the **Lang** entry in the document's catalog dictionary. For left-to-right reading languages, the descriptions follow.

HAlign: *Near* represents the left edge of the window, and *Far* represents the right edge of the window.

VAlign: *Near* represents the top edge of the window, and *Far* represents the bottom edge of the window.

© ISO 2020 – All rights reserved

For right-to-left reading languages, the above descriptions for **HAlign** are reversed.

EXAMPLE A RichMediaPresentation Dictionary

```
24 0 obj                                %RichMediaPresentation dictionary
   <</Type /RichMediaPresentation
      /Toolbar false
      /NavigationPane false
      /PassContextClick false
      /Style /Windowed
      /Window 25 0 R
   >>
endobj

25 0 obj                                %RichMediaWindow dictionary
    <</Type /RichMediaWindow
      /Height
      <</Default 216
         /Max 432
         /Min 72
      >>
      /Width
      <</Default 288
         /Max 576
         /Min 72
      >>
      /Position
      <</Type /RichMediaPosition        %RichMediaPosition dictionary
         /HAlign /Far
         /VAlign /Near
         /HOffset 18
         /VOffset 18
      >>
   >>
endobj
```

In the above example, the toolbar or the navigation pane shall not be displayed on initial activation of the annotation. **Context** click events shall be handled by the interactive PDF processor and shall not be passed to the annotation handler. The annotation appears as a separate window with an initial width of four inches and an initial height of three inches and will be aligned 0.25 inches off the top right (in left to right reading languages) corner of the document area.

13.7.2.3 RichMediaContent dictionary

13.7.2.3.1 General

The *RichMediaContent dictionary* contains content that is present within the annotation as referenced by the RichMediaSettings dictionary (see "Table 334 — Entries in a RichMediaSettings dictionary"). "Table 341 — Entries in a RichMediaContent dictionary" details the elements of the RichMediaContent dictionary.

Table 341 — Entries in a RichMediaContent dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, it shall be *RichMediaContent*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Assets	name tree	(*Optional; PDF 2.0*) A name tree of embedded file specification dictionaries as detailed in 7.11.3, "File specification dictionaries".
Configurations	array	(*Optional; PDF 2.0*) An array where each element is an indirect object reference to a RichMediaConfiguration dictionary.
Views	array	(*Optional; PDF 2.0*) An array where each element is an indirect object reference to a 3D view dictionary. See "Table 315 — Entries in a 3D view dictionary" for the details of the entries of this dictionary. Default value: If no views are specified, default values shall be used for the components of a view dictionary, including rendering/lighting modes, background colour, and camera data.

13.7.2.3.2 The assets name tree

The **Assets** entry takes a name tree of *embedded file specification dictionaries*. (For usage, see "Table 341 — Entries in a RichMediaContent dictionary" and "Table 343 — Entries in a RichMediaInstance dictionary".)

The text string that represents the file name in the name tree shall match the values stored for both the **F** and **UF** keys, which shall be the same value. The file name shall be encoded as a relative **URI** and has the following naming restrictions:

- The string shall be a PDF text string.
- The string shall not contain any embedded NULL (U+0000) characters.
- The number of characters in the string shall be between 1 and 255 inclusive.
- The string shall not contain any of these six characters: COLON (U+003A) (:), ASTERISK (U+002A) (*), QUOTATION MARK (U+0022) ("), LESS-THAN SIGN (U+003C) (<), GREATER-THAN SIGN (U+003E) (>), and VERTICAL LINE (U+007C) (|).
- The last character shall not be a FULL STOP (U+002E) (.).

EXAMPLE Assets name tree

```
29 0 obj                                        % Assets name tree
   << /Names
      [(3D.prc) 30 0 R
         … more name value pairs where each value is a file specification dictionary reference
…
      ]
   >>
endobj

30 0 obj                                        % File specification dictionary for 3D file
    <</Type /Filespec
      /F (3D.prc)
      /UF (3D.prc)
      /EF <</F 40 0 R>>                         % Stream containing the 3D file
   >>
endobj

40 0 obj                                        % Embedded file stream for 3D file
   << /Type /EmbeddedFile                       % 3D.prc
      /Length …
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        /Filter …
    >>
stream
… Data for 3D.prc …
endstream
endobj
```

13.7.2.3.3 RichMediaConfiguration dictionary

The *RichMediaConfiguration dictionary* describes a set of instances that are loaded for a given scene configuration. The configuration that shall be loaded when an annotation is activated is referenced by the Configuration key in the RichMediaActivation dictionary specified in the RichMediaSettings dictionary. "Table 342 — Entries in a RichMediaConfiguration dictionary" details the elements of the RichMediaConfiguration dictionary.

Table 342 — Entries in a RichMediaConfiguration dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, it shall be *RichMediaConfiguration*.
Subtype	name	(*Optional; PDF 2.0*) A name specifying the primary content type for the configuration. Valid values are *3D*, *Sound*, and *Video*. Default value: If no value is specified, the run time shall determines the scene type by referring to the type of asset file specified by the first element in the **Instances** array.
Name	text string	(*Optional; PDF 2.0*) A unique name for the configuration.
Instances	array	(*Optional; PDF 2.0*) An array of indirect object references to RichMediaInstance dictionaries. (See "Table 343 — Entries in a RichMediaInstance dictionary".)

A *RichMediaConfiguration* may be an aggregate of several RichMediaInstance dictionaries with different values for subtype. The **Subtype** entry helps to describe the behavior for the collection of those RichMediaInstance dictionaries.

Setting the **Subtype** suggests the author's intended use of the assets, which better informs the choices when presenting content-specific user interfaces during the authoring or editing process.

13.7.2.3.4 RichMediaInstance dictionary

The *RichMediaInstance dictionary*, referenced by the **Instances** entry of the RichMediaConfiguration dictionary ("Table 342 — Entries in a RichMediaConfiguration dictionary"), describes a single instance of an asset with settings to populate the artwork of an annotation, as described in "Table 343 — Entries in a RichMediaInstance dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 343 — Entries in a RichMediaInstance dictionary

Key	Type	Value
Type	name	*(Optional; PDF 2.0)* The type of PDF object that this dictionary describes. If present, it shall be *RichMediaInstance*.
Subtype	name	*(Required; PDF 2.0)* A name specifying the content type for the instance. Valid values are *3D*, *Sound*, and *Video*. The subtype shall match the asset file type of the instance.
Asset	dictionary	*(Required; PDF 2.0)* A dictionary that shall be an indirect object reference to a file specification dictionary that is also referenced in the **Assets** name tree specified in the RichMediaContent dictionary of the annotation.

EXAMPLE 1 RichMediaInstance dictionaries

```
15 0 obj                                        %RichMediaInstances array
   [16 0 R
      … more references to RichMediaInstance dictionaries …
   ]
endobj

16 0 obj                                        %RichMediaInstance dictionary
   <</Type /RichMediaInstance
      /Subtype /3D
      /Asset 30 0 R                             %Reference to 3D filespec in Assets
   >>
endobj
```

13.7.2.3.5 View dictionary

View dictionaries specify a unique view of the artwork that may be used when the annotation is activated or selected. A View dictionary entry in a rich media annotation contains a superset of the information present in a 3D view (**3DV**) dictionary as specified in "Table 315 — Entries in a 3D view dictionary". "Table 344 — Additional entries in a 3D view dictionary" describes the entries in addition to those present within a 3D view dictionary.

Table 344 — Additional entries in a 3D view dictionary

Key	Type	Value
Snapshot	stream	(*Optional; PDF 2.0*) A stream that shall contains an image XObject (see 8.9, "Images") to be displayed when the view is invoked. Default value: No image shall be used, the currently active rich media content is displayed.
Params	array	(*Optional; PDF 2.0*) An array containing View Params dictionaries (see 13.7.2.3.6, "View params dictionary"). Default value: an empty array. NOTE (2020) The **Params** key was accidently omitted from earlier PDF 2.0 specifications and was reinstated in this document.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

If the **Snapshot** entry exists, the image it describes shall be displayed in place of the current artwork. Because the snapshot image should be integrated by the player with other elements, such as user interface controls for video playback, the colour space of the image may be modified by the interactive PDF processor.

13.7.2.3.6 View params dictionary

13.7.2.3.6.1 General

The *View Params dictionary* provides a reference to a RichMediaInstance dictionary within the **Instances** array, and additional **Data** that is passed to the instance. This data may be formatted to serve the individual purpose of the content being referenced.

Table 345 — Entries in a View Params dictionary

Key	Type	Value
Instance	dictionary	(*Required; PDF 2.0*) A dictionary that shall be an indirect object reference to a RichMediaInstance dictionary that is also present in the **Instances** array of the annotation.
Data	text string or stream	(*Required; PDF 2.0*) A text string or stream containing state data that shall be passed to the instance when the view is triggered.

The **Data** entry ("Table 345 — Entries in a View Params dictionary") is used to store and load opaque data specific to the **Instance** referenced in the same dictionary when a **View** is activated.

NOTE This system allows content-specific state information to be stored when a view is created, such as during a comment or drawing markup.

13.7.2.3.6.2 Saving state data

When the **View** is created, the run time shall iterate through all active instances for the current configuration and query them for state data. For 3D content, an ECMAScript *StateEvent* is invoked with the type property set to the value save. After the event is handled, the value for the data property, if it is present and can be interpreted as a string, shall be stored as the value for **Data**.

13.7.2.3.6.3 Loading state data

When the **View** is activated, the run time shall iterate through each View Params dictionary and load the state data for each instance referenced. For 3D content, an ECMAScript *StateEvent* is invoked with the type property set to the value load and the data property set to the value stored for **Data**.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 useful to the interchange of documents among PDF processors:

- *Procedure sets* (14.2, "Procedure sets") that define the implementation of PDF operators (*deprecated in PDF 2.0*)
- *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") allow a PDF processor 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 organisation on the content of a document
- *Tagged PDF* (14.8, "Tagged PDF"), a set of conventions for using marked-content and logical structure facilities to enable the extraction and reuse of a document’s content for other purposes
- Various ways of enhancing the repurposing and accessibility of a document (14.9, "Repurposing and accessibility support"), including natural language identification of document content
- 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
- Subclause 14.12, “Document parts" describes how additional page-level grouping information can be added to a PDF file. A primary use of document parts is to facilitate job ticket-based workflows that process large documents section by section.
- Subclause 14.13, "Associated files" describes a means to associate content in other formats with selected objects of a PDF file and to identify the relationship between them.

14.2 Procedure sets

This feature has been deprecated since PDF 1.4.

The PDF operators used in content streams are grouped into categories of related operators called *procedure sets* (see "Table 346 — Predefined procedure set"). 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 language compatible output device. The names identify PostScript language 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 346 — Predefined procedure set".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 346 — Predefined procedure set

Name	Category of operators
PDF	Painting and graphics state
Text	Text
ImageB	Grayscale images or image masks
ImageC	Colour images
ImageI	Indexed (colour-table) images

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 actual content or structure) is called *document metadata*. Beginning with PDF 1.4, it is possible to include metadata for individual objects in a document. Such object-specific metadata is called object-level metadata.

Metadata can be stored in a PDF document in the following ways:

- For document data and for object-level metadata: In a *metadata stream* (*PDF 1.4*) associated with the document or a component of the document (14.3.2, "Metadata streams"). Metadata streams are the preferred method in PDF 2.0.
- For document metadata only: In a *document information dictionary* associated with the document (14.3.3, "Document information dictionary"). Except for the **CreationDate** and **ModDate** entries, the use of the document information dictionary for document metadata is deprecated in PDF 2.0.

PDF's document information dictionaries provided the initial means of including metadata in a PDF file. When metadata streams began to be used in PDF 1.4, XMP was introduced to provide a much richer mechanism to represent metadata entries. Using XMP, a document's title can be in more than one language or a document's authors can be represented as a list. Machine-generated metadata for date and time of creation or last modification of a PDF document can adequately be represented in the document information dictionary and in the document's metadata stream. In some cases document information dictionary entries are required, such as when **PieceInfo** dictionaries (see "14.5, "Page-piece dictionaries") are used; they can be present in both the document information dictionary and the document-level metadata stream as needed.

14.3.2 Metadata streams

Metadata, both for an entire document and for objects within a document, can be stored in stream dictionaries (*PDF 1.4*). These streams are called *metadata streams*. Besides the entries common to all stream dictionaries (see "Table 5 — Entries common to all stream dictionaries"), a metadata stream dictionary shall contain the additional entries listed in "Table 347 — Additional entries in a metadata stream dictionary".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 347 — 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*.

The contents of a metadata stream shall be the metadata represented in Extensible Markup Language (XML) and the grammar of the **XML** representing the metadata shall be defined according to the extensible metadata platform specification (ISO 16684-1).

A metadata stream representing document-level metadata can be attached to a PDF document through the **Metadata** entry in the document catalog dictionary (see 7.7.2, "Document catalog dictionary").

A metadata stream representing object-level metadata can be attached to the object through the **Metadata** entry in a stream or dictionary representing the object or associated with the object (see "Table 348 — Additional entry for components having metadata").

Object level metadata can also be associated with marked-content within a content stream, by including a metadata stream in the property list dictionary for this marked-content. Because a stream dictionary is always an indirect object, a property list containing a metadata stream cannot be encoded inline in the content stream, but needs to be encoded as a named resource (see 14.6.2, "Property lists").

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. When there is ambiguity about exactly which stream or dictionary may bear the **Metadata** entry, the metadata shall be attached as closely as possible to the object that actually stores the data resource described.

Table 348 — Additional entry for components having metadata

Key	Type	Value
Metadata	stream	(*Optional; PDF 1.4*) A metadata stream containing metadata for the component.

NOTE 1 Metadata describing an **ICCBased** colour space would be attached to the ICC profile stream describing it, and metadata for embedded font files would be attached to font file streams rather than to font dictionaries. Metadata describing a tiling pattern would be attached to the pattern stream's dictionary, but a shading needs to have metadata attached to the shading dictionary rather than to the shading pattern dictionary that refers to it.

NOTE 2 In tables defining document objects, the **Metadata** entry is listed only for those document objects in which it is most likely to be used. However, a **Metadata** entry can appear in other objects as long as those objects are represented as streams or dictionaries.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.3.3 Document information dictionary

The **Info** entry in the trailer of a PDF file (see 7.5.5, "File trailer") is optional. If present it shall hold a *document information dictionary* (see "Table 349 — Entries in the document information dictionary"). Earlier versions of the PDF file format used the *document information dictionary* to represent document level metadata. In PDF 2.0 such use is deprecated except for two entries, **CreationDate** and **ModDate**. For any other document level metadata, a metadata stream (see 14.3.2 "Metadata streams") should be used instead.

Where a document information dictionary contains keys other than **CreationDate** and **ModDate**, the value associated with any such key shall be a text string.

Document information dictionaries are also used with Threads (see "Table 162 — Entries in a thread dictionary").

Table 349 — Entries in the document information dictionary

Key	Type	Value
Title	text string	(*Optional; PDF 1.1; deprecated in PDF 2.0*) The document's title. NOTE 1 The **dc:title** entry in the document's metadata stream can be used to represent the document's title.
Author	text string	(*Optional; deprecated in PDF 2.0*) The name of the person who created the document. NOTE 2 The **dc:creator** entry in the document's metadata stream can be used to represent the person or persons who created the document. This note was corrected in this document (2020).
Subject	text string	(*Optional; PDF 1.1; deprecated in PDF 2.0*) The subject of the document. NOTE 3 The **dc:description** entry in the document's metadata stream can be used to represent the subject the document.
Keywords	text string	(*Optional; PDF 1.1; deprecated in PDF 2.0*) Keywords associated with the document. NOTE 4 The **pdf:Keywords** entry in the document's metadata stream can be used to represent the keywords for the document.
Creator	text string	(*Optional; deprecated in PDF 2.0*) If the document was converted to PDF from another format, the name of the PDF processor that created the original document from which it was converted. NOTE 5 The **xmp:CreatorTool** entry in the document's metadata stream can be used to represent the creation tool of the document.
Producer	text string	(*Optional; deprecated in PDF 2.0*) If the document was converted to PDF from another format, the name of the PDF processor that converted it to PDF. NOTE 6 The **pdf:Producer** entry in the document's metadata stream can be used to represent the tool that saved the document as a PDF.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
CreationDate	date	(*Optional*) The date and time the document was created, in human-readable form (see 7.9.4, "Dates"). NOTE 7 The **xmp:CreateDate** entry in the document's metadata stream can be used to represent document's creation date and time.
ModDate	date	(*Required if **PieceInfo** is present in the document catalog dictionary; otherwise optional; PDF 1.1*) The date and time the document was most recently modified, in human-readable form (see 7.9.4, "Dates"). NOTE 8 The **xmp:ModifyDate** entry in the document's metadata stream can be used to represent the date and time the document was most recently modified.
Trapped	name	(*Optional; PDF 1.3; deprecated in PDF 2.0*) 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 9 The value of this entry can be set automatically by the software creating the document's trapping information, or it can be known only to a human operator and entered manually. NOTE 10 The **pdf:Trapped** entry in the document's metadata stream can be used to represent the trapping information for the document.

EXAMPLE This example shows a document information dictionary containing just the creation and last modification date, together with the same document's metadata stream containing the creation and last modification date and several other document level metadata fields.

```
101 0 obj                                %document information dictionary
      <</CreationDate (D:20140314124211+01'00)
        /ModDate (D:20140924212303+02'00)
      >>
endobj

102 0 obj                                %document level metadata stream
      << /Type /Metadata
         /Subtype /XML
         /Length 103 0 R
      >>
stream

<?xpacket begin="Ôªø" id="W5M0MpCehiHzreSzNTczkc9d"?>

<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="My XMP Tool Kit v3.7">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/">
         <xmp:CreateDate>2014-03-14T12:42:11+01:00</xmp:CreateDate>
         <xmp:ModifyDate>2014-09-24T21:23:03+02:00</xmp:ModifyDate>
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            <xmp:CreatorTool>My Word Processor v10.7</xmp:CreatorTool>
            <xmp:MetadataDate>2014-09-24T21:23:03+02:00</xmp:MetadataDate>
         </rdf:Description>
         <rdf:Description rdf:about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
            <pdf:Producer>My Word Processor PDF Exporter Module v2.1</pdf:Producer>
         </rdf:Description>
         <rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
            <dc:format>application/pdf</dc:format>
            <dc:title>
               <rdf:Alt>
                 <rdf:li xml:lang="x-default">Annual report 2014</rdf:li>
                 <rdf:li xml:lang="en">Annual report 2014</rdf:li>
                 <rdf:li xml:lang="de">Jahresbericht 2014</rdf:li>
               </rdf:Alt>
            </dc:title>
            <dc:creator>
               <rdf:Seq>
                 <rdf:li>John Doe</rdf:li>
                 <rdf:li>Mary Miller</rdf:li>
               </rdf:Seq>
            </dc:creator>
         </rdf:Description>
      </rdf:RDF>
   </x:xmpmeta>

   <?xpacket end="w"?>
   endstream
   endobj
```

14.3.4 Reconciling two sources of document metadata

Information about time and date of creation might differ from the most recent modification of a PDF document. If this is the case, the following rules shall apply when writing a PDF document:

- When writing the time and date of creation for the first time, typically when a new document is created, a PDF processor shall ensure that the data in the document information dictionary and the document level metadata stream – if both are written – are fully equivalent.
- When writing modifications to an existing PDF document, if the PDF document already contains time and date of creation in both the document information dictionary and in the document's metadata stream, and the two are not equivalent, a PDF processor should leave the inconsistent values unchanged.
- When writing modifications to an existing PDF document, if the PDF document contains time and date of creation only in the document information dictionary or in the document's metadata stream but not both, a PDF processor may add the information to the other, as long as both are fully equivalent.
- When writing the time and date of the most recent modification, typically when an existing document has been modified, a PDF processor shall ensure that the data in the document information dictionary and the document level metadata stream – if both are written – are fully equivalent.

If a PDF processor encounters inconsistent data for time and date of creation or for most recent modification in the document information dictionary and in the document's metadata stream, it is at the discretion of the PDF processor how to use this data.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.4 File identifiers

PDF file identifiers shall be defined by the **ID** entry in a PDF file's trailer dictionary (see 7.5.5, "File trailer"). 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 PDF file at the time it was originally created and shall not change when the PDF file is updated. The second byte string shall be a changing identifier based on the PDF file's contents at the time it was last updated (see 7.5.6, "Incremental updates"). When a PDF file is first written, both identifiers shall be set to the same value. If the first identifier in the reference matches the first identifier in the referenced file's **ID** entry, and the last identifier in the reference matches the last identifier in the referenced file's **ID** entry, it is very likely that the correct and unchanged PDF file has been found. If only the first identifier matches, a different version of the correct PDF file has been found.

PDF writers should attempt to ensure the uniqueness of file identifiers. This may be achieved by computing them by means of a message digest algorithm such as MD5 (described in *Internet RFC 1321*), using the following information:

- The current time;
- A string representation of the PDF file's location;
- The size of the PDF file in bytes.

14.5 Page-piece dictionaries

A page-piece dictionary (*PDF 1.3*) may be used to hold private PDF processor 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 31 — Entries in a page object") or form dictionary (see "Table 93 — Additional entries specific to a Type 1 form dictionary"). Beginning with PDF 1.4, private data may also be associated with the PDF document by means of the **PieceInfo** entry in the document catalog dictionary (see "Table 29 — Entries in the catalog dictionary").

NOTE PDF writers can 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 PDF processor that produces it (such as information on object grouping for a graphics editor or the layer information used by Adobe Photoshop®) but can be ignored by general-purpose PDF processors.

As "Table 350 — Entries in a page-piece dictionary" shows, a page-piece dictionary may contain any number of entries, each keyed by the name of a distinct PDF processor or of a well-known data type recognised by a family of PDF processors. The value associated with each key shall be a data dictionary containing the private data that shall be used by the PDF processor. 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 PDF processor other than the actual content of the document, page, or form.

Table 350 — Entries in a page-piece dictionary

Key	Type	Value
any valid second-class name	dictionary	A data dictionary (see "Table 351 — Entries in a data dictionary").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 351 — Entries in a 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 PDF processor.
Private	(any)	(*Optional*) Any private data appropriate to the PDF processor, typically in the form of a dictionary.

The **LastModified** entry indicates when a PDF processor last altered the content of the document, 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 31 — Entries in a page object" and "Table 93 — Additional entries specific to a Type 1 form dictionary"), or the **ModDate** entry of the document information dictionary (see "Table 349 — Entries in the document information dictionary"), to ascertain whether the data dictionary corresponds to the current content of the document, 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.

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 PDF processor. 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, can use marked-content to identify a set of related objects as a group to be processed as a single unit. A text-processing application can 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 14.7.5, "Structure content").

All marked-content operators except **EMC** shall take a *tag* operand indicating the role or significance of the marked-content element to the PDF processor. All such tags should have second-class names registered with ISO (see Annex E, "Extending PDF") 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".

NOTE 3 The *tag* operand of marked-content operators have no relationship to Tagged PDF (see 14.8 "Tagged PDF") and thus is not rolemapped.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

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. "Table 352 — Marked-content operators" summarises the marked-content operators.

NOTE 4 A marked-content sequence cannot cross page boundaries.

The **Contents** entry of a page object (see 7.7.3.3, "Page objects"), whether a single stream or an array of streams, is considered a single stream with respect to marked-content sequences.

Table 352 — 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 representing 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 representing 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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
      …
   ET
EMC
```

and

```
BT
   BMC
      …
   EMC
ET
```

are valid, but

```
BMC
   BT
```

© ISO 2020 – All rights reserved

```
            ...
        EMC
    ET
```

and

```
    BT
        BMC
            ...
    ET
        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 information (either private or of types defined in this document) meaningful to the PDF processor. PDF processors 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).

NOTE 1 Property lists are used by several PDF features, including optional content (see 8.11, "Optional content", tagged PDF (see 14.8, "Tagged PDF"), object metadata (see 14.3.2, "Metadata streams") and Associated Files (see 14.13.5, "Associated files linked to graphics objects").

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.

NOTE 2 (2020) This means that indirect references of the form of 10 0 R can never occur within content streams.

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 organisation 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 PDF writers to choose what structural information to include and how to represent it, while enabling PDF processors to navigate a PDF file without knowing the producer's structural conventions.

PDF logical structure shares basic features with standard document markup languages such as HTML 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 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

MarkInfo entry in the document catalog dictionary (see 7.7.2, "Document catalog dictionary") shall specify a *mark information dictionary*, whose entries are shown in "Table 353 — Entries in the mark information dictionary". It provides additional information relevant to specialised uses of structured PDF documents.

Table 353 — 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 14.7.6.4, "User properties"). Default value: *false*.
Suspects	boolean	(*Optional; PDF 1.6; deprecated in PDF 2.0*) A flag indicating the presence of tag suspects. 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 catalog dictionary (see 7.7.2, "Document catalog dictionary"). "Table 354 — Entries in the structure tree root" 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.

Table 354 — 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 (see 7.9.6, "Name trees") that maps element identifiers (see "Table 355 — Entries in a structure element dictionary") to the structure elements they denote.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 14.7.5.4, "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 14.7.5.4, "Finding structure elements from content items" for further discussion.
ParentTreeNextKey	integer	(*Optional*) An integer greater than any key in the parent tree **ParentTree** and that shall be used as the key for the next entry added to the parent tree.
RoleMap	dictionary	(*Optional*) A dictionary that shall map name objects designating names of structure types used in the document to a name object designating the name of 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 14.7.6.2, "Attribute classes").
Namespaces	array	*(Required if any structure elements have namespace identifiers; PDF 2.0)* An array of namespaces used within the document (see 14.7.4.2, "Namespace dictionary").
PronunciationLexicon	array of file specifications	(*Optional; PDF 2.0*) An array containing one or more indirect references to file specification dictionaries, where each specified file shall be a pronunciation lexicon, which is an XML file conforming to the Pronunciation Lexicon Specification (PLS) Version 1.0. These pronunciation lexicons may be used as pronunciation hints when the document's content is presented via text-to-speech. Where two or more pronunciation lexicons apply to the same text, the first match – as defined by the order of entries in the array and the order of entries inside the pronunciation lexicon file – should be used. See 14.9.6, "Pronunciation hints" for further discussion.
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for the entire structure tree. See 14.13, "Associated files".

Structure elements shall be represented by a dictionary, whose entries are shown in "Table 355 — Entries in a structure element dictionary". The **K** entry shall specify the children of the structure element, which may be zero or more items of the following kinds:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- 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 14.7.5, "Structure content".

Table 355 — 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, "Extending PDF".
P	dictionary	(*Required; shall be an indirect reference*) The structure element or the structure tree root that is the immediate parent of this structure element 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 354 — Entries in the structure tree root") defines the correspondence between element identifiers and the structure elements they denote.
Ref	array	(*Optional; PDF 2.0*) An array containing zero, one or more indirect references to structure elements. A **Ref** identifies the structure element or elements to which the item of content, contained within this structure element, refers (e.g. footnotes, endnotes, sidebars, etc.).
Pg	dictionary	(*Optional; required if K is an integer object or an array containing integer objects; 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 in any combination: • 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 (see "Table 357 — Entries in a marked-content reference dictionary") • An object reference dictionary denoting a PDF object (see "Table 358 — Entries in an object reference dictionary") Each of these objects other than the first (structure element dictionary) shall be considered to be a content item; see 14.7.5, "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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 14.7.6, "Structure attributes" and 14.7.6.3, "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 14.7.6.2, "Attribute classes" and 14.7.6.3, "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 14.7.6.3, "Attribute revision numbers"). The value shall be a non-negative integer. Default value: *0*.
T	text string	(*Optional*) The title of the structure element, a text string representing it in human-readable form. The title should characterise 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").
Alt	text string	(*Optional*) An alternative 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 or an acronym.
ActualText	text string	(*Optional; PDF 1.4*) Text that is an exact replacement for the content enclosed by 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").
AF	array of dictionaries	(*Optional; PDF 2.0*) An array of one or more file specification dictionaries (7.11.3, "File specification dictionaries") which denote the associated files for this entire structure element. See 14.13, "Associated files".
NS	dictionary	*(Optional; PDF 2.0)* An indirect reference to a namespace dictionary defining the namespace this element belongs to (see 14.7.4, "Namespaces"). If not present, the element shall be considered to be in the default standard structure namespace (see 14.8.6, "Standard structure namespaces").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
PhoneticAlphabet	Name	(*Optional; PDF 2.0*) Property for a structure element that indicates the phonetic alphabet used by a **Phoneme** property. Applies to the structure element and its children, except where overridden by a child structure element. Currently defined values are: • *ipa* for the International Phonetic Alphabet by the International Phonetic Association • *x-sampa* for Extended Speech Assessment Methods Phonetic Alphabet (X-SAMPA) • *zh-Latn-pinyin* for Pinyin Latin romanization (Mandarin) • *zh-Latn-wadegile* for Wade-Giles romanization (Mandarin) Other values may be used. Default value: *ipa*. See 14.9.6, "Pronunciation hints" for further discussion.
Phoneme	text string	(*Optional; PDF 2.0*) Property for a structure element that may be used as pronunciation hint. It is an exact replacement for content enclosed by the structure element and its children. The value for a **Phoneme** property is to be interpreted based on the **PhoneticAlphabet** property in effect. See 14.9.6, "Pronunciation hints" for further discussion.

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 PDF processors, PDF defines a set of standard structure types; see 14.8.4, "Standard structure types". PDF writers 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 should be provided in the structure tree root using the **RoleMap** entry or, in the case of namespaces, the **RoleMapNS** entry within an **NS** entry, to map the structure types used in the document to their nearest equivalents in the standard set.

The **RoleMap** dictionary shall be comprised of a set of keys representing structure element types rolemapped to other structure element types. The corresponding value for each of these keys shall be a single name identifying the target structure element type.

A structure type 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 structure type with the same name as a standard role, even though its use differs from the standard role.

NOTE 1 A structure type named Advertising can be mapped to the standard type **Part**. The equivalence need not be exact; the role map merely indicates an approximate analogy between types, allowing PDF processors to share nonstandard structure elements in a reasonable way.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 2 The same structure type can 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 can define a bidirectional mapping. A PDF processor using the role map needs to follow the chain of associations until it either finds a structure type it recognises or returns to one it has already encountered.

NOTE 3 In PDF versions earlier than 1.5, standard element types were never remapped.

14.7.4 Namespaces

14.7.4.1 General

PDF documents are used to represent many classes of documents as well as many classes of content within those documents. The *standard structure types* (see 14.8.4, "Standard structure types") address many classes of documents. The mechanism for providing a role mapping for custom structure types allows the inclusion of non-standard, custom, types. However, prior to PDF 2.0, there has been no mechanism for identifying and exchanging these custom structure types, other than to use their mapping to the standard structure types.

The concept of *namespaces* is well understood in the XML world, allowing custom XML tagsets to be interchanged and identified. Schemata are often defined for these namespaces using a variety of schema languages to allow programmatic exchange of tagsets. Many PDF documents are authored by conversion from other formats, many of which have rich structures and content with their own structures. PDF 2.0 introduces the namespace mechanism as a component of its logical structure, where one or more namespaces may be specified as being used within the document (see 14.7.4.2, "Namespace dictionary").

14.7.4.2 Namespace dictionary

A *namespace dictionary* shall be used to define a namespace (see 14.7.4, "Namespaces") within the structure tree (see 14.7.2, "Structure hierarchy"). "Table 356 — Entries in a namespace dictionary" shows the entries in the namespace dictionary. The **NS** entry shall specify the namespace name, which should take the form of a uniform resource identifier (URI).

NOTE 1 It is not generally expected that a URI for a namespace name will resolve. It is instead used for uniqueness. A URI specified here can correspond to an existing XML namespace (e.g. "http://www.w3.org/1998/Math/MathML" for MathML 3.0).

An optional **Schema** entry may be provided to identify the schema file for the namespace.

NOTE 2 There is no requirement that the schema be provided in any specific format (e.g. RelaxNG or W3C Schema), though the expectation is that the format would be machine readable.

A **RoleMapNS** entry may also be provided to map the entries in the namespace to those of another namespace. For a document that is compliant with tagged PDF see 14.8.6, "Standard structure namespaces" for namespace requirements.

NOTE 3 Role mapping to one of the standard structure namespaces can be achieved either directly in the provided role map or transitively through one or more namespaces which eventually provide a role map to one of the standard structure namespaces.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 356 — Entries in a namespace dictionary

Key	Type	Value
Type	name	(*Optional; PDF 2.0*) The type of PDF object that this dictionary describes. If present, shall be *Namespace*.
NS	text string	(*Required; PDF 2.0*) The string defining the namespace name which this entry identifies (conventionally a uniform resource identifier, or URI).
Schema	file specification	(*Optional; PDF 2.0*) A file specification identifying the schema file, which defines this namespace.
RoleMapNS	dictionary	*(Optional; PDF 2.0)* A dictionary that shall, if present, map the names of structure types used in the namespace to their approximate equivalents in another namespace. The dictionary shall be comprised of a set of keys representing structure element types in the namespace defined within this namespace dictionary. The corresponding value for each of these keys shall either be a single name identifying a structure element type in the default standard structure namespace or an array where the first value shall be a structure element type name in a target namespace with the second value being an indirect reference to the target namespace dictionary.

When the owner of an attribute object (see Table 360 — Entries common to all attribute object dictionaries) is specified by an **NS** entry, the namespace name shall be considered as identifying the owner. For common namespace names which correspond to the values of owner entries defined in Table 376 — Standard structure attribute owners, they shall be considered equivalent.

14.7.5 Structure content

14.7.5.1 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 subclauses.

14.7.5.1.1 Content items

Content items are of two kinds:

- Marked-content sequences within content streams (14.7.5.2, "Marked-content sequences as content items")
- Complete PDF objects such as annotations and XObjects (see 14.7.5.3, "PDF objects as content items")

The **K** entry in a structure element dictionary (see "Table 355 — Entries in a structure element dictionary") specifies 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 shall not have other content items nested within them for purposes of logical structure. The hierarchical relationship among structure

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 corresponding to a structure content item shall not have another marked-content sequence for a structure 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. Logical structure information associated with a page may be ignored when importing that page into another document with a reference XObject (see 8.10.4.3 "Special considerations").

14.7.5.2 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 contain a property list (see 14.6.2, "Property lists") containing an **MCID** entry, which 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 14.7.5.4, "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 355 — Entries in a structure element dictionary"). This reference may have two forms:

- A dictionary object called a *marked-content reference*. "Table 357 — Entries in a marked-content reference dictionary" shows the contents of this type of dictionary, which shall specify the

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

marked-content identifier, as well as other information identifying the stream in which the sequence is contained. Example 2 in this subclause illustrates the use of a marked-content reference to the marked-content sequence shown in Example 1.

- 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.

Table 357 — 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 indirect reference to the PDF object referencing the stream identified by the **Stm** key. NOTE A common use for this would be to identify the annotation dictionary owning the appearance stream.
MCID	integer	(*Required*) The marked-content identifier of the marked-content sequence within its content stream.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

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
```

EXAMPLE 3

```
1 0 obj                                          %Containing structure element
```

© ISO 2020 – All rights reserved

```
      <</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 is associated with a structure element (see Example 4 in this subclause). In this case, the entire form XObject is 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 are associated with structure elements (see Example 5 in this subclause). 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 shall not be part of a logical structure content item.

A form XObject that is painted with multiple invocations of the **Do** operator shall 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
        <</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
```

14.7.5.3 PDF objects as content items

When a structure element's content consists of an entire PDF object, such as an XObject directly or indirectly referenced by a page description or an annotation, the object shall be identified in the structure element's **K** entry by an object reference dictionary ("Table 358 — Entries in an object reference dictionary").

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 14.7.5.2, "Marked-content sequences as content items".

Table 358 — 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 need to be accessed by means of marked-content sequences enclosing particular invocations of the **Do** operator rather than through object references.)

14.7.5.4 Finding structure elements from content items

Because a stream cannot 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, shall also use the parent tree to refer to their parent structure elements.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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 354 — Entries in the structure tree root"). 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 359 — Additional dictionary entries for structure element access"). The values of these entries shall be as follows:

- For an object identified as a content item by means of an object reference (see 14.7.5.3, "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 need to 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 359 — Additional dictionary entries for structure element access"). 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, or in an annotation dictionary. Its value shall be the integer key under which the entry corresponding to the object shall be found in the structural parent tree.

Table 359 — 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
      <</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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
      >>
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
```

14.7.6 Structure attributes

14.7.6.1 General

A PDF processor 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 360 — Entries common to all attribute object dictionaries") identifying the conforming product that owns the attribute information. Other entries, except the **NS** entry, 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, attributes may be used to represent user properties (see 14.7.6.4, "User properties").

NOTE Earlier versions of PDF also provided for the use of streams as attributes.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 360 — Entries common to all attribute object dictionaries

Key	Type	Value
O	name	(*Required*) The name of the PDF processor creating the attribute data. The value shall either be a *NSO*, *UserProperties* (see "Table 361 — Additional entries in an attribute object dictionary for user properties"), one of the values from 14.8.5, "Standard structure attributes", or conform to the guidelines described in Annex E, "Extending PDF". If the value for the **O** entry is *NSO* then the **NS** entry shall be present, and shall identify the owner of the attribute object.
NS	dictionary	(*Required if the value of the* **O** *entry is NSO; not permitted otherwise; PDF 2.0)* An indirect reference to a namespace dictionary defining the namespace that attributes with this attribute object dictionary belong to (see 14.7.4, "Namespaces"). If not present, the attributes in this attribute object dictionary do not have a namespace. NOTE Because the **NS** entry is now reserved within the attribute object dictionary, attributes from existing namespaces with a matching name will not be able to be used.

Any PDF processor may attach attributes to any structure element, even one created by another PDF processor. Multiple PDF processors may attach attributes to the same structure element. The **A** entry in the structure element dictionary (see "Table 355 — Entries in a structure element dictionary") shall hold either a single attribute object or an array of at least one object.

When an array of attribute objects is provided, the value of the **O** and **NS** keys may be repeated across attribute objects. If a given attribute is specified more than once, the later (in array order) entry shall take precedence.

14.7.6.2 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 354 — Entries in the structure tree root"). 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 355 — Entries in a structure element dictionary") shall contain a class name or an array of class names (typically accompanied by revision numbers as well; see 14.7.6.3, "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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.7.6.3 Attribute revision numbers

The features described in this subclause are deprecated with PDF 2.0.

When a PDF processor modifies a structure element or its contents, the change may affect the validity of attribute information attached to that structure element by other PDF processors. A system of revision numbers shall allow PDF processors to detect such changes and update their own attribute information accordingly, as described in this subclause.

A structure element shall have a revision number, that shall be stored in the **R** entry in the structure element dictionary (see "Table 355 — Entries in a structure element dictionary") or default to 0 if no **R** entry is present. Initially, the revision number shall be 0. When a PDF processor 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 there is no **R** entry and the revision number is to be incremented from the default value of 0 to 1, an **R** entry will have to 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 can be associated with more than one structure element (whose revision numbers can 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 PDF processor 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

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

structure element's revision number, the PDF processor may choose to delete all unknown attribute objects from its **A** and **C** arrays. These two actions shall be mutually exclusive: the PDF processor should either increment the structure element's revision number or remove its attribute objects, but it shall not do both.

NOTE 5 Any PDF processor creating attribute objects needs to be prepared for the possibility that they can be deleted at any time by another PDF processor.

14.7.6.4 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 PDF 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 361 — Additional entries in an attribute object dictionary for user properties").

Table 361 — 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 362 — Entries in a user property dictionary").

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 362 — Entries in a user property dictionary").

Table 362 — Entries in a user property dictionary

Key	Type	Value
N	text string	(*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, PDF writers should use only text string, number, and boolean values. PDF processors 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, PDF processors should use a default format.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 353 — Entries in the mark information dictionary". This entry allows PDF processors 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
```

14.7.7 Example of logical structure

The 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 14.7.6, "Structure attributes" and 14.8.5, "Standard structure attributes").

The example in this subclause 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
                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
                            >>
                     >>
          /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
             /Span <</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

             /Span <</MCID 1>>                   %Start of marked-content sequence1
             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
                             >>
                     >>
          /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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
            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

         /Span  <</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

         /Span  <</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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
                  /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
            /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 (number 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
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

```
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 stylised 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, as well as annotations and form fields) to be extracted and reused for other purposes.

A tagged PDF document is one that conforms to all rules described in all of the subclauses in 14.8, "Tagged PDF".

A tagged PDF document shall contain a mark information dictionary (see "Table 353 — Entries in the mark information dictionary") with a value of *true* for the **Marked** entry.

NOTE Tagged PDF is intended for use by tools that perform operations such as:

- Extraction of text and graphics for pasting into other applications
- Automatic reflow of page contents – text as well as associated graphics and images or annotations and form fields – to fit a display area of a different size than was assumed for the original layout
- Processing of content for such purposes as searching, indexing, and spell-checking
- Conversion to other common file formats (such as HTML, XML, and RTF) with document structure preserved
- Making content accessible to users with disabilities

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 is described by one or more page content streams (including any subsidiary streams such as form XObjects), annotations and form fields. Tagged PDF defines some further rules for organising and marking content streams so that additional information may 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 the content order as intended by the content author to guide the layout process if the PDF processor repurposes the page content (see 14.8.2.5, "Page content order and logical content order").
- Guaranteeing that the logical order of content can be deterministically derived from a

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

combination of logical structure and page content (see 14.8.2.5, "Page content order and logical content order").

- Representing text in a form from which a Unicode representation may be unambiguously derived (see 14.8.2.6, "Unicode mapping in tagged PDF").
- Representing word breaks unambiguously (see 14.8.2.6.2, "Identifying word breaks").
- Marking all content with information for making it accessible to users with visual impairments or other disabilities (see 14.9, "Repurposing and accessibility support").

14.8.2.2 Real content and Artifacts

14.8.2.2.1 General

The content in a document may be divided into two classes:

- The *real content* of a document comprises graphics objects, annotations and form fields representing material intentionally introduced by the document's author and necessary to understand the content of the document.
- All other content is considered to be *artifacts*, whether generated by the PDF writer in the course of pagination, layout, or other mechanical processes or introduced by the document author for decoration or other purposes that are not relevant for understanding the content of the document.

The document's logical structure encompasses all real content and describes how real content objects relate to one another. Where artifacts are to be included in the structure tree, they shall be included through the **Artifact** structure element type, and shall not be considered real content.

A document's real content may include graphics objects in the page content stream and subsidiary XObjects and annotations, including widget annotations.

NOTE 1 The above paragraph was clarified in this document (2020).

To support PDF processors 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 as described in 14.9, "Repurposing and accessibility support".

NOTE 2 ISO 14289 (PDF/UA) specifies the use of tagged PDF to accommodate the needs of users with disabilities.

Tagged PDF processors may make various choices about what page content to consider relevant in a tagged PDF document. A text-to-speech engine, for instance, may decide not to speak running heads or page numbers when the page is turned. In general, PDF processors may do any of the following:

- Disregard elements of page content (for example, skip Link annotations) 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 repurposing)
- Substitute an element with its alternative description (see 14.9.3, "Alternate descriptions")

NOTE 3 Depending on their goals, different PDF processors can make different decisions in this regard. The purpose of tagged PDF is not to prescribe what the PDF processor does, but to provide sufficient declarative and descriptive information to allow it to make appropriate choices about how to process the content.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

14.8.2.2.2 Specification of Artifacts

For tagged PDF files, an artifact should be explicitly distinguished from real content through either of the following methods:

- By enclosing it in a marked-content sequence with the tag **Artifact**:

EXAMPLE 1

```
/Artifact BMC
EMC
```

EXAMPLE 2 (marked-content sequence Artifact with a property list entry)

```
/Artifact <<propertyList>> BDC
EMC
```

- By inclusion in the logical structure tree through the use of the **Artifact** structure element type (see "Table 375 — standard structure type Artifact").

NOTE 1 The purpose of the **Artifact** structure element type is to accommodate artifact content in cases that have positional context relative to real content within the structure tree. An example of such content is line numbers.

For artifacts defined using the marked-content sequence method, the form indicated in EXAMPLE 1 shall be used to identify a generic artifact; the form indicated in EXAMPLE 2 shall be used for those artifacts that have an associated property list. "Table 363 — Property list entries for artifacts" shows the properties that may be included in such a property list.

Any content that is not included in the structure tree is an artifact even when not enclosed in a marked-content sequence using the tag **Artifact**.

NOTE 2 The phrase “any content” above refers to all page content as well as annotations.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 363 — 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 *Background*. • *Pagination artifacts* are ancillary page features such as running heads, folios (page numbers) or Bates Numbering. • *Layout artifacts* are purely cosmetic typographical or design elements such as footnote rules or decorative ornaments. • *Page artifacts* are production aids extraneous to the document itself, such as cut marks and colour bars. • *Background artifacts* can occur from document templates that are often repeated unchanged across many pages and include 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.
BBox	rectangle	*(Optional)* 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).

© ISO 2020 – All rights reserved

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. NOTE (2020) The **Attached** key was accidently omitted from the earlier PDF 2.0 specification and was reinstated in this document.
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*. Valid values are *Header*, *Footer*, *Watermark*, *PageNum* (PDF 2.0), *LineNum* (PDF 2.0), *Redaction* (PDF 2.0) and *Bates* (PDF 2.0). Additional values may be specified for this entry, provided they comply with the naming conventions described in Annex E, "Extending PDF".

Some properties defined elsewhere may also be used as entries in the property list of an artifact, including **Alt** (see 14.9.3, "Alternate descriptions"), **ActualText** (see 14.9.4, "Replacement text"), **E** (see 14.9.5, "Expansion of abbreviations and acronyms") or **Lang** (see 14.9.2, "Natural language specification").

Where it is necessary to represent the content of an artifact as text, the property **ActualText** can be used, for instance, to contain the page number for an artifact with a **Subtype** of *PageNum* or the Bates number for an artifact of subtype *Bates*.

NOTE 3 Bates numbering is used in the legal, medical, and business fields in some countries to place identifying numbers and/or date/time-marks on images and documents. For example, it is added during the discovery stage of preparations for trial or when identifying business receipts. This process provides identification, protection, and automatic consecutive numbering.

14.8.2.3 Soft hyphens

In tagged PDF, the visible hyphen that is introduced through the incidental division of a word at the end of a line but which would not be present otherwise, may be represented as a *soft hyphen*, mapped to the Unicode value U+00AD, in one of the ways described in 14.8.2.6, "Unicode mapping in tagged PDF".

NOTE 1 The soft hyphen character represented by the Unicode value U+00AD is distinct from an ordinary hard hyphen, whose Unicode value is U+002D.

The writer of a tagged PDF document shall distinguish explicitly between soft and hard hyphens so that a PDF processor can unambiguously determine which type a given character represents.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 2 In some languages, the situation is more complicated: there can be multiple characters affected by hyphenation, and hyphenation can change the spelling of words. See the Example in 14.9.4, "Replacement text".

14.8.2.4 Hidden or invisible page content

For a variety of reasons, elements of a document's real content can be invisible on the page: they can be clipped; their colour can match the background; or they can be obscured by other, overlapping objects. For the purposes of tagged PDF, page content shall be considered to include all graphics objects in their entirety, regardless of whether they are visible when the document is displayed or printed.

NOTE For example, invisible elements can become visible when content is repurposed, or a text-to-speech engine could choose to speak invisible text.

14.8.2.5 Page content order and logical content order

14.8.2.5.1 General

Page content order shall be defined by the sequencing of graphics objects within a page's content stream.

Logical content order – the ordering for semantic purposes – shall be defined by a depth-first traversal of the document's logical structure hierarchy.

The page content order in a tagged PDF should coincide with the logical content order.

NOTE 1 Page content order is constrained by the need to render objects in an order that produces the desired visual appearance. Logical content order is constrained by the need to reflect the order of the content as intended by its author. For example, the running text of a page, as encoded in the page's content stream, can contain places where it is not possible to make the order in which the text progresses match the logical content order.

Content within a single marked-content sequence (see 14.6, "Marked content") shall be in logical content order for that item of content.

NOTE 2 Both in terms of how page content is encoded in a tagged PDF as well as in terms of how a PDF processor can process that content, efficiency can be impacted if the page content order and logical content order sequences do not coincide. In some cases it is not feasible to make the sequences coincide:

- Regarding the sequence of page objects within a given page, page objects can visually overlap in a way that requires reverse ordering.
- A logical object can extend over more than one PDF page, such as in the case of a headline spanning two pages in a facing pages layout.
- A page can contain the beginnings of two separate articles, each of which is continued onto a later page of the document. In logical content order terms, the last words of the first article appearing on the page are not followed by the first words of the second article on the same page, but rather by the continuation of the first article on a different page.

NOTE 3 Artifacts not contained within an **Artifact** structure element are not considered part of the logical content order because only structure elements are part of the logical content order.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.2.5.2 Sequencing of annotations

Annotations associated with a page are not interleaved within the page's content stream but are listed in the **Annots** array in its page object (see 7.7.3.3, "Page objects"). The position of an annotation in the logical content order is 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 14.7.5, "Structure content"). Structure elements of type **Annot**, **Link**, or **Form** (see 14.8.4.7, "Inline level structure types") explicitly specify the association between a marked-content sequence and a corresponding annotation through an object reference dictionary as described in 14.7.5.3, "PDF objects as content items".

14.8.2.5.3 Reverse-order show strings

The marked-content tag **ReversedChars** informs the PDF processor that show strings within a marked-content sequence contain characters in reverse order. In order for such text to be extracted or read out aloud, the sequence of the characters as found in the show string operator shall be reversed before using them. If the sequence encompasses multiple show strings, only the individual characters within each string shall be reversed.

NOTE 1 In writing systems that are read from right to left (such as Arabic or Hebrew), one expects 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 54 — Glyph metrics". 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, "Organisation and use of fonts") whose character codes are given in reverse order.

The show strings in a **ReversedChars** block may have a SPACE (U+0020) character or other white-space characters at the beginning or end to indicate a word break (see 14.8.2.6.2, "Identifying word breaks") but shall not contain interior SPACE characters or other white-space characters.

NOTE 2 This limitation is not serious, since a SPACE or other white-space character typically 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 interactive PDF processors.

EXAMPLE The sequence

```
/ReversedChars
      BMC
         ( olleH) Tj
         -200 0 Td
         ( .dlrow) Tj
      EMC
```

represents the text

```
Hello world.
```

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.2.6 Unicode mapping in tagged PDF

14.8.2.6.1 General

PDF documents conforming with 14.8, "Tagged PDF" should map every character code in any of the content streams or appearance streams of a document to a corresponding Unicode value. Every character code that belongs to a structure element in the structure tree shall map to Unicode, except where an associated **Alt** or **ActualText** entry applies to the content to which the character code belongs.

NOTE 1 These Unicode values can then be used for such operations as copy-and-paste, searching, text-to-speech conversion, and exporting to other applications or file formats.

NOTE 2 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.

The methods for mapping a character code to a Unicode value are described in 9.10.2, "Mapping character codes to Unicode values".

Private Use Area Unicode values should only be used if no other Unicode value is available, as no pre-defined meaning is associated with the Unicode values in the Private Use Area.

NOTE 3 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") can affect the character stream used by PDF processors. For example, some PDF processors could choose to use the **Alt** or **ActualText** entry and ignore all text and other content associated with the structure element and its descendants.

In some cases a required character may not be available in a given font, for example the soft hyphen character. Such a character may be represented either by adding it to the font's encoding or **CMap** and using **ToUnicode** to map it to the appropriate Unicode value, or by using an **ActualText** entry in the associated structure element to provide substitute characters.

14.8.2.6.2 Identifying word breaks

A document's text content defines not only the characters in a page's text but also the words. Unlike a character, which is defined unambiguously, a word is defined by script and context. A repurposing 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 vocalised; spelling checkers and other applications have varying definitions for 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 PDF processor of a tagged PDF document may find words by sequentially examining the Unicode character stream, as augmented by replacement text specified with **ActualText** (see 14.9.4, "Replacement text").

For this purpose any white-space characters that would be present to separate words in a pure text representation shall be present in the tagged PDF representation of the text.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE 1 As a consequence, the PDF processor can determine word breaks without having to rely on heuristics based on information such as glyph positioning on the page, font changes, or glyph sizes.

The identification of what constitutes a word shall be unrelated to how the text happens to be grouped into show strings. The division into show strings shall have no semantic significance. In particular, a SPACE (U+0020) or other word-breaking character shall be present in a character stream even if a word break happens to fall at the end of a show string.

NOTE 2 Some PDF processors identify words by simply separating them at every SPACE character. Others can be slightly more sophisticated and treat punctuation marks such as hyphens or Em dashes as word separators as well. Still others can identify line-break opportunities by using an algorithm similar to the one in Unicode Standard Annex #29, Unicode Text Segmentation.

14.8.3 Basic layout model

14.8.3.1 General

Tagged PDF's standard structure types and attributes can 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).

NOTE 1 The goal is to provide sufficient information for PDF processors 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, XML, 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.

14.8.3.2 Reference area

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 a **Figure**, **Form** or **Formula**, or other floating element.

The standard structure types can be divided into four main categories according to the roles they play in page layout:

- Grouping elements (see 14.8.4.4, "Grouping level structure types") group other elements into sequences or hierarchies but hold no content directly and have no direct effect on layout.
- Block-level structure elements (BLSEs) (see 14.8.4.5, "Block level structure types") describe the overall layout of content on the page, proceeding in the block-progression direction.
- Inline-level structure elements (ILSEs) (see 14.8.4.7, "Inline level structure types") describe the layout of content within a BLSE, proceeding in the inline-progression direction.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

 © ISO 2020 – All rights reserved

- Some elements can be grouping elements, BLSEs or ILSEs, depending on context or usage. For example, a **Figure** structure element can be treated as either a grouping element, a BLSE or an ILSE.

14.8.3.3 Progression direction

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

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

Standard structure types are divided into categories:

- "Document level structure types" identify a whole document or a fragment of a document. In most cases a tagged PDF will contain exactly one document, but in some other cases a tagged PDF contains several documents that have been merged together into one PDF file, or the tagged PDF is a document containing document fragments which have been inserted into the document.
- "Grouping level structure types" make it possible to organise the overall structure of content in a tagged PDF. For example, a book typically consists of chapters, or a newspaper page consists of several parts called articles, and a scientific publication typically contains sections and several levels of sub-sections.
- "Block level structure types" are structure types that enclose actual content, like a heading or paragraph.
- "Inline level structure types" are structure types that enable structural organisation of content inside block level elements and inside specialised structure elements used as block level elements.

Some structure types — for example **Table** or **Figure** — may be used as block level types or as inline types, whereas others (e.g., **H1**) may only be used as block level types. Whenever a provision in clause 14.8, "Tagged PDF", refers to block level types, all structure elements used as block level elements shall be included. Whenever a provision in clause 14.8, "Tagged PDF", refers to inline level types, all structure elements used as inline level elements shall be included.

To determine the category that is applicable to a structure element that may either be a block level structure element or an inline level structure element, the following applies:

- If the structure element is used inside a block level element, it is an inline level structure element
- In all other cases it is a block level structure element.

All structure elements occurring within a tagged PDF document shall have a type matching one of those defined as a Standard Structure Type, or a role map providing a mapping from the non-standard type to a Standard Structure Type.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.2 Nesting of standard structure elements

Annex L, "Parent-child relationships between the standard structure elements in the standard structure namespace for PDF 2.0" defines the nesting rules for standard structure elements. In addition, supplemental rules apply for some standard structure elements. These supplemental rules are defined in the subclauses specific to each of the standard structure types.

Structure elements other than the standard structure elements identified in clause 14.8.4 "Standard structure types" or in the standard structure namespaces (see 14.8.6.2, "Role maps and namespaces") shall nest in relation to standard structure elements according to the requirements of the structure elements to which they are role mapped.

NOTE The nesting rules detailed in Annex L, "Parent-child relationships between the standard structure elements in the standard structure namespace for PDF 2.0" address in detail the following system model:

- Structure elements can be empty.
- If a structure element contains neither structure elements nor real content it is empty.
- Unless specifically constrained in 14.8, "Tagged PDF", any number of structure elements can exist in any order inside a parent structure element.
- Real content is contained inside block level or inline level structure elements.
- Document level structure elements that are not empty contain other document level, grouping or block level elements, and cannot contain inline level structure elements or content.
- Grouping structure elements that are not empty contain other document level, grouping or block level elements, and cannot contain inline level structure elements or content.
- Inline level elements can contain other inline level structure elements but cannot contain any other type of structure element.

14.8.4.3 Document level structure types

In a tagged PDF file a *logical document* is a portion of content typically perceived as a semantically self contained document of any granularity. Examples range from short memos to articles, presentations or fillable forms to comprehensive publications like magazines or books.

A *logical document fragment* is a portion of content that constitutes – or is intended or perceived as – just a part of a logical document, regardless of whether it was extracted from a logical document or originated as a document fragment. A tagged PDF file may consist of one or more logical documents or logical document fragments as described in "Table 364 — Document level structure types".

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 364 — Document level structure types

Structure Type	Category	Description
Document	Document	Encloses a logical document. EXAMPLE 1 A mail merge PDF typically contains a number of letters to different recipients. This implies that the PDF at the top level is one document, containing multiple documents at its child level where each such document is a letter to a recipient. EXAMPLE 2 A Proceedings PDF generally includes individual papers given at a conference. This implies that the PDF at the top level is one document containing several documents.
DocumentFragment	Document	*(PDF 2.0)* Encloses a *logical document fragment*. A document fragment is typically created by extracting it as a part from an original complete document. As a consequence the structure of the fragment may be incomplete, and content may start at an arbitrary point within the original logical content structure. EXAMPLE 3 The following are just two possible types of document fragments: • an extract from an original document is inserted into a new containing document • extracts from original documents are concatenated into a new document

NOTE 1 A logical document can contain any number of **DocumentFragment** elements, including nested logical document fragments.

Each logical document fragment shall define its own logical structure. The logical structure of a logical document fragment may start in the middle of what was the parent document's logical structure. For example, it could start at an arbitrary paragraph or at a heading level 2 or 3 instead of heading level 1 or even in the middle of a list.

NOTE 2 **DocumentFragments** are especially useful when hierarchical aspects of logical structure from the original document are to be maintained but are incomplete and thus would be difficult to understand or process.

Within each **Document** or **DocumentFragment** structure element, all heading structure elements shall either be **Hn** or **H**. See "Table 366 — Block level structure types" for more information on heading structure elements.

An XMP metadata stream (see 14.3.2, "Metadata streams") in a **Document** or **DocumentFragment** structure element may be used to include document metadata for a logical document nested inside a tagged PDF.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.4 Grouping level structure types

This subclause describes structure types for grouping elements, as detailed in "Table 365 — Grouping level structure types", that are used to organise the overall structure of content in a tagged PDF.

Table 365 — Grouping level structure types

Structure Type	Category	Description
Part	Grouping	Encloses a grouping of structure elements without consideration for their hierarchy. NOTE 1 **Part** is the semantic equivalent of **Div**. A structure element with the type of **Part** shall inherit the containment requirements and limitations of its parent element. Where the parent element is itself a structure element of type **Part**, then the inheritance shall recurse to the first parent element whose type is not **Part**. EXAMPLE 1 The following are just a few of the possible types of content that could be marked as **Part**: • frontmatter or backmatter in a book • a table of contents or bibliography section in a document • main body of content in a document • advertising section in a magazine • a group of pages (for example, a spread in a magazine) • a group of form fields • a group of **Figure** standard structure elements • a publisher's indicia
Sect	Grouping	Encloses a grouping of structure elements with consideration for their hierarchy. EXAMPLE 2 The following are just a few of the possible types of content that could be marked as **Sect**: • clauses in a technical document • components of an article in a newspaper or magazine • Elements of a recipe (e.g., "name", "instructions", "ingredients", etc.)

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Structure Type	Category	Description
Div	Grouping	Encloses content structured in fashion that is orthogonal to the semantic structure. It can be used as a role mapping target for custom tags for which no suitable standard structure element is available, or where attributes are applied in a non-semantic fashion. NOTE 2 Nesting **Div** structure elements allows content to be subdivided. A structure element with the type of **Div** shall inherit the containment requirements and limitations of its parent element. Where the parent element is itself a structure element of type **Div**, then the inheritance shall recurse to the first parent element whose type is not **Div**. NOTE 3 **Div** does not change the semantic level of the structure hierarchy. EXAMPLE 3 The following is just one of the possible types of content that could be marked as **Div**: • associating a language attribute to group of content items without implying any semantic aspects
Aside	Grouping	*(PDF 2.0)* Encloses content that is distinct from other content within its parent structure element. EXAMPLE 4 The following are just a few of the possible types of content that could be marked as **Aside**: • Callout elements • Sidebar elements • Commentary accompanying an article • Background information for a section in a text book
NonStruct	Grouping	(Non-structural 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 should not be interpreted or exported to other document formats. Its descendants shall be processed normally. A structure element with the type of **NonStruct** shall inherit the containment requirements and limitations of its parent element. Where the parent element is itself a structure element of type **NonStruct**, then the inheritance shall recurse to the first parent element whose type is not **NonStruct**.

NOTE (2020) This document updated the definition of **Part** and returned **Sect** and **NonStruct** to the above table with revised definitions.

14.8.4.5 Block level structure types

This subclause describes structure types for paragraph-like elements, as detailed in "Table 366 — Block level structure types", that consist of running text and other content laid out in the form of conventional paragraphs (as opposed to more specialised types of content such as lists and tables).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 366 — Block level structure types

Structure Type	Category	Description
P	Block	*(Paragraph)* A low-level division of content. Although in many cases it will be used for paragraphs it may enclose any low-level division of content. EXAMPLE The following is just one of the possible types of content that could be marked as **P**: • A paragraph in a newspaper article or a paragraph in the chapter of a novel.
H*n*	Block	*(With n being a sequence of digits representing an unsigned integer greater than or equal to 1)* Encloses a low-level division of content usually referred to as a heading. NOTE 1 Low-level content would typically be perceived as a sub-section of a document, whether a section heading, chapter or any other identifiable subdivision of content within a logical document. See the **Title** element regarding high-level division of content. Any such heading structure element shall always consist of the uppercase letter "H" and one or more digits, representing an unsigned integer greater than or equal to 1, without leading zeroes or any other prefix or postfix. The heading level is indicated through the chosen structure element type, for example **H1** indicates a heading at the topmost level within a document, **H2** a heading at the second level, and so forth. NOTE 2 This implies that **H7** can be used to indicate a heading on the seventh level, whereas h7, H07, H-7 or H_7 cannot be used as heading structure elements. The heading level indicated by a heading should reflect the heading level of the tagged content. NOTE 3 The **Lbl** structure element can be used to enclose section enumeration content or its functional equivalent present inside the heading. EXAMPLE A section heading in a text book or newspaper article is one example where a heading level would be indicated.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Structure Type	Category	Description
H	Block	Encloses a low-level division of content usually referred to as a *heading*. The heading level is not indicated through this structure element, but instead is derived from the nesting of the logical structure within a given **Document** or **DocumentFragment** structure element. The **H** heading structure elements should always be the first structure element within its parent structure element. The **H** heading structure element shall always be the only heading structure element within its parent structure element. NOTE 1 This implies that within a given logical document it is not acceptable that an **H** heading structure element is used if a **H1**, **H2**, **H3** etc. heading structure element is also used inside the same **DocumentFragment** structure element. NOTE 2 The **Lbl** structure element can be used to enclose a section number or similar present inside the heading. EXAMPLE 1 As the use of the **H** heading structure element requires strict document structuring it is typically used only for machine generated documents, documents created from a well structured content data set or documents whose creation is fully controlled by a program such that its structure is strictly guaranteed. EXAMPLE 2 Scientific publications and technical specifications often follow very strict structuring rules and thus are suitable candidates for use of the **H** heading structure element.
Title	Grouping or Block	*(PDF 2.0)* Encloses content that is usually referred to as the title of a document or high-level division of content. NOTE 3 High-level content would typically be perceived as the title of an article, section, chapter or any other identifiable top-level subdivision of content within a logical document. See the **Hn** and **H** elements regarding low-level division of content. It should occur only once inside the parent grouping structure element, and it should occur at or near the beginning of the content inside that grouping structure element. EXAMPLE 3 The title of a book, brochure or leaflet are some types of content that can be marked with the **Title** structure element.
FENote	Grouping, Block or Inline	*(PDF 2.0)* Used to markup footnotes and endnotes. Footnotes and endnotes are content that is not normally read as part of the enclosing content from which it is referenced, but rather consulted at the reading person's discretion. In order for text to be considered a footnote or endnote, there should be a reference from the enclosing content to the footnote or endnote. Such reference may be achieved by means of a **Link** structure element through a structure destination in its link annotation (see "Table 368 — General inline level structure types"), or use of **Ref** in structure elements (see "Table 355 — Entries in a structure element dictionary"). NOTE 4 Text that is labelled as a note – like this paragraph –but is intended to be normally read together with the enclosing content is not a footnote or endnote.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.6 Sub-block level structure type

"Table 367 — Sub-block level structure type" defines a structure type similar to block level types but intended for use in an inline context. An example of this usage is provided in H.8.4, "Example of Sub standard structure type".

Table 367 — Sub-block level structure type

Structure Type	Category	Description
Sub	Inline	*(PDF 2.0)* (Sub-division of a block level element) Encloses content typically perceived as a sub-division inside a block level structure element. A **Lbl** structure element may be used inside a **Sub** structure element to enclose a line number, verse number or similar, if present. If a **Sub** structure element is used, all other content inside the same block level element that is the parent of the **Sub** structure element, should also be enclosed in **Sub** structure elements. Examples: • a verse in a poem or sacred scripture • a line-numbered document • a line of source code • a line in a postal address

14.8.4.7 Inline level structure types

14.8.4.7.1 General

Unless otherwise noted in 14.8.4.2, “Nesting of standard structure elements”, any inline structure element is optional. It may occur once or more than once inside its parent structure element. The other children of its parent structure element, if any are present, may be other inline structure elements or actual content. Inline structure elements and portions of actual content inside a parent structure element may occur in any combination and in any order.

Unless restricted by their type, structure elements and portions of actual content inside a parent structure element may occur in any combination and in any order.

14.8.4.7.2 General inline level structure types

"Table 368 — General inline level structure types" defines standard structure types for inline level structure types.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 368 — General inline level structure types

Structure Type	Category	Description
Lbl	Inline	(Label) Encloses content that distinguishes it from other content inside the same parent element. In a list item (see 14.8.4, "Standard structure types") the **Lbl** structure element may enclose the bullet for list items in unordered lists or digits and characters used for numbering of list items in ordered lists. For headings it may be the number of the chapter. For a definition list it may enclose the term to be defined. For key value pairs it may enclose the key for which a value is provided. For an entry in a table of contents there may be two **Lbl** structure elements, one for a chapter number, and one for the page number at which the chapter starts, whereas the actual text for the chapter heading is the remaining portion of the table of contents list item. EXAMPLE 1 The following are just a few of the possible types of content that could be marked as **Lbl**: • bullets for list items in an unordered list • digits or characters used for numbering list items in an ordered list • the number in the **Caption** for a **Figure** or **Table** • in a form, the label of a form field • in footnotes, the number or symbol matching the reference from the text • in headings, the enumeration of the headings • in a dictionary, the word for which a translation is provided • in a definition list, the term for which a definition is provided • in key value pairs, the key for which a value is provided • in a question and answer sequence in an interview, visual cues for designating questions and answer
Span	Inline	A generic inline portion of content having no particular inherent characteristics. It can be used, for example, to delimit a range of text with certain attributes. EXAMPLE 4 A word inside a sentence is in a different language than the surrounding text, and is contained in a **Span** with a **Lang** attribute indicating the applicable language. EXAMPLE 5 A custom set of structure element types defines custom inline structure elements which are mapped to the **Span** structure element in the **RoleMap** to enable a PDF processor unaware of the custom set of structure element types to essentially ignore the structure element.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Structure Type	Category	Description
Em	Inline	*(PDF 2.0)* (Emphasis) Encloses content for the purpose of emphasis. The level of stress that a particular piece of content has is given by its number of ancestor **Em** structure elements. The placement of stress emphasis changes the meaning of the sentence. The element thus forms an integral part of the content. The precise way in which stress is used in this way depends on the language. EXAMPLE 2 These examples show how changing the stress emphasis changes the meaning. First, a general statement of fact, with no stress: <P>Cats are cute animals.</P> By emphasizing the first word, the statement implies that the kind of animal under discussion is in question (maybe someone is asserting that dogs are cute): <P><Em>Cats</Em> are cute animals.</P> By moving it to the adjective, the exact nature of the cats is reasserted (maybe someone suggested cats were mean animals): <P>Cats are <Em>cute</Em> animals.</P>
Strong	Inline	*(PDF 2.0)* Encloses content for the purpose of strong importance, seriousness or urgency for its contents. EXAMPLE 3 In this example the **Strong** element is used to denote the content that the user is intended to read first: <P>Your tasks for today:</P> <L> <LI><LBody><Strong>Turn off the oven.</Strong></LBody></LI> <LI><LBody>Put out the trash.</LBody></LI> <LI><LBody>Do the laundry.</LBody></LI> </L>
Link	Grouping, Block or Inline	An association between content enclosed by the **Link** structure element and a corresponding link annotation (see 12.5.6.5, "Link annotations").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Structure Type	Category	Description
Annot	Grouping, Block or Inline	Either an association between the content enclosed by the **Annot** structure element and one or more corresponding PDF annotations (see 12.5, "Annotations"), or a mechanism to include one or more PDF annotations in the structure tree. If more than one annotation is referenced in an **Annot** structure element, they shall be of the same annotation type. **Annot** shall not be used for link annotations (see the **Link** structure element) or widget annotations (see the **Form** structure element). All other annotation types may be referenced by this structure element. EXAMPLE 6 The following are just a couple of the possible types of content that could be marked as **Annot**: • Markup annotations indicating change requests like requests for deletions, modifications or insertions. • Highlighting of certain content.
Form	Grouping, Block or Inline	Either an association between content enclosed by the Form structure element and a corresponding widget annotation or a mechanism to include a widget annotation in the structure tree. In a tagged PDF, **Form** shall be used for each PDF widget annotation that belongs to the real content of the document. NOTE **Form** structure elements often include **Lbl** structure elements to mark up a form field's label (if any). EXAMPLE 7 The following are the possible types of content that could be marked as **Form**: • Form fields in a PDF containing a fillable form would be marked as **Form** structure element. • Non-interactive forms, that is, content enclosed in a structure element with the PrintField attribute, would be marked with **Form** structure elements.

NOTE 1 (2020) This document redefined available categories for **Link** and **Annot** structure elements types.

Tagged PDF link elements (standard 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) if **A**, **Dest** and **PA** keys of all of them have identical values
- One object reference (see 14.7.5.3, "PDF objects as content items") to one link annotation associated with the content

NOTE 2 An **SD** entry in the **GoTo** or **GoToR** action in a **Link** annotation facilitates linking directly to a target structure element as opposed to just targeting an area on a page.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.7.3 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 369 — Ruby and Warichu structure types" defines standard structure types for Ruby and warichu text.

Table 369 — Ruby and Warichu structure types

Structure Type	Category	Description
Ruby	Inline	Structure element that wraps around an entire ruby assembly. It shall contain one **RB** element followed by either an **RT** element or a three-element sequence consisting of **RP**, **RT**, and **RP**. **Ruby** structure elements and their content elements shall not break across multiple lines.
RB	Inline	(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	Inline	(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.
RP	Inline	(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 PARENTHESIS or RIGHT PARENTHESIS or similar bracketing character).
Warichu	Inline	(Warichu) The wrapper around the entire warichu assembly. It shall contain a three-element sequence consisting of **WP**, **WT**, and **WP**. **Warichu** structure elements (and their child structure elements) may wrap across multiple lines, according to the warichu breaking rules described in the Japanese Industrial Standard (JIS) X 4051-2004.
WT	Inline	(Warichu text) The smaller-size text of a warichu comment that is formatted into two lines and placed between surrounding **WP** elements.
WP	Inline	(Warichu punctuation) The punctuation that surrounds the content in the **WT** structure element. It typically contains text (usually a single LEFT PARENTHESIS or RIGHT PARENTHESIS or similar bracketing character). According to JIS X 4051-2004, the parentheses surrounding a warichu may be converted to a SPACE (nominally 1/4 EM in width) at the discretion of the formatter.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.8 Other structure types

14.8.4.8.1 General

There are a number of other structure types that have a distinct internal structure on their own, or that may be used as grouping level structure elements, as block level structure elements or as inline level structure elements, depending on the context in which they are used.

14.8.4.8.2 Standard structure types for lists

This clause describes structure types for organising the content of lists. H.8.3, "Hierarchical lists" provides an example of hierarchical list entries. "Table 370 — List standard structure types" defines standard structure types for lists.

Table 370 — List standard structure types

Structure Type	Category	Description
L	Block or Inline	(List) Encloses content consisting of a sequence of items that are semantically related with each other. If a **Caption** is present, it shall be either the first or last child in the **L** (list) structure element. The **ListNumbering** attribute (see 14.8.5.5, "List attribute") may be used to indicate the type of ordered or unordered list. The **ContinuedList** and **ContinuedFrom** attributes (see 14.8.5.5, "List attributes") may be used to indicate relationships with other **L** (list) structure elements. EXAMPLE 1 Bulleted lists, numbered lists, tables of contents, indexes, dictionaries, and lists of key value pairs are all examples of content that would use the **L** structure element.
LI	Internal to **L** (List) structure elements	(List Item) Encloses content for an individual member of a list. Children (see Annex L, "Parent-child relationships between the standard structure elements in the standard structure namespace for PDF 2.0") of the list item may occur in any order or combination. NOTE **LI** structure elements often include **Lbl** structure elements (see Table 368 — General inline level structure types) to mark up a list item's label (if any).
LBody	Internal to **LI** (List item) structure elements	(List Item Body) Encloses the actual content of a list item. EXAMPLE 2 In a dictionary list the list item body contains the translation or definition of the term.

To represent *hierarchical lists*, the child **L** (list) structure elements shall be a direct child of its parent **L** (list) structure element or contained within a **Div** structure element belonging to the parent **L** (list).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE Lists can occur within the content of an **LBody** structure element. Such lists are not considered part of the hierarchy.

14.8.4.8.3 Table structure types

"Table 371 — Table standard structure types" defines standard structure types for tables.

Table 371 — Table standard structure types

Structure Type	Category	Description
Table	Block	A two-dimensional logical structure of cells, possibly including a complex substructure. If a **Caption** is present, it shall be either the first or last child of the **Table** structure element.
TR	Internal to a **Table** structure	A row of table header cells (**TH**) or table data cells (**TD**) in a table.
TH	Internal to a **Table** structure	A table header cell containing content describing one or more rows, columns or rows and columns of the table. The following attributes can be used with the **TH** structure element: • **RowSpan** • **ColSpan** • **Headers** • **Scope** • **Short**
TD	Internal to a **Table** structure	A table cell containing content that is part of the table's content. The following attributes can be used with the **TD** structure element: • **RowSpan** • **ColSpan** • **Headers**
THead	Internal to a **Table** structure	*(Optional)* A group of **TR** structure elements that constitute the header of a table. The **THead** structure element is optional insofar as when rows of header cells are present they may, but are not required to be, so enclosed.
TBody	Internal to a **Table** structure	*(Optional)* A group of **TR** structure elements that constitute the main body portion of a table. The **TBody** structure element is optional insofar as when rows of cells are present they may, but are not required to be, so enclosed.
TFoot	Internal to a **Table** structure	*(Optional)* A group of **TR** structure elements that constitute the footer of a table. The **TFoot** structure element is optional insofar as when rows of cells belonging to footer row(s) are present they may, but are not required to be, so enclosed.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

If the **Headers** attribute (see 14.8.5, "Standard structure attributes") is not specified, any cell in a table may have multiple headers associated with it. These headers are defined either explicitly by the **Headers** attribute, or implicitly, by the following algorithm:

To find headers for any data or header cell, begin from the current cell position and use the current value of **WritingMode** to search towards the first cell in the appropriate horizontal/vertical direction. The search terminates when any of these conditions is reached:

- the edge of the table is reached
- a data cell is found after a header cell
- a header cell has the **Headers** attribute set — the headers that are specified are appended to the row/ column list that is being built

When a header cell is found in the search and the (implicit or explicit) **Scope** attribute of the header cell is either *Both* or *Row/Column*, the header cell is appended to the end of the list of row/column headers, resulting in a list of headers ordered from most specific to most general.

NOTE This algorithm works for languages with different intrinsic directionality of the script (such as right-to-left) because the structure always reflects the logical content order of the table.

14.8.4.8.4 Caption structure type

The standard structure type **Caption**, defined in "Table 372 — Standard structure type Caption", encloses content that serves as a caption for tables, lists, images, formulas, media objects or other types of content.

Table 372 — Standard structure type Caption

Structure Type	Category	Description
Caption	Grouping or Block	For lists and tables a **Caption** structure element may be used as defined for the **L** (list) and **Table** structure elements. In addition a **Caption** may be used for a structure element or several structure elements. A structure element is understood to be "captioned" when a **Caption** structure element exists as an immediate child of that structure element. The **Caption** shall be the first or the last structure element inside its parent structure element. The number of captions cannot exceed 1. While captions are often used with figures or formulas, they may be associated with any type of content. NOTE In principle, captions can appear in a nested fashion. For example, several smaller images belonging to a group of images can each be accompanied by a caption, and the group of these images as a whole is accompanied by a caption as well. EXAMPLE The following are just a few of the possible types of content that could be marked as **Caption**: • Caption for an image, list, table, or formula • Caption for a group of images or a sequence of graphics • Group of images with a caption, plus a caption for each of the images inside the group of images

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.4.8.5 Figure structure type

"Table 373 — Standard structure type Figure" defines the **Figure** standard structure type.

The standard structure type **Figure** encloses content that represents 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").

A **Figure** element may have logical substructure, including other **Figure** elements. For repurposing purposes it may be treated as visually static, without examining its internal contents. It should have a **BBox** attribute (see 14.8.5, "Standard structure attributes").

For repurposing and accessibility purposes, a **Figure** element should have either an **Alt** entry or an **ActualText** entry in its structure element dictionary (see 14.9.3, "Alternate descriptions" and 14.9.4, "Replacement text").

NOTE **Alt** is a description of the graphics objects enclosed by the **Figure** element, whereas **ActualText** gives the exact text equivalent of a graphical illustration that has the appearance of text.

Table 373 — Standard structure type Figure

Structure Type	Category	Description
Figure	Grouping, Block or Inline	Encloses graphical content. The **BBox** attribute (see 14.8.5, "Standard structure attributes") should be present for a figure appearing in its entirety on a single page to indicate the area of the figure on the page. EXAMPLE 1 Some examples of content that would be marked as **Figure** include: • an image • a drawing • a chart, including the text that denotes values on each axis

14.8.4.8.6 Formula structure type

"Table 374 — Standard structure type Formula" defines the **Formula** standard structure type.

The standard structure type **Formula** shall not appear between the **BT** and **ET** operators delimiting a text object (see 9.4, "Text objects").

A **Formula** element may have logical substructure, including other **Formula** elements. For repurposing purposes it may be treated as visually static, without examining its internal contents. It should have a **BBox** attribute (see 14.8.5, "Standard structure attributes").

For repurposing and accessibility purposes, a **Formula** element should have either an **Alt** entry or an **ActualText** entry in its structure element dictionary (see 14.9.3, "Alternate descriptions" and 14.9.4, "Replacement text").

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

NOTE **Alt** is a description of the content enclosed by the **Formula** element, whereas **ActualText** gives the exact text equivalent of a formula has the appearance of text.

Table 374 — Standard structure type Formula

Structure Type	Category	Description
Formula	Grouping, Block or Inline	Encloses a formula. The **BBox** attribute ("see 14.8.5, "Standard structure attributes") should be present for a formula appearing in its entirety on a single page to indicate the area of the formula on the page. EXAMPLE Some examples of content that would be marked as Formula include: • a mathematical equation or a part thereof • a chemical formula • a mathematical proof

14.8.4.8.7 Artifact structure type

"Table 375 — standard structure type Artifact" defines the **Artifact** standard structure type.

Table 375 — standard structure type Artifact

Structure Type	Category	Description
Artifact	Grouping, Block or Inline	*(PDF 2.0)* Encloses content for which semantics require a reference in the structure tree even when such content is not part of the document's real content. The **Artifact** structure type may be used to enclose content that would not otherwise be tagged based on the rules of tagged PDF. A processor of tagged PDF should normally ignore any content items and structure elements that are direct or indirect descendants of an **Artifact** structure element. EXAMPLE Some documents include pages with line numbers. Whereas for tagged PDF such line numbers would typically be considered artifacts, the **Artifact** structure element allows authors to ensure context is maintained by making it possible to place the line numbers in the logical structure without forcing end users to consume them as part of the logical content order.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

14.8.5 Standard structure attributes

14.8.5.1 General

In addition to the standard structure types, PDF defines standard structure attributes for standard structure elements in addition to those attributes in a structure element dictionary that are already defined in "Table 355 — Entries in a structure element dictionary".

The example in 14.7.7, "Example of logical structure" illustrates the use of standard structure attributes.

As discussed in 14.7.6, "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, "Repurposing and 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 355 — Entries in a structure element dictionary");
- As entries in property lists attached to marked-content sequences with the tag **Span** (see 14.6, "Marked content").

14.8.5.2 Standard attribute owners

Each attribute object has an owner, specified by the object's **O** entry, or, if the value of **O** is *NSO*, by the object's **NS** 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 as shown in "Table 376 — Standard structure attribute owners".

Table 376 — Standard structure attribute owners

Owner value for the attribute object's O entry	Description
Layout	Attributes governing the layout of content
List	Attributes governing the numbering of lists
PrintField	Attributes governing Form structure elements for non-interactive form fields

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Owner value for the attribute object's O entry	Description
Table	Attributes governing the organisation of cells in tables
Artifact	Attributes governing Artifact structure elements
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 4.0
HTML-5.00	Additional attributes governing translation to HTML, version 5.0
OEB-1.00	Additional attributes governing translation to OEB (Open eBook), version 1.0
RTF-1.05	Additional attributes governing translation to Microsoft Rich Text Format, version 1.05
CSS-1	Additional attributes governing translation to a format using CSS, version 1
CSS-2	Additional attributes governing translation to a format using CSS, version 2.1
CSS-3	Additional attributes governing translation to a format using CSS, version 3
RDFa-1.10	Additional attributes governing translation to a format using RDFa version 1.1
ARIA-1.1	Additional attributes governing translation to a format using WAI-ARIA version 1.1

NOTE (2020) The three owner values for CSS were changed in the above table to better reflect CSS numbering.

An attribute object owned by a specific export format, such as XML-1.00, shall be applied only when processing PDF content based on that format. Such format-specific attributes shall override any corresponding attributes owned by **Layout, List, PrintField, Table** or **Artifact**. 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 subclause 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 PDF processor determines an attribute's value to be the first item in the following list that applies:

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

- The value of the attribute specified in the element's **A** entry, owned by an owner as specified by the **O** entry, or, if the value of the **O** entry is *NSO*, the **NS** entry, excluding **Layout, PrintField, Table**, **List** and **Artifact**, if present, and if processing based on the format indicated by the owner value
- The value of the attribute specified in the element's **A** entry, owned by **Layout, PrintField, Table, List** or **Artifact**, if present
- The value of the attribute specified in a class map associated with the element's **C** entry, if there is one
- The resolved value of the parent structure element, if the attribute is inheritable
- The default value for the attribute, if there is one

NOTE 2 The properties **Lang, Alt, ActualText**, and **E** do not appear in attribute dictionaries. The rules governing their application are discussed in 14.9, "Repurposing and 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 whose owner is any other owner excluding **List, Table, PrintField** and **Artifact**.

NOTE 1 The intent is that these parameters can be used to repurpose the content or export it to some other document format with at least basic styling preserved.

"Table 377 — Standard layout attributes" summarizes the standard layout attributes and the structure elements to which they apply.

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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

Table 377 — Standard layout attributes

Structure Elements	Attributes	Inheritable
Any structure element	Placement	No
	WritingMode	Yes
	BackgroundColor	No
	BorderColor	Yes
	BorderStyle	No

© ISO 2020 – All rights reserved

Structure Elements	Attributes	Inheritable
	BorderThickness	Yes
	Color	Yes
	Padding	No
Any BLSE; ILSEs with Placement other than Inline	SpaceBefore	No
	SpaceAfter	No
	StartIndent	Yes
	EndIndent	Yes
BLSEs containing text	TextIndent	Yes
	TextAlign	Yes
Figure, Form, Formula and Table elements	BBox	No
	Width	No
	Height	No
TH (Table header); TD (Table data)	Width	No
	Height	No
	BlockAlign	Yes
	InlineAlign	Yes
	TBorderStyle	Yes
	TPadding	Yes
Any ILSE; BLSEs containing ILSEs or containing direct or nested content items	LineHeight	Yes
	BaselineShift	No
	TextDecorationType	Yes, only for directly nested ILSEs
	TextPosition	Yes
	TextDecorationColor	Yes
	TextDecorationThickness	Yes
Grouping elements	ColumnCount	No
	ColumnWidths	No
	ColumnGap	No
Vertical text	GlyphOrientationVertical	Yes

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Structure Elements	Attributes	Inheritable
Ruby text	RubyAlign	Yes
	RubyPosition	Yes

NOTE 2 TextPosition was corrected in the above table (2020).

14.8.5.4.2 General layout attributes

The layout attributes described in "Table 378 — Standard layout attributes common to all standard structure types" may apply to structure elements of any of the standard at the block level (BLSEs) or the inline level (ILSEs).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 378 — 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. The value shall be one of the following: *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 14.8.5.4.5, "Content and Allocation Rectangles") 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 14.8.5.4.5, "Content and Allocation Rectangles") 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 14.8.5.4.5, "Content and Allocation Rectangles") 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: *Block* for BLSEs, *Inline* for ILSEs. 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 PDF processors. Tagged PDF merely identifies the elements as floating and indicates their desired placement.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
WritingMode	name	*(Optional; inheritable)* Indicates the directions of layout progression inside Block Level Structure Elements (BLSEs) (inline progression) and regarding the sequence of BLSEs (block progression). **WritingMode** may be used as an attribute for any structure element. The value shall be one of the following: *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. *TbLr* Inline progression from top to bottom; block progression from left to right. This is the typical writing mode for writing systems like classical Mongolian. *LrBt* Inline progression from left to right; block progression from bottom to top. There is currently no known writing system to which this writing mode applies. *RlBt* Inline progression from right to left; block progression from bottom to top. There is currently no known writing system to which this writing mode applies. *BtRl* Inline progression from bottom to top; block progression from right to left. This is the typical writing mode for the Ancient Berber writing system. *BtLr* Inline progression from bottom to top; block progression from left to right. This is the typical writing mode for the Batak writing system. The specified layout directions shall apply to the given structure element and all of its descendants. Default value: *LrTb*. For elements that are represented in 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 types **TH** and **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, Unicode Bidirectional Algorithm*, available from the Unicode Consortium.
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 its background were transparent.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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.
BorderStyle	name or array	*(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: • 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. • A name from the list below representing the border style to apply to all four edges. Valid values are: *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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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. Default value: *0*.
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 14.8.5.4.5, "Content and Allocation Rectangles"). A positive value enlarges the background area; a negative value trims it, possibly allowing the border to overlap the element's text or graphic. There are two forms: • A number representing the width of the padding for all four edges. • An array of four entries, each entry specifying the width of the padding for one edge, in the order of the before, after, start and end edges. 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 379 — Additional standard layout attributes specific to block-level structure elements" describes layout attributes that shall apply only to block-level structure elements (BLSEs).

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Table 379 — 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 14.8.5.4.4, "Layout Attributes for ILSEs"). 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.
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 14.8.5.4.2, "General 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 14.8.5.4.2, "General 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
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 **LI** (List item), **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. Valid values are: *Start* Aligned with the start edge. *Center* Centred 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*.
BBox	rectangle	*(Optional; 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 structure element's bounding box (the rectangle that completely encloses its visible content). The **BBox** attribute should be present for structure elements whose content does not lend itself to reflow or any other visual rearrangement of the content inside it. NOTE 1 Examples of types of structure elements that do not lend themselves to reflow include **Figure** and **Formula** structure elements. NOTE 2 The semantics of the visual presentation of charts, illustrations consisting of more than one graphics object, or formulas can suffer if the objects inside them are rearranged, as is typical for content reflow. A structure element with a **BBox** attribute may contain other structure elements inside it. NOTE 3 A formula, for example, can lose its meaning if the parts in the formula are visually rearranged. At the same time, the parts inside the formula could be individually tagged, for example with inline level structure elements. EXAMPLE Formulas, graphic art, vector drawings, images are types of structure elements for which a **BBox** attribute is appropriate.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
Width	number or name	*(Optional; not inheritable; illustrations, tables, and table cells only; should be used for table cells; sometimes required for **Figure**, **Form** or **Formula** elements with **Placement** attribute)* The width of the element's content rectangle (see 14.8.5.4.5, "Content and Allocation Rectangles"), measured in default user space units in the inline-progression direction. This attribute shall apply only to elements of structure type **Figure**, **Formula**, **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 is 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; sometimes required for **Figure**, **Form** or **Formula** elements with **Placement** attribute)* The height of the element's content rectangle (see 14.8.5.4.5, "Content and Allocation Rectangles"), measured in default user space units in the block-progression direction. This attribute shall apply only to elements of structure type **Figure**, **Formula**, **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 is 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. Valid values are: *Before* Before edge of the first child's allocation rectangle aligned with that of the table cell's content rectangle. *Middle* Children centred 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 14.8.5.4.5, "Content and Allocation Rectangles") shall become the reference area for all of its descendants. Default value: *Before*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
InlineAlign	name	*(Optional; inheritable; table cells only)* The alignment, in the inline-progression direction, of content within the table cell. Valid values are: *Start* Start edge of each child's allocation rectangle aligned with that of the table cell's content rectangle. *Center* Each child centred 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 ILSEs that are children of the given element. The table cell's content rectangle (see 14.8.5.4.5, "Content and Allocation Rectangles") 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 379 — Additional standard layout attributes specific to block-level structure elements"). 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 14.8.5.4.5, "Content and Allocation Rectangles"). 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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

14.8.5.4.4 Layout Attributes for ILSEs

The attributes described in "Table 380 — Standard layout attributes specific to inline-level structure elements" 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.

© ISO 2020 – All rights reserved

Table 380 — 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 of text is 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 non-zero value specified for **BaselineShift**. *Auto* Adjustment for the value of **BaselineShift** shall not be made. Default value: *Normal*. The meaning of the term "reasonable value" is left to the PDF processor to determine. It should be approximately 1.2 times the font size, but this value may vary depending on the export format. 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. 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
TextPosition	name	*(Optional; inheritable; PDF 2.0)* The position of the element relative the immediately surrounding content. Valid values are: *Sup* Position is elevated, like for superscript. *Sub* Position is lowered, like for subscript *Normal* Position is neither elevated nor lowered. Default value: *Normal* **TextPosition** does not imply any specific semantic. NOTE 2 As a consequence, it cannot be determined whether text with a **TextPosition** attribute of *Sup* is a footnote number, an exponent, an index or some other use of the text. For mathematical expressions, MathML structure elements provide a richer semantic for superscripted or subscripted content.
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 text decoration colour for this element shall be the current fill colour in effect at the start of its associated content.
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; inheritable only for directly nested ILSEs)* The text decoration, if any, to be applied to the element's text. Valid values are: *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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
RubyAlign	name	*(Optional; inheritable; PDF 1.5)* The justification of the lines within a ruby assembly. Valid values are: *Start* The content shall be aligned on the start edge in the inline-progression direction. *Center* The content shall be centred 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. For example, if the **RT** element has a shorter width than the **RB** element, the **RT** element shall be aligned as specified in its **RubyAlign** attribute.
RubyPosition	name	(*Optional; inheritable; PDF 1.5*) The placement of the **RT** structure element relative to the **RB** element in a ruby assembly. Valid values are: *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*.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved

Key	Type	Value
GlyphOrientationVertical	number or name	*(Optional; inheritable; PDF 1.5)* Specifies the orientation of glyphs when the inline-progression direction is top to bottom or bottom to top. Valid values are: *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. *Auto* Specifies 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 3 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 is 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 other than **TH** and **TD**, 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.

STANDARDSISO.COM : Click to view the full PDF of ISO 32000-2:2020

© ISO 2020 – All rights reserved