How to set fallback fonts for PDF rendering and creation

Written by Apitron Documentation Team

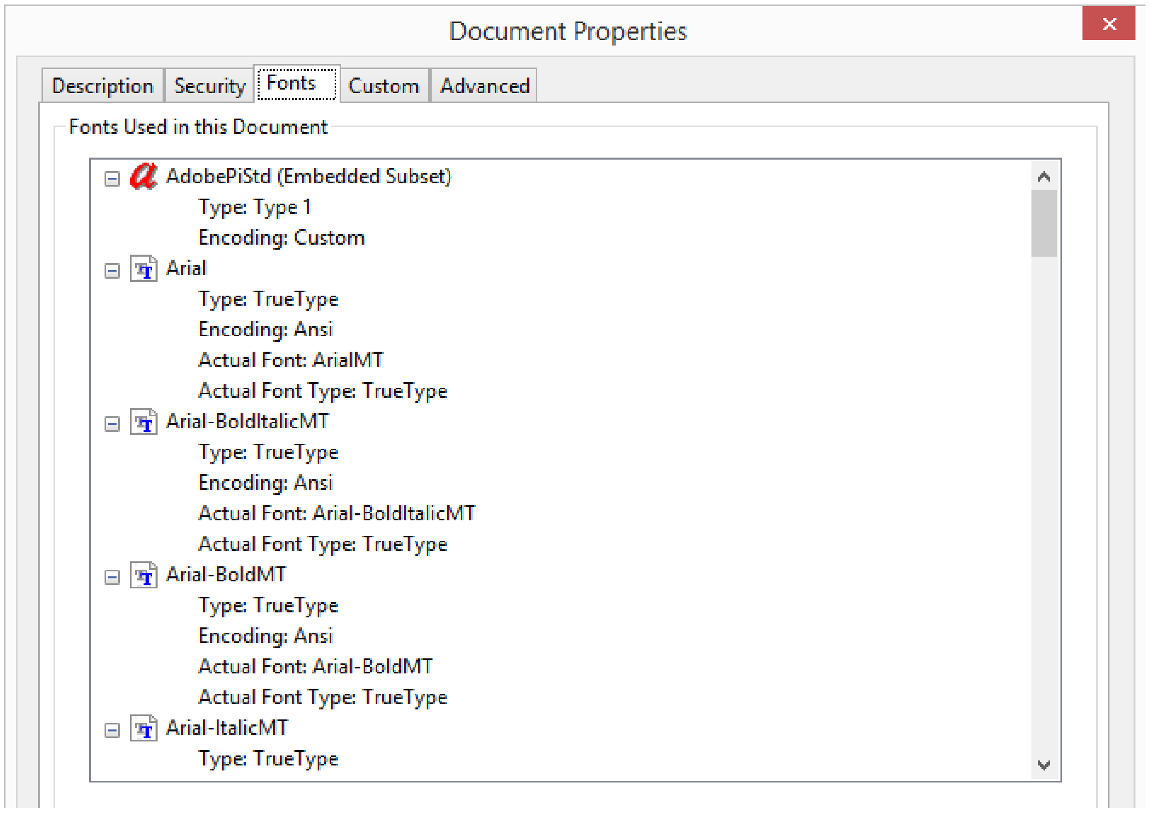
Introduction

PDF specification allows authors and software writers to create completely self-contained documents which include all necessary information for their rendering. It provides font data embedding – an important mechanism that can be used to get the documents consistently presented on many platforms and devices. While it looks good and natural from the first sight, it makes documents bigger because you have to embed the font's data used in document.

One of the alternatives to embedding is using standard Adobe 14 fonts (Times-Roman, Helvetica, Courier, Symbol, Times-Bold, Helvetica-Bold, Courier-Bold, ZapfDingbats, Times-Italic, Helvetica-Oblique, Courier-Oblique, Times-BoldItalic, Helvetica-BoldOblique, Courier-BoldOblique) which a conforming reader should understand and display correctly.

Other alternative (and most widely used) is to use external fonts, which are usually provided along with the operating system. These fonts can be referred to by their names, and PDF reader tries to find them inside the well-known system folders when it loads the document.

To examine the fonts used inside the particular PDF document, go to the File-> Properties...-> Fonts (tab) in Adobe PDF Reader. See the sample below:



AdobePiStd (Embedded Subset)
Type: Type 1
Encoding: Custom
Arial
Type: TrueType
Encoding: Ansi
Actual Font: ArialMT
Actual Font Type: TrueType
Arial-BoldItalicMT
Type: TrueType
Encoding: Ansi
Actual Font: Arial-BoldItalicMT
Actual Font Type: TrueType
Arial-BoldMT
Type: TrueType
Encoding: Ansi
Actual Font: Arial-BoldMT
Actual Font Type: TrueType
Arial-ItalicMT
Type: TrueType

Pic. 1 Examining fonts used in PDF document

You may notice that *AdobePiStd* is marked as embedded subset, while *Arial-based* fonts are being loaded externally from default system folder (*C:\Windows\Fonts* if you're running the reader on Windows). Furthermore, you may also note that these *Arial-based* fonts get replaced by the fonts shown as *Actual* fonts. Thus *Arial* becomes substituted by *ArialMT* and so on.

External fonts and font substitution may work well under some conditions, but what if you were to create a PDF reading or processing app using Apitron PDF Rasterizer or Apitron PDF Kit .NET components for a platform where system fonts are completely inaccessible e.g. Silverlight? Or you would like to provide a custom font fallback mechanism for cases where you have a very limited system fonts set, e.g. an Android phone app.

To help you with this, we have introduced new API for specifying font fallbacks. See the code section below for a real-world example.

Code sample

The following piece of code demonstrates how to set up font fallback in any of our products.

```
// let's set Arial as fallback for all not found fonts
using (Stream fontStream = File.Open("Arial.ttf", FileMode.Open))
{
    // register the fallback font, it can be either TTF or TTC stream,
    // font name and other attributes will be inferred automatically
    EngineSettings.RegisterUserFont(fontStream);
    // map the font to all not found fonts using *, it's possible to map particular fonts
    // using their names
    EngineSettings.UserFontMappings.Add(new KeyValuePair<string, string[]>("Arial", new[] {"*"}));
}
```

So, in order to font fallback to work you have to register the fonts and map them using `EngineSettings` class static methods.

Conclusion

Apitron PDF Kit and Apitron PDF Rasterizer products provide a great base for any PDF processing application. Understanding their settings doesn't require much time, but can help you to find the solution for the trickiest cases. Contact us if you need any help or have questions and we'll be happy to help.