

Amyuni White Paper

**Missing PDF Fonts: Why it Happens
and What You Can Do About it**

Introduction

This document is the first of two that will look at some of the challenges faced by developers and non-developers who work with PDF technologies and who are curious about what causes fonts in a PDF to render incorrectly or even go missing. Specifically, these documents provide an overview of some of the problems associated with missing font information in PDFs.

The first document presents the Portable Document Format as well as industry terms and concepts related to that format. The problem of missing font information will also be introduced. The second document expands on those terms and concepts and explores some of the common scenarios in which PDFs are either missing partial or entire font information.

Brief Overview of PDF

The Portable Document Format was originally conceived in 1991 as the Camelot Project, by Adobe's co-founder Dr. John Warnock. Inspired by the device independence of PostScript, Dr. Warnock wanted to develop a technology that could accurately display and print electronic documents across different operating systems, hardware, or applications. His answer was the PDF.

Unlike its predecessor (i.e., PostScript), PDF was first and foremost a file format and not a programming language. Although PDF evolved from PostScript, the primary difference is that PostScript is a true page description language and PDF is not. PDF does not contain programming constructs such as looping, control-flow constructs, or variables. Rather, PDF was envisioned to go further than PostScript by being able to describe how pages behave and what type of information a document could contain. Years later, the PDF would encompass complex features and functionalities such as search capabilities, audio, and even video.

On July 1, 2008, PDF became an open standard published by ISO as ISO 32000-1: 2008.

PDF Structure

PDFs are essentially collections of data objects organized in a hierarchical manner that describe how one or more pages in a document must be displayed. These data objects can describe a page, a resource, other objects, a sequence of operating instructions, and so on. Furthermore, a data object can reference other objects and be referenced by other objects (i.e., an object can be a parent object and a child object at the same time).

PDF documents contain four main types of objects that define its structure:

- the document catalog object
- page objects
- page content objects
- document and page resources

The document object typically contains a cross reference table and page objects. It can also contain elements such as document information, named destinations, thumbnails, and bookmarks.

Page Objects

Page objects can contain one or more content objects as well as several other types of elements such as page cropping information, hyperlinks, article threads, file annotations, form fields, digital signatures, and child pages in the document. Page objects also contain references to all the resources used by a page.

Content Objects

Content objects contain marking operators (i.e., drawings) and use resources such as fonts, images or colorspaces that are needed to fully render the page.

Resource Objects

PDF defines a number of resource objects such as fonts, images, color spaces, patterns, etc. Fonts are needed to render text, color spaces represent colors used in the document, patterns define how backgrounds are painted, etc.

PDF Organization

PDFs are sectioned into four separate areas:

- the header
- the body
- the cross-reference table (xref)
- the trailer

The Header

The header contains a comment that identifies the nature of a PDF document and the specifications to which it adheres. For example, the comment outlined in Figure 1 indicates that the document conforms to Version 1.7 of the PDF specification.

Figure 1. Header

```
%PDF-1.7  
%ÿÿÿÿ
```

The Body

The body of a PDF is where the content objects in the document are located. These objects include text streams, image data, fonts, annotations, and so on (see Figure 2 on page 4). The body can also contain numerous types of invisible (non-display) objects that help implement the document's interactivity, security features, or logical structure. Each object has three essential components: a numerical identifier, a fixed position (also known as an offset), and its content.

Figure 2. Examples of Objects in the Body

```

7 0 obj
<<
/Type /Font
/Subtype /TrueType
/BaseFont /CGFGAX+TRReservedPIFont,BoldItalic/FirstChar 32
/LastChar 35
/Widths [
220 265 187 567]
/FontDescriptor 8 0 R
>>
endobj
8 0 obj
<<
/Type /FontDescriptor
/FontName /CGFGAX+TRReservedPIFont,BoldItalic/Flags 4
/FontBBox [ -140 -269 1027 906 ]
/Ascent 704
/Descent 269
/FontFile2 9 0 R
...
>>
Endobj

```

The Cross-Reference Table

The cross-reference table (see Figure 3 on page 5) lists the locations of all the objects in a PDF document. The cross-reference table is divided into sections where each section begins with the starting and ending identifiers of the objects in that section. With the cross-reference table, a PDF parser can randomly identify object offsets and quickly access object locations throughout the document without having to read the entire file.

Figure 3. XREF Table

```

Xref
0 9
0000000000 65535 f
0000000017 00000 n
0000000067 00000 n
0000001244 00000 n
0000001264 00000 n
0000001370 00000 n
0000002027 00000 n
0000009301 00000 n
0000009321 00000 n
0000009424 00000 n

```

The Trailer

Even though the trailer is technically the end of a PDF document, it is the first entry point that applications use to access the essential components of a PDF. The trailer contains pointers that parsers and applications use to locate the cross-reference table and other important objects in a PDF.

Examples of important objects include the root object (that identifies the beginning of a page tree) and info objects (that contain vital metadata).

Figure 4. Trailer

```

Trailer
<<
/Size 101
/Root 4 0 R
/Info 99 0 R
/ID [ <50947B130F0E7443397A><50947B130F0E057443397A> ]
>>
startxref
1052783
%%EOF

```

Terms and Concepts

Before outlining the challenge of missing fonts in PDFs, it is important to review some of the underlying concepts and technologies that will be used throughout the rest of the documents.

Glyphs and Characters

Norman Walsh defines a glyph as: “the actual shape (bit pattern, outline) of a character image. For example, an italic “a” and a Roman “a” are two different glyphs representing the same underlying character. In this strict sense, any two images which differ in shape constitute different glyphs.” Consequently, glyphs are organized into different types of fonts. By contrast a character is an abstract symbol that is given shape through a glyph’s design.

Character Codes

A character code is a digit associated to a specific character. For example, a character with the character code “37” displays a different glyph depending on its typeface (e.g., Calibri, Arial, Webdings, etc). At the most basic level, an application that renders PDF documents only needs to access the character codes, the font information, and the mapping from the character code to the font information. With this information the rendering application extracts the key graphical data to draw a glyph on an output device such as a screen or printer.

Fonts and Typefaces

Although the difference between fonts and typefaces may seem trivial to some, confusion still lingers within some development circles, where the term font families is commonly used when referring to typefaces. This is why it is important to clarify some of the upcoming terms.

A font is a comprehensive group of characters with a specific style of type. It includes the letter and number set, special characters, as well as diacritical marks (accents). Furthermore, a font specifies the member of a type family such as roman, boldface, or italic type.

Within the context of PDF software development, a font is a PDF object commonly referred to as a font object (see Figure 5 on page 7), font dictionary or font data file. A font object contains a set of glyphs, characters, or symbols (such as wingdings). The font object also identifies the font program and contains additional information such as its properties.

Figure 5. Example of a Font Object

```

7 0 obj
<<
/Type /Font
/Subtype /TrueType
/BaseFont /CGFGAX+TRReservedPIFont,BoldItalic/FirstChar 32
/LastChar 255
/Widths [
711 416 440 440 440 440 258 258 258 258 494 550 493 493 493 493 493 220 489
543 543 543 543 220 542 438 ]
/FontDescriptor 8 0 R
>>
endobj

8 0 obj
<<
/Type /FontDescriptor
/FontName /CGFGAX+TRReservedPIFont,BoldItalic/Flags 4
/FontBBox [ -140 -269 1027 906 ]
/AvgWidth 382
/MissingWidth 0
/StemV 0
/CapHeight 0
/ItalicAngle 0
/Ascent 704
/Descent 269
/FontFile2 9 0 R
>>
endobj

9 0 obj
<<
/Filter /FlateDecode
/Length 10 0 R
/Length1 118472
>>

stream...
xœl½ €_Uö?~fv³%Ûd³>Mi•__jhò<H __"*"̀£_*"̀_@¹½"³)&"qXd__5BjXLQf_İP_â
"Ž_²12ĐÊİ(ûî•_~î„_$_!'ß[ ____æââ_\"Óä[_âò^ ă`4ĈâBAĀp_4-ō
endstream
endobj

```

By contrast, a typeface specifies a consistent visual appearance or style which can be a "family" or related set of fonts. Arial, Tahoma, or Helvetica are examples of typefaces. A typeface can contain a series of fonts. For example, a typeface such as Helvetica may include roman, bold, and italic fonts.

Font Technologies: Laying the Foundation

From their inception in the mid-1980s, font technologies have helped jump start the desktop publishing revolution and have enabled the written word to cross over to digital typesetting mediums.

Standards expanded, new font technologies emerged, and within a few years, the world of PDF had become more complex. Not only did those who developed PDF viewers and converters had to adapt to the emerging trends within the PDF industry, but they also had to support the rising demands for different languages.

Asian languages presented PDF developers with new challenges as the existing font technologies could no longer sufficiently answer increasing font complexities. These new challenges helped push font technologies and developers forward.

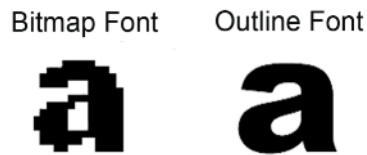
Outline Fonts

Although digital fonts are generally grouped into three format types (namely, bitmap, stroke, and outline (vector) types), this paper will focus on outline fonts.

Unlike bitmap fonts that are collections of raster images of glyphs, outline fonts (also known as scalable fonts) are collections of vector images. This means that outline fonts describe glyphs using points that are interpreted as lines and curves.

The advantage to using vector images is that they can be scaled to varying sizes without losing too many details. By contrast, bitmap fonts lose their detailed edges and often appear jagged or choppy when resized (Figure 6 on page 9).

Figure 6. Font Type Scalability Differences



Hinting: When Scalability Isn't Enough

Even though outline fonts are scalable, there are many instances in which proper rasterization can be compromised. For example, different applications, output devices, or printers can affect rasterization. To address this problem hinting technologies were developed. Hinting is additional mathematical information added to a font to ensure it retains its visual integrity when rasterized under various conditions.

Type 1 (PostScript)

Developed by Adobe Systems, PostScript fonts were developed to answer the demands of emerging laser printing technologies at the time. Using a subset of the PostScript language, Type 1 fonts contain an organized collection of procedures to describe glyph forms.

In addition, glyph outlines were interpreted by Type 1 fonts using a field of mathematical analysis known as (cubic) Bézier curves. When first introduced, Type 1 fonts were the first to include proprietary hinting technology to improve their display capabilities.

Type 1 fonts store information in two files. One file contains the character outlines (referred to as printer fonts) and the other contains the character information to display on screen.

Type 3

Type 3 fonts are essentially the same to Type 1 fonts except that they don't include hinting technology. While Type 1 fonts only use a subset of the PostScript language, Type 3 fonts encompass most of the PostScript language. This makes Type 3 fonts capable of displaying more elaborate designs and ligatures than Type 1 fonts. However, the added weight of the PostScript language into Type 3 fonts also makes their file sizes larger. They therefore take up more memory. Because Type 3 fonts use bit-mapped technology instead of hinting, they often produce poorer display results when they are scaled.

TrueType

Developed by Apple Computers, TrueType fonts are similar to Type 1 fonts, but include some important differences. Like Type 1 fonts, TrueType also uses Bézier curves to describe glyph information; however, TrueType employs quadratic mathematics rather than cubic.

Another difference between TrueType and Type 1 is that TrueType contains both the screen and printer font data in a single file. In addition, hinting information is stored inside the font file. This additional information makes TrueType fonts larger than their original PostScript rivals.

Unlike Type 1 files, however, which are composed of a subset of the PostScript language, TrueType font files are composed of structured tables. Each table contains the necessary information that applications or PDF viewers need to use and display a font. Tables also contain information to ensure that glyphs are displayed correctly when there are different types of internal encodings used in a document.

OpenType

OpenType fonts bring together some of Type 1 and TrueType technologies into one cross-platform format. OpenType's character encoding is based on Unicode and as a result, can support up to 65,536 glyphs, OpenType offers more development flexibility especially when working with Asian character sets and more sophisticated Roman glyphs that may use non-lining numerals, small caps, fractions, ligatures, and swashes. Like TrueType, an OpenType font contains all of its outline, metric, and bitmap information in a single file.

Font File Structures

In addition to their technological differences, fonts can also be categorized according to how they are structured as PDF objects. Generally, fonts can be structured as:

- Simple Fonts
- Composite Fonts

PDFs contain font objects (see Figure 5 on page 7) that essentially act as wrappers for embedded font programs that contain the actual font data. Font programs can be TrueType, OpenType, Type 1, and so forth. Font objects also contain a number of properties and descriptions of the font data in order to enable PDF applications and viewers to use the font in the document.

Simple Fonts

Simple Fonts use a single byte of information to represent a glyph. As a result, a maximum of 256 (2⁸) different glyph representations are possible. The Simple Font category includes the original instances of Type 1 and TrueType fonts.

Composite Fonts

Because of their 256 character encoding limitation, Simple Fonts could not support complex Asian glyphs, where a typical Japanese font can have over 7,000 Kanji, Katakana, and Hiragana characters, or non-horizontal writing.

The solution was the development of Composite Fonts (or CID fonts). Unlike Simple Fonts, Composite Fonts are multi-byte and can thus contain an arbitrary number of glyphs. As a result, Composite Fonts are able to support a wider range of glyphs.

Composite Font technologies enable developers to use any number of base fonts and create new composite fonts. Composite font technologies also enable developers to include two sets of character spacing details (metrics) in fonts. One metric can be used for horizontal writing mode and another for vertical writing mode.

Aside from their ability to handle complex glyphs, Composite Fonts are also flexible and expandable.

CMap File

A CMap is an ASCII text file that contains the PostScript language instructions required to map character codes to CID codes used by Composite Fonts. For example, after a character code is processed (from a keyboard input), the CMap file maps the character code to a corresponding Character Identifier number (CID). The CID code is then passed on to the Composite Font which will in turn generate the appropriate glyph. As we shall see in the next document, CMap files can also be missing and impact proper PDF processing.

Font Embedding

To display, print, or process a PDF accurately, it must contain the necessary font information. If font information is missing, recipients may not be able to display or edit the document properly or, worse, applications may not be able to process the PDF at all.

Embedding fonts in a PDF ensures that they display and print exactly from one system to another as the author intended. The following sections will look at how fonts are embedded in PDFs and introduce the upcoming subject matter for the following document.

Full Font Embedding

The first method of embedding fonts is full font embedding. Full font embedding effectively makes the font part of the PDF thereby preventing font substitution when recipients need to display or print a PDF. Essentially recipients don't need the same fonts to view or edit the document. This method is advisable in situations in which modifications to the PDF are expected.

Full font embedding can also potentially help avoid some of the problems associated with missing system fonts and ensure optimal viewing regardless of the system and platform. In an ideal PDF world, fully embedding all fonts would reduce many development woes.

The main drawbacks to full font embedding are file size and licensing issues. Every embedded font makes the document larger, especially if it contains Chinese, Japanese, or Korean (CJK) fonts, which can be problematic. In fact, CJK fonts are rarely fully embedded due to their large character sets. Also, fully embedded fonts can be extracted and used outside of the PDF file. As a result, this font extraction can create the potential of unlimited font distribution and violate the licensing policy of the font manufacturer. The solution then is to partially embed fonts in a document.

Partial Font Embedding (Subsetting Fonts)

Unlike full font embedding, subsetting a font only embeds the glyph definitions for the characters used (i.e., that are displayed in the PDF).

There are three main reasons one should subset fonts. First, as previously stated, PDFs are primarily for content exchange and viewing. PDF is not an ideal editing format, despite the popularity of PDF editing programs available on the Internet, and it is generally assumed

(rightfully or wrongfully) by the PDF's creator that the recipient will not modify the document's contents. As we shall see in the following document, editing a PDF is not always a straightforward affair.

Second, subsetting fonts reduces document size. For example, the size of the font "Arial Unicode MS" is nearly 20MB; however, subsetting this font to show 10 Kanji characters would instead only add approximately 25KB to the PDF. In cases where CJK fonts are used, full embedding all fonts would result in problematically very large files.

Third, subsetting of fonts avoids licensing issues because the font then becomes unusable for other purposes than rendering the document which is often permitted by the font licensors.

The draw back with partially embedded fonts is that if recipients do not have the fonts on their system, they will not be able to edit the document or will be very limited in their ability to edit text. This is where the problem of missing fonts begins to emerge.

When Fonts Go Missing

Now that some of the key PDF and font concepts have been reviewed, the different problems that can occur when font information is missing can be addressed.

The following document (Part 2) will explore how problems associated with missing font information can start right at the source, with the creation of the PDF document itself. These problems include full and partial font embedding, incomplete font information in TrueType fonts, and missing CMap files.

References:

Walsh, Norman. "Frequently Asked Questions About Fonts."

14 August 1996. <<http://nwalsh.com/comp.fonts/FAQ/index.html>>

Learn more about Amyuni PDF Development Tools at www.amyuni.com

© Amyuni Technologies Inc. All rights reserved. All trademarks are property of their respective owners.