Adobe Portable Document Format

Inventory of long-term preservation risks

Version : **0.2**
Author : **Johan van der Knijff**
Date : **20-10-2009**

Koninklijke Bibliotheek, national library of the Netherlands
Research & Development Division

I. Revision history

Revision number	Date	Author	Comments
0.1	12-10-2009	JvdK	
0.2	20-10-2009	JvdK	Incorporated comments by Barbara Sierman; added risk overview tables, added section on risk mitigation to introduction; several minor corrections and additions.

II. Related documents

Document name	Date	Author

III. Table of contents

1 Introduction

1.1 Scope and target audience

Within the e-Depot repository of the National Library of the Netherlands (KB), the Adobe Portable Document format is the prevalent file format that is used for storing digital publications. Since the KB's mandate includes ensuring the long-term preservation and accessibility of these materials, identifying potential threats to their long-term availability is of crucial importance. This requires a rather in-depth knowledge on the technical background of the PDF format. To this end, Adobe has provided a series of technical References ([1], [2], [5], [6], [7], [8]). However, these documents are primarily aimed at developers who wish to implement support for the PDF format in their software applications. As a result, the level of technical detail as well as the sheer size of these documents (the PDF 1.7 Reference alone is a 1300+ page document) makes them rather daunting for non-programmers. The aim of the current document is to provide a concise and somewhat more accessible overview of those PDF features that are important from a long-term preservation and accessibility point of view. It is primarily targeted at our KB colleagues who are actively involved in the management and operation of the e-Depot (although it might eventually find its way to a wider audience).

1.2 Outline of this document

Chapter 2 gives an overview of the file structure of a PDF document, the object types that are the building blocks of the format, and the logical document structure these objects are organised into. Chapter 3 discusses some general features of the different PDF versions, such as file identifiers, and how specific PDF versions relate to versions of Adobe's Acrobat Reader. The remaining chapters each focus on a specific theme. For each theme (e.g. 'fonts', 'password-protection', etc.), its relevance to digital preservation and accessibility is explained, and the important features that are associated with it are discussed. The risks that are associated with each theme and the implementation history of all discussed features are summarised in separate tables. This will hopefully facilitate getting an overview of the most important risks and understanding the 'history of PDF'. The final section of each chapter explains how 'risky' features can be identified at the level of objects and object attributes in a file. This section is quite technical. Eventually, we want to expand (or even replace) this section by explaining how to identify these features with standard tools such as JHOVE, JHOVE2 or Adobe's proprietary tools. However, at the time of writing this version of this document, we yet need to investigate whether any of these tools offer all required functionality for identifying these features. If not, the object-level descriptions in the current document could be useful as a basis for feature requests in existing tools, or even for the creation of some (provisional) new tool. The final chapter, which is still empty in this version, will be reserved for describing and documenting specific problems with PDF documents, based on actual experiences from the e-Depot.

1.3 On the mitigation of preservation risks

Since this document focuses on the description and identification of preservation risks, a discussion of measures to mitigate these risks would be a logical next step. This mitigation issue may well be covered in future versions of this document. However, it is important to

make a distinction here between existing materials that are currently stored in the e-Depot, and on the other hand new materials that are submitted by publishers.

For the latter, the KB has already formulated a set of guidelines [21], and publishers are requested to adhere to these as closely as possible. The guidelines will be updated from time to time, and the current document is partially intended as a technical reference on which future updates will be based. The guidelines themselves will however be maintained separately from this document.

The situation is quite different for the existing collection, as it encompasses a wide range of materials, many of which were originally produced at a time when the subject of digital preservation itself was still virtually unknown. As a result, we do not yet have a clear overview of the preservation risks associated with these materials. Therefore, for now the current version of this document focuses on the identification of 'risky' features. Once we have a clearer picture of the actual prevalence of these features in the e-Depot, the next step would be to think of measures to mitigate these risks. However, these measures may turn out to be quite different from the recommendations in the publisher guidelines. In order to avoid any possible mix-up between the two, it might actually be better to keep the discussion of mitigation measures out of the current document altogether, and cover this in a separate volume.

2 Structure of a Portable Document Format file

2.1 Introduction

The aim of this chapter is to provide a brief tour of the Portable Document Format. This will hopefully make it easier to follow the main part of this document. We will first cover the actual file structure; then we give a description of the various types of objects than can exist within a PDF file. Finally, we will briefly outline the logical structure of a PDF document.

2.2 PDF file structure

Upon its creation, a PDF file consists of 4 elements (see Figure 1):

1. A *file header* that specifies the PDF version specification to which the file conforms
2. A *body* which contains all the objects that make up the document's content
3. A *cross-reference table* that serves as an index to the exact byte position where each object is located within the file
4. A *trailer* that specifies the location of some special objects (amongst which the cross-reference table)

When a PDF file is updated, instead of modifying the existing content, data is *added* to the end of the file. As a result of this, multiple instances of the body, cross-reference table and trailer may co-exist in one file.

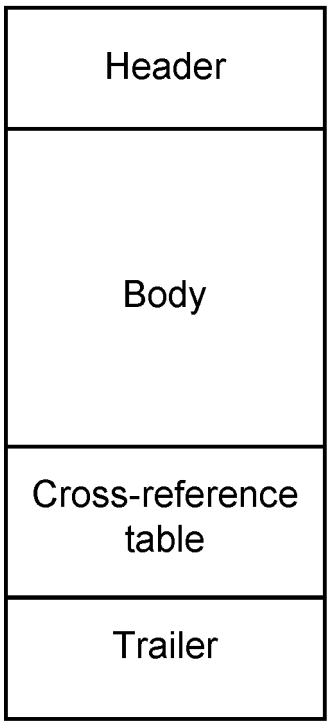

Figure 1 Initial structure of a PDF file

2.2.1 Header

The header of a PDF file is made up of one or two lines. The first line, which is mandatory, defines to which version specification the file conforms. For example, for PDF 1.4 the header line would read:

```
%PDF-1.4
```

Most PDF files also contain a second comment line that contains a sequence of non-printable characters. These characters have no particular meaning, but just serve to tell applications such as e-mail clients and file transfer software that the file's contents should be treated as binary data (as opposed to plain ASCII text).

Note that the '%' character is used in PDF files to define comments: all text between the '%' character and the end of a line is treated as a comment.

2.2.2 Body

The body of a PDF file contains the objects (e.g. text streams, images, fonts, etc.) that make up the document's contents. For a discussion of PDF objects, see section 2.3.

2.2.3 Cross-reference table

The cross-reference table can be thought of as an index that defines, for each object in the file, its exact location. The location is expressed as a byte offset, that is, the number of bytes between the start of the file and the start of the object description within the file. The main reason for having a cross-reference table is that it allows software that reads a PDF file to locate and read objects without having to scan the whole file.

Objects in the cross-reference table are either 'in use' or 'free'. 'Free' objects are essentially obsolete, and should not be used (even though a free object may still be physically present in the file). 'Free' objects can be reactivated (i.e. changed to 'in use' again), and vice versa. The cross-reference table keeps track of the number of times each object has been reactivated (or deleted) through a so-called 'generation number'. For a detailed discussion of the layout of the cross-reference table and its contents please see [1], [2], [5], [6], [7], or [8].

2.2.4 Trailer

The trailer contains the location (byte position) of the cross-reference table, as well as some other special objects.

2.3 PDF objects

This section gives a brief description of the object types that are supported in PDF. There are eight basic types of objects, and a special type of object that allows one object to refer to another object:

2.3.1 Boolean objects

Boolean objects are identified by the keywords 'true' and 'false' (case sensitive!).

2.3.2 Numeric objects

PDF supports numbers as integers and real numbers. Numbers that are in exponential format are not supported by the PDF numeric object.

2.3.3 String objects

A string is a sequence of 8-bit bytes that is used to represent text data. Strings can be defined as either 'literal' strings, which are enclosed in parentheses ((…)), or hexadecimal strings, which are enclosed in angle brackets (<…>). The specification of PDF 1.7 introduced a further subdivision into *text strings*, *PDFDocEncoded strings*, *ASCII strings* and *byte strings*.

2.3.4 Name objects

A name is a uniquely defined sequence of characters, preceded by a slash (/). Whitespace and certain delimiter characters are not allowed within names, but these limitations can be circumvented by representing such characters using their corresponding hexadecimal code.

2.3.5 Array objects

An array is a one-dimensional collection of objects arranged sequentially. An array may be made up of any combination of object types, including other arrays. Arrays are enclosed in square brackets ([…]). An example:

```
[true /MyName (Johan) 2.71]
```

2.3.6 Dictionary objects

A dictionary is a lookup table whose entries are defined as key / value pairs. A key is always a name object, whereas the value can be any type of object, including another dictionary. Dictionary objects are enclosed in double angle brackets (<< … >>). The following is an example of a simple dictionary:

```
<<    /Type /Example
      /Subtype /DictionaryExample
      /Version 0.01
      /IntegerItem 12
      /StringItem (a string)
      /Subdictionary    <<    /Item1 0.4
                              /Item2 true
                              /LastItem (OK)
                        >>
>>
```

2.3.7 Stream objects

A stream object is a sequence of bytes. Unlike string objects, streams can be of unlimited length. A stream object always starts with a dictionary that describes the sequence of bytes in the stream (such as size, filters, and any decode parameters), followed by the actual stream, which is wrapped between the keywords ‘stream’ and ‘endstream’. An example:

```
2 0 obj
<<
/Length 39
>>
stream
BT
/F1 12 Tf
72 712 Td (A short text stream.) Tj
ET
endstream
endobj
```

Note the ‘obj … endobj’ lines at the beginning and end of the object description. These label the stream object as an ‘indirect object’, which is explained in section 2.3.9. Stream objects are always defined as indirect objects.

2.3.8 Null object

This is a ‘neutral’ object, i.e. it does nothing. It is used, amongst other things, for dealing with nonexistent objects.

2.3.9 Indirect objects

Using indirect objects it is possible to assign a unique object identifier to any object. This object identifier, which is always an integer number, can then be referred to by other objects.

The object identifier is made up of a unique object number, followed by a generation number. For example:

```
5 0 obj
      (Some text)
endobj
```

Here, an indirect string object is defined with an object number of 5, generation number 0 and the value 'Some text'. The object can then be referred to from some other location in the file by using an indirect reference, which is defined by the object identifier followed by the keyword 'R':

```
5 0 R
```

An indirect reference to a nonexistent object returns the null object.

2.4 PDF Document structure

Section 2.2 discussed the *file structure* of a PDF document, and section 2.3 gave an overview of the objects that exist within a document. The *document structure* (or logical structure) describes how the objects are organised within the body element of a PDF file. Figure 2 gives an overview of the logical structure of a PDF file, which is a hierarchical tree. At the root of the tree is the *catalog* dictionary. The catalog has a number of entries, most of which are dictionaries (see Section 2.3.6) that contains child objects of their own. For instance, the *Pages* dictionary contains references to an array of *Page* dictionaries; in turn, the individual Page dictionaries refer to the content stream objects that make up the content of a page, annotations, and so on. In practice the structure of any actual PDF document will be more complex than the one shown in Figure 2, because the catalog may contain many more entries than the ones shown in the Figure.

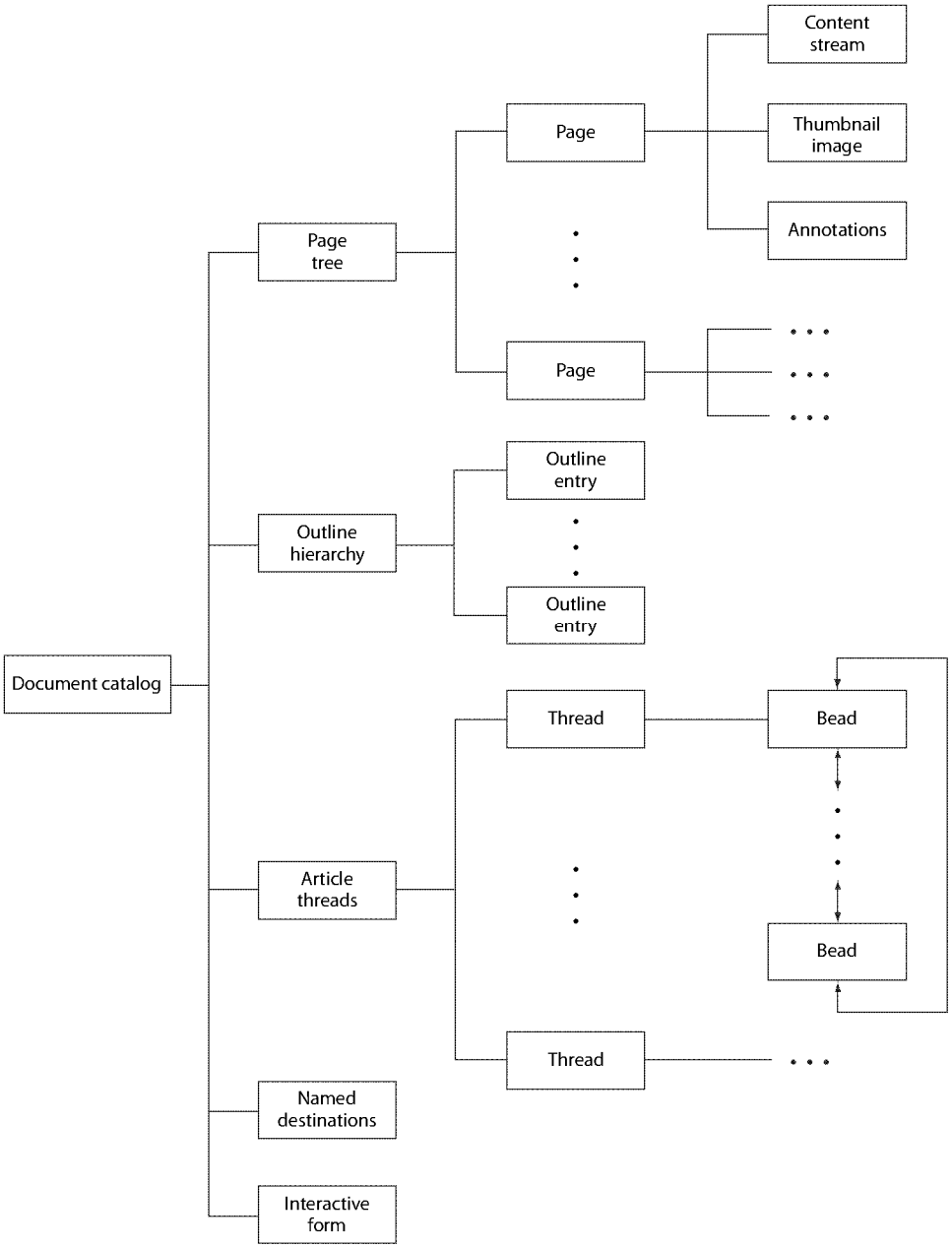

Figure 2 Structure of a PDF document

3 File identifiers and general information

3.1 Identifiers

Before going into the actual features of the Portable Document Format, it is helpful to first explain a number of identifiers that are often used to describe and identify specific file formats. Table 3.1 gives an overview. Some of these identifiers distinguish between different PDF versions, whereas others only describe PDF as a whole.

MIME is an abbreviation of *Multipurpose Internet Mail Extensions*. It was originally a standard to describe the content of e-mail messages. However, it is now commonly used as a descriptor of digital media types in general. A MIME descriptor is made up of a least a *type* part and a *subtype* part, which are separated by a forward slash [9]. For PDF, the MIME type is always 'application/pdf', without distinguishing between PDF versions.

PUID stands for *PRONOM Unique Identifier*. It is the identifier that is used in the online PRONOM file format registry [3]. Each PDF version has its own PRONOM identifier.

e-Depot ID is the identifier that is used to uniquely define each file type in the KB's e-Depot repository. As with the PRONOM identifier, each PDF version has its own e-Depot ID.

Many file types can also be recognised (at least to some extent) by their fixed *extension* (suffix), which is always *.pdf* for the Portable Document Format.

3.2 General information

In addition to these identifiers, Table 3.1 provides some additional information that is mainly useful within the context of the KB's e-Depot. The *release* date gives an indication from which year onwards documents of a particular version exist.

The specifications of the PDF format were laid out to ensure a great degree of *forward compatibility*. This means that old versions of the reader software are always able to display files that have a version number that is more recent. As an example, Adobe Acrobat Reader 1.0 will read PDF 1.7 files, even though PDF 1.7 obviously did not exist when Acrobat Reader 1.0 was released. The PDF specification states that reader (or viewer) applications should be written in such a way that they simply ignore any unknown features (such as new features that did not yet exist when the reader was written). This also implies that if a (new) document contains features that are not recognised by the (old) reader, these features may not display the way they should, or they may not even display at all. Also, aside from Adobe, a myriad of companies, organisations and individuals offer applications for viewing PDF documents. Because of the complexity and feature-richness of the PDF format, many of these third-party applications do not support the full set of features defined in the PDF specification. This may also result in documents not appearing the way they were originally intended.

Because of this, Table 3.1 lists the native Adobe reader that corresponds to each PDF version. Only the native reader guarantees full support of all the features that may be (potentially) present in a document.

Finally, Table 3.1 shows, for each PDF version, the corresponding number of publications in the e-Depot repository.

3.3 Identification of version number

The version number of any given PDF document is defined as a comment in the first line of the file header (see also section 2.2.1). As an example, for a PDF 1.4 document this will be:

```
%PDF-1.2
```

From PDF 1.4 onwards, version info can also be stored as a *version* entry in the file's so-called 'catalog dictionary'. This value can be different from the value in the file header. If this is the case, the value in the catalog should be used. The idea of having two different version identifiers within one file that can have conflicting values may seem strange. However, when an existing PDF file is modified, Acrobat adds the applied changes to the end of the document (instead of overwriting the existing document). One of the advantages of this approach is that changes that were applied to a document can be traced back and, if needed, undone. If, for example, a PDF 1.4 document is modified and subsequently saved in PDF 1.5 format, this will leave the original header (which reads '%PDF-1.4') unchanged, but a new catalog entry is added which can be used to identify the document as PDF 1.5.

Table 3.1 **File identifiers and general information related to different versions of Adobe's Portable Document Format**

	MIME	PUID	e-depot ID	Extension	Released	Native reader	# e-Depot *
PDF 1.0	application/pdf	fmt/14	52	.pdf	1993	Acrobat 1.0	
PDF 1.1	application/pdf	fmt/15	31	.pdf	1996	Acrobat 2.0	
PDF 1.2	application/pdf	fmt/16	1	.pdf	1996	Acrobat 3.0	
PDF 1.3	application/pdf	fmt/17	2	.pdf	1999	Acrobat 4.0	
PDF 1.4	application/pdf	fmt/18	48	.pdf	2001	Acrobat 5.0	
PDF 1.5	application/pdf	fmt/19	49	.pdf	2003	Acrobat 6.0	
PDF 1.6	application/pdf	fmt/20	50	.pdf	2004	Acrobat 7.0	
PDF 1.7	application/pdf	?	62	.pdf	2006	Acrobat 8.0	
PDF 1.7 (ISO 32000-1)	application/pdf	?	?	.pdf	2008	Acrobat 9.0	

* Date of count: DD/MM/2009. Note that these figures are based on metadata stored in the e-Depot. It is known that in several cases the actual document version does not correspond to the version indicated in the metadata, so these figures are only an indication.

4 Authentication

4.1 Background

For the long-term preservation of digital materials it is essential to have some mechanisms that ensure their authenticity. Authenticity includes *fixity*, the integrity of the content information, and *provenance*, or the history of the content information and its changes over time.

The PDF format offers a number of features that may be useful within this context. These are based on attaching a digital signature to a document. Like an ordinary signature, the purpose of a digital signature is to demonstrate that a document was created by the person who signed it, and that no changes were made to the document after it was signed. Digital signatures typically work in the following way. First, a *hash function* is applied to the raw byte data in a file, which produces a so-called *hash value* or *message digest*. This is a fixed-size sequence of bits. The important thing to remember here is that hash functions work in such a way that even very small changes in their input (here: the data in the file) immediately result in drastic changes in their output. As a next step, the message digest is encrypted using a *private key* that is known only to the owner or creator of the original document. The encrypted message digest serves as the signature, and is attached to the file[1]. A recipient (or user) of the file can then verify the integrity of the document using a *signature verifying algorithm*. Given the contents of the file, a public key and the signature, the signature verifying algorithm checks whether the message digest as it is computed from the received file matches the signature. If this is the case, this substantiates the file's authenticity.

A remaining problem is that by itself this approach does not warrant the identity of the owner of the public key. In other words: the signer of a document may pretend to be someone else. Users of digital signatures may therefore register their public key with a certificate authority, which is a third party that issues certificates that contain a user's public key and information on his or her identity [10].

As it happens, the PDF format supports different types of signatures, each of which has a different specific purpose. These signature types may co-exist in one file. This is not described in detail here; however, future versions of this document may add a more exhaustive description of authentication features in PDF.

4.2 History of authentication-related features

The following table gives an overview of authentication features in PDF and from which PDF version onwards they are supported:

[1] The message digest is actually computed over a range of bytes in a file, which is typically the whole file excluding the bytes that are occupied by the signature itself (otherwise, the addition of the signature would by itself immediately invalidate the signature!)

Feature	Support from version
Digital signatures, signature values calculated from message digest	PDF 1.3
Possibility to calculate signatures from *object digest**	PDF 1.5

* An *object digest* is a hash that is calculated by 'selectively walking a subtree of objects in memory'. Object digests can be used as an alternative to message digests.

4.3 Identification

The presence of a signature dictionary –a dictionary whose type is *Sig*– indicates that a document uses some type of digital signature.

5 Fonts

5.1 What is a font?

A *font* can be defined as a collection of *glyphs* for a particular character set. Here, a *glyph* denotes a specific graphical rendering of a character. Figure 3 below illustrates this by showing six different glyph renderings of the character 'a':

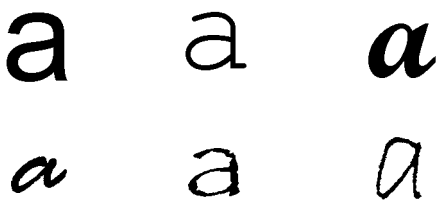

Figure 3 Six glyph renderings of the character 'a'

5.2 Font types

The PDF format allows the use of a number of different font types. The following font types are also known as *simple* fonts (this is to distinguish them from *composite* fonts, which are discussed further on in this section):

Type 1 fonts include some of the most commonly used fonts, the so-called 'standard 14' fonts[2]. *Multiple master* (*MMType1*) fonts are an extension of Type 1 fonts.

TrueType is a font format that was developed by Apple Computer, Inc, and which was later adopted as a standard font format for the Microsoft Windows operating system.

Technically, Type 1 and TrueType are quite similar. A noteworthy feature shared by both font types is that their definition includes so-called 'hinting' information, which improves the rendering quality of these fonts at small sizes and low resolutions [18].

OpenType (supported from PDF 1.6 onwards) is an extension of the TrueType format.

Type 3 fonts are quite different from the aforementioned font types, in that they use streams of PDF graphics instructions to describe glyphs. (All other font types refer to a separate font program for this.) Individual glyphs of a Type 3 font may contain graphic effects (e.g. graduated fills, variable stroke widths) that are not achievable with Type 1 or TrueType fonts. Because of this, Type 3 fonts are often used for logo designs [19]. Type 3 fonts do not support

[2] The 'standard' fonts are: *Times−Roman, Times−Bold, Times−Italic, Times−BoldItalic, Helvetica, Helvetica−Bold, Helvetica−Oblique, Helvetica−BoldOblique, Courier, Courier−Bold, Courier−Oblique, Courier−BoldOblique, Symbol, ZapfDingbats*

'hinting', which means that the rendering quality may deteriorate at small sizes and low resolutions.

Type 0 fonts (supported from PDF 1.2 onwards) are *composite* fonts. The glyphs of Type 0 fonts are obtained from a font-like object that is called a *CIDFont*. There are 2 types of CIDFonts: *Type 0* CIDFonts, whose glyph descriptions are based Type 1 fonts; and *Type 2* CIDFonts, whose glyph descriptions are based on TrueType fonts[3]. Type 0 fonts (and their associated CIDFont objects) are particularly useful when dealing with writing systems for languages with large character sets, such as Chinese, Japanese and Korean.

5.3 Embedding and subsetting of fonts

For a basic understanding of how fonts are represented in PDF, it is important to make a distinction between *font dictionaries* and *font programs*. A font (*Font*) dictionary simply contains general information about a font, such as the font's type, name and PostScript name. The actual glyphs of a font are defined in a *font program*. Both are connected in the following way: a font dictionary usually contains a font descriptor (*FontDescriptor*) entry. The font descriptor is a dictionary that contains entries on various font attributes. One of its (optional) attributes is a *FontFile* entry, which is a stream object that contains the embedded font program. The appearance of documents that use font programs that are not embedded will depend on the viewer application and the environment in which it runs. If a font that matches the PostScript name that is specified in the font dictionary is available on the user's system, this font will be used and the document will –in *most* cases, but read on- display normally. If not, the information in the font descriptor dictionary may be used by the viewer to either generate a substitute font, or select a font that is similar to the original one. However, the outcome of this process is rather unsure, and embedding fonts by default will prevent most of these problems. There is, however, one important issue that is not solved by embedding alone. Suppose you create a document that uses a font called 'MyFont', which is embedded. Now imagine that you send this file to a colleague, who, by some strange coincidence, has a font installed on his computer that is also called 'MyFont', but which is, apart from the name, an entirely *different* font altogether. As the default behaviour of Adobe's reader application is to use the system font if it can be found, your colleague will end up seeing your document with wrong font [20]! This situation can be prevented from happening by embedding only a *subset* of the font. With subsets, only the glyphs that are actually used in the document are embedded (rather than the entire character set). More importantly, embedded subsets are stored under a unique name that, although *derived* from the original font name, is actually *different* from it, so it will never match any system font name. This ensures that reader software will always use the actual embedded font.

The above discussion does not apply to Type 3 fonts, which do not use font programs at all. Instead, the Font dictionary of a Type 3 font has a separate entry where a dictionary can be defined that contains, for each character, the content stream that constructs and paints the corresponding glyph. These content streams are embedded within the document by default.

5.4 Legal aspects of embedded fonts

As font programs are subject to copyright, there may be restrictions to the use of font programs. Such restrictions may, for example, forbid fonts to be embedded at all. Alternatively, embedding may be allowed for the purpose of viewing and printing the

[3] A CIDFont is never used directly, but only indirectly as a component of a Type 0 font.

document, but the further use of the font for creating new or modified documents may be prohibited.

5.5 Implications for digital preservation

PDF documents that use fonts that are not embedded within the file present a serious preservation and accessibility risk. When such files are viewed on a (future) system on which these fonts are not available, they may not display correctly, or may be completely illegible. In order to avoid any possible ambiguity about the font's name, fonts should be subset as well as embedded. There may be a conflict between these long-term preservation requirements and copyright restrictions on certain fonts. Therefore, it is important to use only fonts that can be embedded legally.

5.6 Overview of risks

Risk
• Appearance of documents that contain non-embedded fonts may be different from the appearance as intended by the producer of the document. • Documents that contain non-embedded fonts may be completely illegible. • Documents may use fonts that cannot be legally embedded. • Possible naming conflict of fonts that are embedded without subsetting . • Rendering quality of Type 3 fonts may deteriorate at small sizes or low resolutions*
* This is not really a preservation risk, but as the KB aims to offer high-quality materials, it is nevertheless important to avoid the use of Type 3 fonts as much as possible.

5.7 History of supported fonts

The following table shows which font types are allowed for each PDF version:

Font type	Support from version
TrueType	PDF 1.0
Subset TrueType	PDF 1.0
Type 1	PDF 1.0
Subset Type 1	PDF 1.1
Type 3	PDF 1.0
Type 0	PDF 1.2
CIDFont Type 0	PDF 1.2
CIDFont Type 2	PDF 1.2
OpenType	PDF 1.6

5.8 Identification

The following table summarises how the main font features in a PDF file can be identified:

Feature	Identified by presence of
Font dictionary	object with type *Font* (font type defined by subtype, e.g. *TrueType* , *Type1*, etc)
Font descriptor dictionary	object with type *FontDescriptor* (which is an optional entry in a font dictionary)
Embedded font program	*FontFile*, *FontFile2* or *FontFile3* entry in font descriptor dictionary
Font subset (in font dictionary)	value of *BaseFont* entry that is made up of six uppercase letters, followed by a '+' sign and the original PostScript name *
Font subset (in font descriptor dictionary)	value of *FontName* entry that is made up of six uppercase letters, followed by a '+' sign and the original PostScript name *

* For example: *APYTEB+Helvetica*

6 Colours

6.1 Colour spaces

This section covers features that affect the way colours are displayed in a PDF file.

<Insert a brief discussion of colour spaces here. Mention problems that can occur, e.g. colours may appear differently on different devices (monitor, printer). Explain how ICC profiles and OutputIntents can help when dealing with different devices>

6.2 History of supported colour spaces

The following table shows which colour spaces are allowed from what PDF version onwards:

Colour space	Type	Support from version
DeviceGray	Device-dependent	PDF 1.0
DeviceRGB	Device-dependent	PDF 1.0
DeviceCMYK	Device-dependent	PDF 1.0
CalGray	Device-independent	PDF 1.1
CalRGB	Device-independent	PDF 1.1
Lab	Device-independent	PDF 1.1
ICCBased	Device-independent	PDF 1.3
Indexed	Special	PDF 1.2
Pattern	Special	PDF 1.2
Separation	Special	PDF 1.2
DeviceN	Special	PDF 1.3

6.3 Colour depth

Colour depth defines, for each colour channel, the number of levels than can be expressed. So, for an RGB image this would be the maximum number of discrete levels of red, green and blue. It is often expressed as a *bit depth*, i.e. as the number of bits that are used to represent each colour. The number of colour levels is related to bit depth in the following manner:

$$N_c = 2^b$$

where N_c is the number of colour levels and b is the bit depth. The table below shows the number of colour levels for some typical bit depth values:

b	N_c
1	2
2	4
4	16
8	256
16	65536
32	4294967296

The relevance of colour depth in a digital preservation context is that the specifications of older PDF versions only allow 1, 2, 4 or 8 bit images, whereas from PDF 1.5 onwards 16 bit images are also supported. Generally speaking, image quality improves with increasing colour depth (although other factors are important as well). When deciding on a file format for the long-term preservation of documents that contain high-quality artwork or photographic material, the ability to represent images in the best possible quality may be a crucial requirement. Consequently, the possibility to use large bit depths could be an important factor in the decision process.

6.4 History of supported colour depths

The following table shows which colour depth values are supported from which PDF version onwards:

Colour depth	Support from version
1, 2, 4, 8 bits	PDF 1.0
16 bits	PDF 1.5

6.5 Overview of risks

Risk
• Appearance of images that use device-dependent colour spaces may be different from the appearance as intended by the producer of the document • Use of low bit-depths may compromise appearance of high-quality artwork or photographic material

6.6 Identification

To be written…

7 Embedded data and file attachments

7.1 Background

The PDF format provides several mechanisms for embedding data from external files within a PDF document. Embedded objects are not part of the document's content stream, although the content stream may refer to them through *annotations* (e.g. *link*, *movie*, *sound* or *file attachment* annotations).

The main long-term accessibility risk associated with embedded objects is that they may rely on external applications. For instance, an embedded word processing file will require a word processing application that supports that specific file format. In addition, since *any* type of file can be embedded, it is also possible to embed viruses and other malicious code, which might, when activated,s even be a threat to the entire repository that holds a PDF document.

7.2 Overview of risks

Risk
• Display of embedded content may rely on external applications that may not be available in the future • Embedding of viruses, worms and other malicious code is possible

7.3 History of embedding-related features

The following table shows which embedding-related features are supported from which PDF version onwards:

Feature	Support from version
Embedded sound data	PDF 1.2
Embedded movies	PDF 1.3
Embedded file streams	PDF 1.3

7.4 Identification

The presence of embedded content can be uniquely identified by the presence of the *EF* entry in a file specification dictionary. Here is an example:

```
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
```

Here the upper object (31 0) is the file specification. The *EF* entry points to another object (32 0), which contains the embedded data. Noting that the stream object has the *EmbeddedFile* type, you might wonder whether this is not a better unique identifier for embedded content. The problem with this approach is that not all file streams that contain embedded content have the *EmbeddedFile* type, whereas the reference to these streams always contains an *EF* entry. Scanning for file attachment annotations (i.e. annotation objects with subtype *FileAttachment*) will not identify all references to embedded content either, since other annotation types may refer to embedded objects as well.

8 Encryption and password protection

8.1 Background

From PDF 1.1 onwards it is possible to use encryption to restrict access to a document's content. Encryption is done at the level of strings and streams. Viewing such an encrypted document requires a password. A password-protected PDF file can have up to two passwords: an *owner* password and a *user* password. The *owner* password always provides full, unrestricted access to the file. Besides this, the owner of a file can set or change the user password, and he or she can define special document permissions. The *user* password allows one to decrypt (and consequently display the contents of) a file. Depending on the permissions set by the owner, users may be barred from performing the following operations (which are all enabled or disabled separately):

- Printing of the document
- Changing the document (other than by adding or changing text notes)
- Copying of text and graphics from the document
- Adding and changing text notes

On a side note, from PDF 1.5 onwards, these very same permissions can also be set for unencrypted (non password-protected) documents in the *permissions dictionary*.

Encryption and password-protection pose a significant threat to the long-term accessibility of digital materials. First of all, passwords may get lost, or they may simply not be available to the end users of these materials. Second, even if it is possible to decrypt the contents of a file, the restrictions imposed by the document permissions can introduce problems when trying to migrate such a file to a newer version (or to convert it to another format). This last observation also applies to file permissions set in the permissions dictionary in PDF 1.5 and more recent[4].

[4] However, the PDF Manual contains the following note on document permissions: '*Despite the specification of document permissions in a PDF file, PDF cannot enforce the restrictions specified. It is up to the implementors of PDF viewers to respect the intent of the document creator by limiting access to an encrypted PDF file according to the permissions and passwords contained in the file*' [2]. This seems to suggest that the restrictions (printing, copying, changing) could be circumvented by deliberately not implementing them in the viewer. (In the case of an encrypted file, the password is still needed for decrypting the contents of the file in the first place.

8.2 Overview of risks

Risk
• Password-protected files may become inaccessible when password is lost • Password protection and file permission settings may prevent users from performing certain actions on a document • Password protection and file permission settings may hinder actions that are necessary for migration

8.3 History of encryption-related features

The following table shows which encryption-related features are supported from which PDF version onwards:

Feature	Support from version
Encryption of strings and streams using RC4 encryption algorithm with 40 bit key.	PDF 1.1
Support for 128-byte encryption keys	PDF 1.4
Encryption using alternative unpublished algorithm	PDF 1.4
Additional document permissions[5]	PDF 1.4
Encryption of embedded files in documents that are otherwise unencrypted	PDF 1.5
Encryption using AES (Advanced Encryption Standard) algorithm	PDF 1.6

8.4 Identification

Information that is related to encryption is stored in the encryption dictionary (*Encrypt* entry), which is located in the document's trailer dictionary. If the *Encrypt* entry is absent, this means that the document is not encrypted.

[5] A noteworthy document permission that was added in PDF 1.5 limits printing to a reduced-quality representation of a document.

9 Filters

9.1 Background

Section 2.3.7 already introduced *stream objects*, which are the main building blocks of the content stream of a PDF document. The data inside a stream object are often encoded in a specific way. A *filter* is an optional part of the stream object specification that tells the viewer application how the data inside the stream have to be decoded. There are several reasons why data inside a stream may be encoded.

First of all, the creator of a PDF file may want to avoid the presence of any non-printable characters inside the document to ensure maximum portability of the file across different environments. To achieve this, he can *encode* (transform) all binary data (such as images) into a representation that only contains printable *ASCII* characters. Examples of such encoding algorithms are ASCII hexadecimal encoding and ASCII base-85 encoding. A viewer application that supports the corresponding *decoding* algorithm will be able to perform the reverse transformation, which yields the original binary data as they were before encoding.

Compression (i.e. data volume reduction) is another reason for encoding data in a stream object. Compression is especially useful for image data, which typically occupy large amounts of storage space in uncompressed form. In practice, text data in a PDF file are usually represented in compressed form as well.

Irrespective of the aim of encoding stream data, the important thing is that the viewer application knows which stream objects contain encoded data, and what method was used for encoding. The very purpose of the *filter* specification is to provide this piece of information. In addition, in order to interpret an encoded object in a meaningful way, the viewer application needs to support the decoding algorithm that is defined in the *filter* specification.

The importance of filters in the context of long-term preservation is that PDF files that contain encoded data require the availability of a viewer application that is able to decode the data according to the filter specification. The PDF specification defines a set of standard filters that should be supported by all PDF viewer applications. New filter types are added to this set from time to time, which means that PDF files that use these new encodings may not display correctly on older viewers. Furthermore, in theory it is possible that filter types attain deprecated status, which means that support may be dropped in future versions of the file specification. To our knowledge this has not happened yet to any of the filters in the standard filters set.

A number of data compression algorithms are subject to patents or intellectual property constraints, which may endanger the future accessibility of files that use these encodings. This is sometimes used as an argument against the use of the *LZWDecode* filter [11]. The LZW compression algorithm used to be covered by several patents, which led to some controversy over legal aspects of its use during the mid-‘90s (see e.g. [12]). Even though these patents have expired by now, the use of the *LZWDecode* filter is not permitted in the PDF/A-1 specification [13]. Also, parts of compression algorithms may be subject to so-called ‘submarine patents’, which are patents that were “first published and granted long after the initial application was filed” [14]. Just as an example, there is currently no 100% guarantee that the *JPXDecode* filter (PDF 1.5), which is based on the JPEG2000 standard, is not affected by this (although this is controversial) [15].

Finally, some compression methods are *lossy* (meaning that some of the information contained in the original image data is lost in the compressed representation) and result in a reduction of image quality. This is not strictly a preservation issue, but as the KB aims for high quality graphics it is nevertheless important that lossy compression methods are used with caution (or not used at all).

9.2 Overview of risks

Risk
• Appearance of documents that use filter encodings that are not supported by a specific viewer application may be different from the appearance as intended by the producer of the document • Intellectual property constraints on certain decoding algorithms may endanger future accessibility of documents that use these encodings • Use of lossy encodings may compromise image quality

9.3 History of allowed filters

The following table lists the standard filters that are allowed within PDF documents:

Filter	Description	Support from version
ASCIIHexDecode	Decodes arbitrary binary data encoded in ASCII hexadecimal representation	PDF 1.0
ASCII85Decode	Decodes arbitrary binary data encoded in ASCII base-85 representation	PDF 1.0
LZWDecode	Decompresses text or binary data (e.g. monochrome images)	PDF 1.0
RunLengthDecode	Decompresses binary data (e.g. monochrome images)	PDF 1.0
CCITTFaxDecode	Decompresses binary data (e.g. monochrome images)	PDF 1.0
DCTDecode	Decompresses colour and grayscale image data using an algorithm that is based on the JPEG standard	PDF 1.0
FlateDecode	Decompresses text or binary data	PDF 1.2
JBIG2Decode	Decompresses monochrome image data encoded using the JBIG2 standard	PDF 1.4
JPXDecode	Decompresses data encoded using the wavelet-based JPEG2000 standard, reproducing the original image data	PDF 1.5
Crypt	Decrypts data encrypted by a security handler, reproducing the original data as they were prior to encryption	PDF 1.5

9.4 Identification

Filter types can be identified from the value of the *Filter* entry in the object stream dictionary. It is possible that multiple filters are defined for an object. For instance, the following example shows an object stream that was first compressed using the LZW algorithm, after which the compressed binary data were encoded in ASCII base-85 representation. Note that the *decoding* needs to proceed in the reverse order, which is why the *ASCII85Decode* filter appears first in the Filter dictionary:

```
1 0 obj
<< /Length 534
/Filter [ /ASCII85Decode /LZWDecode ]
>>
stream
J..)6T`?p&<!J9%_[umg"B7/Z7KNXbN'S+,*Q/&"OLT'F
LIDK#!n`$"<Atdi~\Vn%b%)&'cA*VnK\CJY(sF>c!Jnl@
RM]WM;jjH6Gnc75idkL5]+cPZKEBPWdR>FF(kj1_R%W_d
&/jS!;iuad7h?[L-F$+]]0A3Ck*$I0KZ?;<)CJtqi65Xb
Vc3\n5ua:Q/=0$W<#N3U;H,MQKqfg1?:lUpR;6oN[C2E4
ZNr8Udn.'p+?#X+1>0Kuk$bCDF/(3fL5]Oq)^kJZ!C2H1
'TO]Rl?Q:&'<5&iP!$Rq;BXRecDN[IJB`,)o8XJOSJ9sD
S]hQ;Rj@!ND)bD_q&C\g:inYC%)&u#:u,M6Bm%IY!Kb1+
":aAa'S`ViJglLb8<W9k6Yl\\0McJQkDeLWdPN?9A'jX*
al>iG1p&i;eVoK&juJHs9%;Xomop"5KatWRT"JQ#qYuL,
JD?M$0QP)lKn06l1apKDC@\qJ4B!!(5m+j.7F790m(Vj8
8l8Q:_CZ(Gm1%X\N1&u!FKHMB~>
endstream
endobj
```

10 Images

10.1 Background

The PDF specification includes several features that could be a threat to the long-term accessibility of images in documents. What these features have in common is that they may alter the look of images, depending on the context or as a result of user actions.

Alternate images –introduced in PDF 1.3- is a feature that allows one to use multiple versions of an image in one file. A typical use would be to define a reduced resolution version of an image that is displayed when viewing a file on-screen. The full-resolution version of the same image could then be reserved for printing.

PDF 1.5 introduced *optional content*, which allows one to selectively view or hide graphical objects. As an example, different layers of a map (roads, waterways, cities) could be defined as separate elements in an optional content dictionary, allowing the user of a document to view or hide particular layers.

PDF 1.4 introduced the *transparent imaging model*. This enables (graphical) objects to be painted with varying degrees of opacity. The implementation of the transparent imaging model is quite complex. Viewer applications that do not support transparency are generally able to display any document that contains transparent images. However, these images will be displayed as fully opaque, so the document may look very different from the way it was originally meant to.

Summarising, the potential effect of the above features is, that the visual appearance of a document as it is experienced by a user may be very different from the original appearance as it was intended by the producer of the document.

10.2 Overview of risks

Risk
• Use of alternate images may result in document appearance that is different from the appearance as intended by the producer of the document • Use of optional content may result in document appearance that is different from the appearance as intended by the producer of the document • Use of transparency may result in document appearance that is different from the appearance as intended by the producer of the document

10.3 History of image-related features

The following table summarises the implementation of the image-related features discussed in the previous section:

Feature	Support from version
Alternate images	PDF 1.3
Transparency (‘transparent image model’)	PDF 1.4
Optional content	PDF 1.5

10.4 Identification

The following table shows how the aforementioned features can be identified:

Feature	Identified by presence of
Alternate images	*Alternates* or *OC** entry in XObject
Optional content	*OCProperties* dictionary in document catalog
Transparency	type *Group* and subtype (S) *Transparency* in XObject (‘transparency group XObject’)

* The *OC* entry may be used to indicate alternate images from PDF 1.5 onwards

11 Interactivity

11.1 Background

From version 1.2 onwards, PDF documents may contain interactive *forms*. A form is a collection of fields in which users can enter data. The contents of the form fields can be submitted to a specified location using a uniform resource locator (URL). External data can also be imported into a form, and from PDF 1.3 onwards form fields can trigger *JavaScript actions* (see section 14).

The presence of form data within a document may give rise to a number of problems. Form submission and import actions are likely to contain external dependencies, which will break the functionality of forms in a document in the long term. Other form actions may change the visual appearance of forms.

11.2 Overview of risks

Risk
• Form submission / import may rely on external dependencies • Form actions may result in visual appearance of forms that is different from the appearance as originally intended by the producer of the document

11.3 History of interactive features

The following table summarises the support of interactive features in different PDF versions:

Feature	Support from version
Acrobat Forms (AKA 'AcroForms')	PDF 1.2
XFA forms (forms based on Adobe's XML Forms Architecture[6])	PDF 1.5

11.4 Identification

The presence of interactive forms can be identified by the presence of an interactive form (*AcroForm*) dictionary in the document catalog . If a document contains XFA forms, there will be an *XFA* entry inside the Acroform dictionary.

[6] This is not a replacement of Acrobat Forms; from PDF 1.5 onwards both AcroForms as well as XFA Forms are allowed.

12 Links and references to external data

12.1 Background

A PDF file may contain links and references to external data sources. This may lead to problems if the document that contains the references becomes separated from the external objects it refers to, or if these external objects become inaccessible. Also, even if the actual objects are accessible, viewing them may require external applications that may not be available at some future date (the very same issue was already discussed in section 7.1 for the case of embedded data). Therefore, from a long-term preservation point of view, PDF documents should be self-contained and free of external dependencies. References to external objects may come in a number of different shapes and sizes.

Link annotations (introduced in PDF 1.0) may point to data in another (external) file, or to a webpage. From PDF 1.1 onwards they can also contain *launch* actions that can be used to open (external) files or applications on the computer on which the document is viewed. A launch action can target *any* external file or application. This may cause problems when a PDF file is migrated to a different (e.g. future) environment, where the target file or application may not be available.

Objects that occupy large amounts of data (e.g. images) can be represented a *stream objects* in PDF. From PDF 1.2 onwards, stream objects can also refer to external files. Since both text and images are usually represented as stream objects, this means that virtually *any* part of the content stream can be an external reference.

The specification of the PDF 1.2 file format introduced *Movie annotations* and *Sound annotations*. In PDF 1.2, *Movie annotations* are links that refer to external movie files (from PDF 1.3 onwards, they can also refer to movies that are embedded within the file). Movie annotations have become largely obsolete with the introduction of *Screen annotations* in PDF 1.5. *Sound annotations* may either refer to sound objects that are embedded in the file itself, or to external sound files (see also section 13). Movies and sounds can also be referred to by an ordinary link annotation that contains a *Sound*, *Movie* or *Rendition* action.

The PDF 1.3 specification introduced *web capture content*. This allows (among other things), information from a given URL to be imported and added to an existing PDF file. This feature is fundamentally incompatible with long-term preservation, as it actively changes a file's representation upon viewing.

The PDF 1.4 specification introduced *Reference XObjects*, which allow one PDF document to import data from another PDF document (which may either be an external file or an embedded filestream).

12.2 Overview of risks

Risk
• External objects that are referred to may become separated from referring document, or may become inaccessible altogether. This will affect the appearance of such documents. • Display of external objects may rely on other external applications that may not be available in the future.

12.3 History of features that use external references

The following table shows the various features that use external references, and from which PDF version onwards they can be found:

Feature	Support from version
Link annotations	PDF 1.0
Launch actions (within link annotations)	PDF 1.1
Stream objects that refer to external files	PDF 1.2
Movie annotations (may also refer to embedded movies from PDF 1.3 onwards)	PDF 1.2
Sound annotations (may also refer to embedded sounds)	PDF 1.2
Web capture content	PDF 1.3
Reference XObjects (may also refer to embedded filestreams)	PDF 1.4
Screen annotations (may also refer to embedded movies)	PDF 1.5

12.4 Identification

The following table shows how the presence of any of the features discussed in the previous sections can be identified in a PDF document.

Feature	Identified by presence of
Link annotation with reference to another PDF document	*GoToR* ('Go-to remote') action
Link annotation with reference to file on local file system or network drive	*Launch* action.
Link annotation with reference to location on the internet	*URI* (uniform resource identifier) action
Movie annotation	Annotation object with subtype *Movie*
Sound annotation	Annotation object with subtype *Sound*
Screen annotation	Annotation object with subtype *Screen*
Link annotation, destination movie	*Movie* action or *Rendition* action *
Link annotation, destination sound	*Sound* action *
External file stream	*F* (file specification) item in stream dictionary
Reference XObject	*Ref* entry in XObject dictionary
Web capture content	*SpiderInfo* dictionary in document catalog

* In order to determine whether sound or movies are embedded or from an external file we would need to look at the underlying object(s))

13 Multimedia

13.1 Background

The PDF format supports a number of methods for including and viewing multimedia content such as sounds and movies. The main long-term preservation risks of multimedia within a PDF file are:

- The display of multimedia content may depend on external applications that may not be available in the future
- The multimedia content itself may be an external file, and the link between this file and the referring document may be lost over time
- The format in which multimedia content is stored may become obsolete in the future

The specification of the PDF 1.2 file format introduced support for playing multimedia content using *Movie annotations* and *Sound annotations*, both of which were already briefly mentioned in section 12.1. From PDF 1.5 onwards, *Screen annotations* are the preferred way of referring to multimedia content, and the use of Movie annotations is discouraged from this version onwards. PDF 1.6 introduced *3D annotations*, which are used for viewing three-dimensional models. In addition to these annotation types, *link annotations* (section 12.1) can also be used to refer to multimedia content.

13.2 Movie annotations

In PDF 1.2 *Movie annotations* are links that refer to external movie files; in later versions they can also refer to movies that are embedded within the file itself. The PDF Reference Manual leaves the precise file format for movies unspecified, other than stating that it should be "self-describing". In practice this means that virtually any format is allowed. In turn, this implies that the application that reads the PDF file (or a plug-in, or some external application that is used for this) should support playback for a virtually unlimited number of movie formats. Since movie formats may become obsolete over time, the accessibility of Movie annotations is likely to be problematic in the long run[7]. The introduction of *Screen annotations* in PDF 1.5 has made Movie annotations largely obsolete, and their use is discouraged from this version onwards.

13.3 Sound annotations

Sound annotations can either refer to sound objects that are embedded in the file itself, or link to external sound files. For embedded sounds, the PDF Reference Manual gives some guidelines on which sound formats to use for optimum portability. However, the support of compressed sound data, with apparently no limitations on which particular compression methods to be used, means that in practice *any* format is allowed. Similarly, the specification of external sound files goes no further than stating that sound files should be '"self-describing", containing all the information necessary to render the sound'.

[7] In fact, even the PDF Reference Manual explicitly states here that 'portability is not guaranteed'

The playback of both embedded and external sound data requires that the application that reads the PDF file (or a plug-in, or some external application that is used for this) should support playback of the supported formats. Since the supported file formats are described in such general terms, this means that in practice virtually *any* possible sound format should be supported, on any platform. Since sound formats may become obsolete over time, the accessibility of Sound annotations is likely to be problematic in the long run.

13.4 Screen annotations

From PDF 1.5 onwards, *Screen annotations* are the preferred method for referring to media content. Screen annotations allow more control over how media clips are displayed. The points that were made before on the long-term accessibility of content that is referred to by Movie annotations largely apply to Screen annotations as well (even though the PDF 1.5 Reference is somewhat more specific in its recommendations of which media formats should be supported by PDF readers [6]).

13.5 3D annotations

PDF 1.6 introduced the possibility to incorporate and view models of three-dimensional artwork. The actual data are stored as *3D streams*, which conform to the Universal 3D (U3D) format. The contents of a 3D stream are accessed through a *3D annotation*, which renders a view of the three-dimensional object on the page of a document. Users can rotate and move the objects, so that they can view them from different angles. In PDF 1.6, some features related to the viewing of 3D objects require the use of embedded JavaScript. The specification of PDF 1.7 removed some of these dependencies on JavaScript. Also, PDF 1.7 introduced many new features related to the appearance and behaviour of 3D artwork.

13.6 Link annotations

Link annotations were already discussed in section 12.1. Link annotations can refer to movies, sounds and 3-D artwork using specific *actions* that are part of the link annotation.

13.7 Overview of risks

Risk
• Display of multimedia content may depend on external applications that may not be available in the future • Multimedia content itself may be an external file, and the link between this file and the referring document may be lost over time • Format in which multimedia content is stored may become obsolete over time

13.8 History of multimedia features

The following table lists the various multimedia features in PDF and when they were introduced:

Feature	Support from version
Sound annotations	PDF 1.2
Movie annotations * (movie as external file)	PDF 1.2
Movie annotations * (movie as external or embedded file)	PDF 1.3
Screen annotations	PDF 1.5
3-D annotations	PDF 1.6
* From PDF 1.5 onwards, Screen annotations replace the functionality of Movie annotations. Movie annotations are deprecated in later versions, even though they are still supported.	

13.9 Identification

The following table shows how the presence of any of the features discussed in the previous sections can be identified in a PDF document.

Feature	Identified by presence of
Movie annotation	Annotation object with subtype *Movie*
Sound annotation	Annotation object with subtype *Sound*
Screen annotation	Annotation object with subtype *Screen*
3D annotation	Annotation object with subtype *3D*
Link annotation, destination movie	*Movie* action or *Rendition* action *
Link annotation, destination sound	*Sound* action
Link annotation, destination 3-D object	*GoTo3DView* action

14 Scripting

14.1 Background

The specification of the PDF 1.3 file format introduced *JavaScript actions*. JavaScript is a scripting (programming) language that, among other things, can be used to automate certain tasks, and for controlling and manipulating form fields (see section 11.1). From PDF 1.6 onwards, JavaScript is also used for displaying and manipulating three-dimensional artwork (section 13.5).

The use of JavaScript in PDF documents introduces a number following long-term preservation risks. First, JavaScript actions only work properly within a viewer application that has a built-in JavaScript interpreter (i.e. an application or plugin that "understands" JavaScript and compiles it to machine code). Although all current versions Adobe Acrobat Reader support JavaScript, documents that use JavaScript may not display correctly on viewers without JavaScript support. Second, JavaScript actions in form fields may change the visual appearance of forms (this problem was already mentioned in section 11.1). Third, JavaScript actions may introduce external dependencies that may be hard to identify. Finally, various IT security experts reported a number of security flaws in Adobe's reader applications that allow the execution of malicious code on a user's system [16], [17]. These flaws can be exploited in a number of different ways, some of them involving the use of JavaScript. This can actually pose a threat to the entire repository in which a document is stored.

14.2 Overview of risks

Risk
• Appearance of documents that use JavaScript in viewers that do not support JavaScript may be different from the appearance as intended by the producer of the document • JavaScript actions in form fields may change visual appearance of forms • JavaScript actions can introduce external dependencies that may result in the document appearing differently from the way it was intended by its producer • Presence of security leaks in Acrobat reader that can be exploited using JavaScript

14.3 History of scripting features

The following table lists the scripting features in PDF and when they were introduced:

Feature	Support from version
JavaScript actions	PDF 1.3

14.4 Identification

The use of JavaScript can be identified by the presence of *JavaScript* actions in a document.

15 Structure

15.1 Background

From PDF 1.3 onwards, it is possible to store information about a document's logical structure in so-called *structure elements*. This makes it possible to explicitly tag chapters, sections, subsections, figures, captions, and so on. A hierarchical *structure tree* defines how the structure elements are related to each other.

In PDF 1.4 this concept is extended to *tagged PDF*, which allows page content (text, tables, and graphics) to be extracted and reused for other purposes. Some possible examples are:

- Extraction of text and graphics for pasting into other applications
- Automatic reflow of text and graphics to fit a page of a different size than was assumed for the original layout
- Text processing (e.g. searching, indexing, spell-checking)
- Conversion to other file formats (e.g. HTML, XML) with preservation of document structure and basic styling information
- Making content accessible to the visually impaired

Tagged PDF achieves this by defining a set of standard *structure types* that describe the logical structure of a document in terms of paragraphs, headings, articles, and tables. Styling information is stored in the form of *structure attributes*. Also, tagged PDF makes an explicit distinction between actual page content and layout and pagination artefacts (e.g. page numbers, headers, footers).

The presence of structure information is relevant to the long-term preservation of a document in a number of ways. First, structure enhances the feasibility of any future migration of a document's contents to some other file format. It also improves the accessibility of a document. For example, structure enables re-flow of text and graphics on handheld devices, and a meaningful interpretation of a document by screen reader software.

15.2 Overview of risks

Risk
• Possibilities for migration will be limited for documents that do not use tagging • Limited accessibility to the visually impaired of documents that do not use tagging • Re-flow of text and graphics on handheld devices may not be possible for documents that do not use tagging

15.3 History of structure-related features

The following table lists the various structure-related features in PDF and when they were introduced:

Feature	Support from version
Structure elements	PDF 1.3
Tagged PDF	PDF 1.4
Separate tagging of elements whose page order cannot be determined (*TagSuspect*)	PDF 1.6

15.4 Identification

A document that conforms to the tagged PDF conventions can be identified from the following characteristics ('loose' definition, but see below):

1. the document catalog contains a mark information (*MarkInfo*) dictionary
2. the *MarkInfo* dictionary contains a *Marked* entry (this is a Boolean flag)
3. the value of *MarkInfo* is *true*

If the above conditions are all met, the document will most likely conform to tagged PDF conventions. However, there are situations in which some elements of a document may not completely conform to tagged PDF. From PDF 1.6 onwards, such elements can be given a special tag (*TagSuspect*), and their presence in the document is indicated by a special *Suspects* flag in the mark information dictionary. By default, the value of *Suspects* (if it exists at all) is *false*, and it is *true* if the document contains any *TagSuspect* elements. So, the more strict way of identifying tagged PDF would be ('strict' definition):

1. the document catalog contains a mark information (*MarkInfo*) dictionary
2. the *MarkInfo* dictionary contains a *Marked* entry (this is a Boolean flag)
3. the value of *MarkInfo* is *true*
4. if a *Suspects* flag exists, its value is *false*

16 Known issues

This chapter is reserved for the description and documentation of specific problems, issues, case studies and any next-level weirdness encountered with PDF documents ‘in the wild’ (or, more likely, the KB’s somewhat less wild e-Depot repository) .

Acknowledgements

Judith Rog originally started research on the PDF format at the KB, and her earlier work provided a good starting point for the current document. I also thank Judith for her help in getting me started with PDF, and for providing me with much of the literature upon which this report is based. Thanks are also due to Barbara Sierman for her helpful comments on earlier drafts of this document.

References

[1] Bienz, T., Cohn, R. & Meehan, J.R., 1996. Portable Document Format Reference Manual, Version 1.2. Adobe Systems Incorporated.

[2] Adobe Systems Incorporated, 2000. PDF Reference, second edition - Portable Document Format Version 1.3. Addison-Wesley.

[3] PRONOM file format registry. Link: http://www.nationalarchives.gov.uk/PRONOM/ (accessed 9 September 2009)

[4] RC4. Wikipedia, the free encyclopedia. Link: http://en.wikipedia.org/wiki/RC4 (accessed 15 September 2009)

[5] Adobe Systems Incorporated, 2001. PDF Reference, third edition - Portable Document Format Version 1.4. Addison-Wesley.

[6] Adobe Systems Incorporated, 2003. PDF Reference, fourth edition – Adobe Portable Document Format Version 1.5.

[7] Adobe Systems Incorporated, 2004. PDF Reference, fifth edition – Adobe Portable Document Format Version 1.6.

[8] Adobe Systems Incorporated, 2006. PDF Reference, sixth edition – Adobe Portable Document Format Version 1.7.

[9] Internet media type. Wikipedia, the free encyclopedia. Link: http://en.wikipedia.org/wiki/Internet_media_type (accessed 15 September 2009)

[10] Digital signature. Wikipedia, the free encyclopedia. Link: http://en.wikipedia.org/wiki/Digital_signature (accessed 15 September 2009)

[11] Chou, C., 2006. Guidelines for Creating Archival Quality PDF Files. Florida Digital Archive.

[12] Battilana, M.C., 2004. The GIF Controversy: A Software Developer's Perspective. Link: http://lzw.info/ (accessed 24 September 2009)

[13] ISO, 2005. Document management – Electronic document file format for long-term preservation – Part 1: Use of PDF 1.4 (PDF/A-1).

[14] Submarine patent. Wikipedia, the free encyclopedia. Link: http://en.wikipedia.org/wiki/Submarine_patent (accessed 6 October 2009)

[15] Joint Photographic Experts Group. JPEG 2000 Committee Drafts. Link: http://www.jpeg.org/jpeg2000/CDs15444.html (accessed 6 October 2009)

[16] Two New Vulnerabilities in Adobe Acrobat Reader. Link: http://www.f-secure.com/weblog/archives/00001671.html (accessed 6 October 2009)

[17] Adobe Reader/Acrobat JavaScript Method Handling Vulnerability. Link: http://www.f-secure.com/vulnerabilities/SA30832 (accessed 6 October 2009)

[18] Phinney, T.W. TrueType & PostScript Type 1: What's the Difference? Link: http://www.truetype-typography.com/articles/ttvst1.htm (accessed 8 October 2009)

[19] Type 3 fonts. Link: http://www.prepressure.com/fonts/basics/type3 (accessed 8 October 2009)

[20] Sprague, R. No 1. Font Issues. PDF Planet. Link: http://www.planetpdf.com/mainpage.asp?WebPageID=362 (accessed 12 October 2009)

[21] Rog, J., 2007. PDF Guidelines - Recommendations for the creation of PDF files for long-term preservation and access. Koninklijke Bibliotheek / National Library of the Netherlands.