PDF

From Wikipedia, the free encyclopedia

For other uses, see PDF (disambiguation).

The **Portable Document Format** (**PDF**) is a file format developed by Adobe in 1993 to present documents, including text formatting and images, in a manner independent of application software, hardware, and operating systems.[2][3] Based on the PostScript language, each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, vector graphics, raster images and other information needed to display it. PDF was standardized as ISO 32000 in 2008, and no longer requires any royalties for its implementation.[4]

PDF files may contain a variety of content besides flat text and graphics including logical structuring elements, interactive elements such as annotations and form-fields, layers, rich media (including video content) and three dimensional objects using U3D or PRC, and various other data formats. The PDF specification also provides for encryption and digital signatures, file attachments and metadata to enable workflows requiring these features.

History and standardization[edit]

Main article: History of the Portable Document Format (PDF)

Adobe Systems made the PDF specification available free of charge in 1993. In the early years PDF was popular mainly in desktop publishing workflows, and competed with a variety of formats such as DjVu, Envoy, Common Ground Digital Paper, Farallon Replica and even Adobe's own PostScript format.

PDF was a proprietary format controlled by Adobe until it was released as an open standard on July 1, 2008, and published by the International Organization for Standardization as ISO 32000-1:2008,[5][6] at which time control of the specification passed to an ISO Committee of volunteer industry experts. In 2008, Adobe published a Public Patent License to ISO 32000-1 granting royalty-free rights for all patents owned by Adobe that are necessary to make, use, sell, and distribute PDF-compliant implementations.[7]

PDF 1.7, the sixth edition of the PDF specification that became ISO 32000-1, includes some proprietary technologies defined only by Adobe, such as Adobe XML Forms Architecture (XFA) and JavaScript extension for Acrobat, which are referenced by ISO 32000-1 as normative and indispensable for the full implementation of the ISO 32000-1 specification. These proprietary technologies are not standardized and their specification is published only on Adobe's website.[8][9][10][11][12] Many of them are also not supported by popular third-party implementations of PDF.

On July 28, 2017, ISO 32000-2:2017 (PDF 2.0) was published.[13] ISO 32000-2 does not include any proprietary technologies as normative references.[14]

Technical foundations[edit]

The PDF combines three technologies:

- A subset of the PostScript page description programming language, for generating the layout and graphics.
- A font-embedding/replacement system to allow fonts to travel with the documents.
- A structured storage system to bundle these elements and any associated content into a single file, with data compression where appropriate.

PostScript[edit]

PostScript is a page description language run in an interpreter to generate an image, a process requiring many resources. It can handle graphics and standard features of programming languages such as `if` and `loop` commands. PDF is largely based on PostScript but simplified to remove flow control features like these, while graphics commands such as `lineto` remain.

Often, the PostScript-like PDF code is generated from a source PostScript file. The graphics commands that are output by the PostScript code are collected and tokenized. Any files, graphics, or fonts to which the document refers also are collected. Then, everything is compressed to a single file. Therefore, the entire PostScript world (fonts, layout, measurements) remains intact.

As a document format, PDF has several advantages over PostScript:

- PDF contains tokenized and interpreted results of the PostScript source code, for direct correspondence between changes to items in the PDF page description and changes to the resulting page appearance.
- PDF (from version 1.4) supports transparent graphics; PostScript does not.
- PostScript is an interpreted programming language with an implicit global state, so instructions accompanying the description of one page can affect the appearance of any following page. Therefore, all preceding pages in a PostScript document must be processed to determine the correct appearance of a given page, whereas each page in a PDF document is unaffected by the others. As a result, PDF viewers allow the user to quickly jump to the final pages of a long document, whereas a PostScript viewer needs to process all pages sequentially before being able to display the destination page (unless the optional PostScript Document Structuring Conventions have been carefully compiled and included).

Technical overview[edit]

File structure[edit]

A PDF file is a 7-bit ASCII file, except for certain elements that may have binary content. A PDF file starts with a header containing the magic number and the version of the format such as `%PDF-1.7`. The format is a subset of a COS ("Carousel" Object Structure) format.[15] A COS tree file consists primarily of *objects*, of which there are eight types:[16]

- Boolean values, representing *true* or *false*
- Numbers
- Strings, enclosed within parentheses (`(...)`), may contain 8-bit characters.
- Names, starting with a forward slash (`/`)
- Arrays, ordered collections of objects enclosed within square brackets (`[...]`)
- Dictionaries, collections of objects indexed by Names enclosed within double pointy brackets (`<<...>>`)
- Streams, usually containing large amounts of data, which can be compressed and binary, between the `stream` and `endstream` keywords, preceded by a dictionary
- The null object

Furthermore, there may be comments, introduced with the percent sign (`%`). Comments may contain 8-bit characters.

Objects may be either *direct* (embedded in another object) or *indirect*. Indirect objects are numbered with an *object number* and a *generation number* and defined between the `obj` and `endobj` keywords if residing in the document root. Beginning with PDF version 1.5, indirect objects (except other streams) may also be located in special streams known as *object streams* (marked `/Type /ObjStm`). This technique enables non-stream objects to have standard stream filters applied to them, reduces the size of files that have large numbers of small indirect objects and is especially useful for *Tagged PDF*. Object streams do not support specifying an object's *generation number* (other than 0).

An index table, also called the cross-reference table, is typically located near the end of the file and gives the byte offset of each indirect object from the start of the file.[17] This design allows for efficient random access to the objects in the file, and also allows for small changes to be made without rewriting the entire file (*incremental update*). Before PDF version 1.5, the table would always be in a special ASCII format, be marked with the `xref` keyword, and follow the main body composed of indirect objects. Version 1.5 introduced optional *cross-reference streams*, which have the form of a standard stream object, possibly with filters applied. Such a stream may be used instead of the ASCII cross-reference table and contains the offsets and other information in binary format. The format is flexible in that it allows for integer width specification (using the `/W` array), so that for example a document not exceeding 64 KiB in size may dedicate only 2 bytes for object offsets

At the end of a PDF file is a footer containing:

- The `startxref` keyword followed by an offset to the start of the cross-reference table (starting with the `xref` keyword) or the cross-reference stream object
- And the `%%EOF` end-of-file marker.

If a cross-reference stream is not being used, the footer is preceded by the `trailer` keyword followed by a dictionary containing information that would otherwise be contained in the cross-reference stream object's dictionary:

- A reference to the root object of the tree structure, also known as the *catalog* (`/Root`)
- The count of indirect objects in the cross-reference table (`/Size`)
- And other optional information.

There are two layouts to the PDF files: non-linear (not "optimized") and linear ("optimized"). Non-linear PDF files consume less disk space than their linear counterparts, though they are slower to access because portions of the data required to assemble pages of the document are scattered throughout the PDF file. Linear PDF files (also called "optimized" or "web optimized" PDF files) are constructed in a manner that enables them to be read in a Web browser plugin without waiting for the entire file to download, since they are written to disk in a linear (as in page order) fashion.[18] PDF files may be optimized using Adobe Acrobat software or QPDF.

Imaging model[edit]

The basic design of how graphics are represented in PDF is very similar to that of PostScript, except for the use of transparency, which was added in PDF 1.4.

PDF graphics use a device-independent Cartesian coordinate system to describe the surface of a page. A PDF page description can use a matrix to scale, rotate, or skew graphical elements. A key concept in PDF is that of the *graphics state*, which is a collection of graphical parameters that may be changed, saved, and restored by a *page description*. PDF has (as of version 1.6) 24 graphics state properties, of which some of the most important are:

- The *current transformation matrix* (CTM), which determines the coordinate system
- The *clipping path*
- The *color space*
- The *alpha constant*, which is a key component of transparency

Vector graphics[edit]

As in PostScript, vector graphics in PDF are constructed with *paths*. Paths are usually composed of lines and cubic Bézier curves, but can also be constructed from the outlines of text. Unlike PostScript, PDF does not allow a single path to mix text outlines with lines and curves. Paths can be stroked, filled, clipping. Strokes and fills can use any color set in the graphics state, including *patterns*.

PDF supports several types of patterns. The simplest is the *tiling pattern* in which a piece of artwork is specified to be drawn repeatedly. This may be a *colored tiling pattern*, with the colors specified in the pattern object, or an *uncolored tiling pattern*, which defers color specification to the time the pattern is drawn. Beginning with PDF 1.3 there is also a *shading pattern*, which draws continuously varying colors. There are seven types of shading pattern of which the simplest are the *axial shade* (Type 2) and *radial shade* (Type 3).

Raster images[edit]

Raster images in PDF (called *Image XObjects*) are represented by dictionaries with an associated stream. The dictionary describes properties of the image, and the stream contains the image data. (Less commonly, a raster image may be embedded directly in a page description as an *inline image*.) Images are typically *filtered* for compression purposes. Image filters supported in PDF include the following general purpose filters:

- **ASCII85Decode**, a filter used to put the stream into 7-bit ASCII,
- **ASCIIHexDecode**, similar to ASCII85Decode but less compact,
- **FlateDecode**, a commonly used filter based on the deflate algorithm defined in RFC 1951 (deflate is also used in the gzip, PNG, and zip file formats among others); introduced in PDF 1.2; it can use one of two groups of predictor functions for more compact zlib/deflate compression: *Predictor 2* from the TIFF 6.0 specification and predictors (filters) from the PNG specification (RFC 2083),
- **LZWDecode**, a filter based on LZW Compression; it can use one of two groups of predictor functions for more compact LZW compression: *Predictor 2* from the TIFF 6.0 specification and predictors (filters) from the PNG specification,
- **RunLengthDecode**, a simple compression method for streams with repetitive data using the run-length encoding algorithm and the image-specific filters,
- **DCTDecode**, a lossy filter based on the JPEG standard,
- **CCITTFaxDecode**, a lossless bi-level (black/white) filter based on the Group 3 or Group 4 CCITT (ITU-T) fax compression standard defined in ITU-T T.4 and T.6,
- **JBIG2Decode**, a lossy or lossless bi-level (black/white) filter based on the JBIG2 standard, introduced in PDF 1.4, and
- **JPXDecode**, a lossy or lossless filter based on the JPEG 2000 standard, introduced in PDF 1.5.

Normally all image content in a PDF is embedded in the file. But PDF allows image data to be stored in external files by the use of *external streams* or *Alternate Images*. Standardized subsets of PDF, including PDF/A and PDF/X, prohibit these features.

Text[edit]

Text in PDF is represented by *text elements* in page content streams. A text element specifies that *characters* should be drawn at certain positions. The characters are specified using the *encoding* of a selected *font resource*.

Fonts[edit]

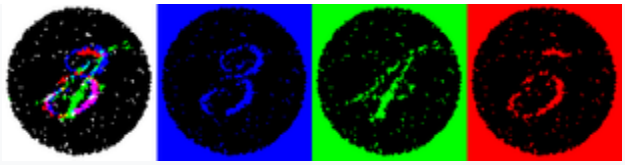
A font object in PDF is a description of a digital typeface. It may either describe the characteristics of a typeface, or it may include an embedded *font file*. The latter case is called an *embedded font* while the former is called an *unembedded font*. The font files that may be embedded are based on widely used standard digital font formats: **Type 1** (and its compressed variant **CFF**), **TrueType**, and (beginning with PDF 1.6) **OpenType**. Additionally PDF supports the **Type 3** variant in which the components of the font are described by PDF graphic operators.

Steganography

From Wikipedia, the free encyclopedia

Jump to navigationJump to search

For the process of writing in shorthand, see Stenography. For the prefix "Stego-" as used in taxonomy, see List of commonly used taxonomic affixes.



The same image viewed by white, blue, green and red lights reveals different hidden numbers.

Steganography (/ˌstɛgəˈnɒgrəfi/ (listen) *STEG-ə-NOG-rə-fee*) is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word *steganography* comes from Greek *steganographia*, which combines the words *steganós* (στεγανός), meaning "covered or concealed", and *-graphia* (γραφή) meaning "writing".[1]

The first recorded use of the term was in 1499 by Johannes Trithemius in his *Steganographia*, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages appear to be (or to be part of) something else: images, articles, shopping lists, or some other cover text. For example, the hidden message may be in invisible ink between the visible lines of a private letter. Some implementations of steganography that lack a shared secret are forms of security through obscurity, and key-dependent steganographic schemes adhere to Kerckhoffs's principle.[2]

The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages, no matter how unbreakable they are, arouse interest and may in themselves be incriminating in countries in which encryption is illegal.[3]

Whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned both with concealing the fact that a secret message is being sent and its contents.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. For example, a sender might start with an innocuous image file and adjust the color of every hundredth pixel to correspond to a letter in the alphabet. The change is so subtle that someone who is not specifically looking for it is unlikely to notice the change.

History[edit]

A chart from Johannes Trithemius's *Steganographia* copied by Dr John Dee in 1591

The first recorded uses of steganography can be traced back to 440 BC in Greece, when Herodotus mentions two examples in his *Histories*.[4] Histiaeus sent a message to his vassal, Aristagoras, by shaving the head of his most trusted servant, "marking" the message onto his scalp, then sending him on his way once his hair had regrown, with the instruction, "When thou art come to Miletus, bid Aristagoras shave thy head, and look thereon." Additionally, Demaratus sent a warning about a forthcoming attack to Greece by writing it directly on the wooden backing of a wax tablet before applying its beeswax surface. Wax tablets were in common use then as reusable writing surfaces, sometimes used for shorthand.

In his work *Polygraphiae,* Johannes Trithemius developed his so-called "Ave-Maria-Cipher" that can hide information in a Latin praise of God. "*Auctor Sapientissimus Conseruans Angelica Deferat Nobis Charitas Potentissimi Creatoris*" for example contains the concealed word *VICIPEDIA*.[5]

Techniques[edit]

Deciphering the code. *Steganographia*

Digital messages

Modern steganography entered the world in 1985 with the advent of personal computers being applied to classical steganography problems.[7] Development following that was very slow, but has since taken off, going by the large number of steganography software available:

- Concealing messages within the lowest bits of noisy images or sound files. A survey and evaluation of relevant literature/techniques on the topic of digital image steganography can be found here.[8]
- Concealing data within encrypted data or within random data. The message to conceal is encrypted, then used to overwrite part of a much larger block of encrypted data or a block of random data (an unbreakable cipher like the one-time pad generates ciphertexts that look perfectly random without the private key).
- Chaffing and winnowing.
- Mimic functions convert one file to have the statistical profile of another. This can thwart statistical methods that help brute-force attacks identify the right solution in a ciphertext-only attack.
- Concealed messages in tampered executable files, exploiting redundancy in the targeted instruction set.
- Pictures embedded in video material (optionally played at slower or faster speed).
- Injecting imperceptible delays to packets sent over the network from the keyboard. Delays in keypresses in some applications (telnet or remote desktop software) can mean a delay in packets, and the delays in the packets can be used to encode data.
- Changing the order of elements in a set.
- Content-Aware Steganography hides information in the semantics a human user assigns to a datagram. These systems offer security against a nonhuman adversary/warden.
- Blog-Steganography. Messages are fractionalized and the (encrypted) pieces are added as comments of orphaned web-logs (or pin boards on social network platforms). In this case the selection of blogs is the symmetric key that sender and recipient are using; the carrier of the hidden message is the whole blogosphere.
- Modifying the echo of a sound file (Echo Steganography).[9]
- Steganography for audio signals.[10]
- Image bit-plane complexity segmentation steganography
- Including data in ignored sections of a file, such as after the logical end of the carrier file[11].
- Adaptive steganography: Skin tone based steganography using a secret embedding angle.[12]
- Embedding data within the control-flow diagram of a program subjected to control flow analysis[13]

History[edit]

In the mid-1980s Xerox pioneered an encoding mechanism for a unique number represented by tiny dots spread over the entire print area. Xerox developed the machine identification code "to assuage fears that their color copiers could be used to counterfeit bills"[4] and received U.S. Patent No 5515451 describing the use of the yellow dots to identify the source of a copied or printed document.[5]

In October 2004, consumers first heard of the hidden feature, when it was used by Dutch authorities to track down counterfeiters who had used a Canon color laser printer.[6] In November 2004, *PC World* reported the machine identification code had been used for decades in some printers, allowing law enforcement to identify and track down counterfeiters.[4] The Central Bank Counterfeit Deterrence Group (CBCDG) has denied that it developed the feature.[5]

In 2005, the civil rights group Electronic Frontier Foundation (EFF) encouraged the public to send in sample printouts and subsequently decoded the pattern.[7] The pattern has been demonstrated on a wide range of printers from different manufacturers and models.[8] The EFF stated in 2015 that the documents that they previously received through the FOIA[9] suggested that *all* major manufacturers of color laser printers entered a secret agreement with governments to ensure that the output of those printers is forensically traceable.[10]

In 2007, the European Parliament was asked about the question of invasion of privacy.[11][5]

Protection of privacy and circumvention[edit]

Copies or printouts of documents with confidential personal information, for example health care information, account statements, tax declaration or balance sheets, can be traced to the owner of the printer and the creation date of the documents can be revealed. This traceability is unknown to many users and inaccessible, as manufacturers do not publicize the code that produces these patterns. It is unclear which data may be unintentionally passed on with a copy or printout. In particular, there are no mentions of the technique in the support materials of most affected printers (exceptions see below). In 2005 Electronic Frontier Foundation (EFF) sought a decoding method and made available a Python script for analysis.[17]

In 2018, scientists from the TU Dresden developed and published a tool to extract and analyze the steganographic codes of a given color printer and subsequently to anonymize prints from that printer. The anonymization works by printing additional yellow dots on top of the Machine Identification Code.[1][2][3] The scientists made the software available in order to support whistleblowers in their efforts to publicize grievances.[18]

Comparable processes[edit]

Other methods of identification are not as easily recognizable as yellow dots. For example, a modulation of laser intensity and a variation of shades of grey in texts are already feasible. As of 2006, it was unknown whether manufacturers were using these techniques.[19]