

# JPedal

## PDF font handling White paper



**IDR Solutions Limited**

# PDF Font Handling

## Version History

1.0 Mark Stephens/Julie Stewart 18th August 2004

## Scope

This document aims to give the background on how fonts work in PDF files, how Java handles fonts and how JPedal handles fonts.

## Fonts in PDF files

PDF files use Type 1, Type 3 and TrueType fonts. These can be included in a file in TWO ways:-

1. Include just the font name. The PDF file will just include the name of the font (ie Arial) and the PDF viewer will have to locate this font or use an alternative.
2. Include both the name and a font stream - a binary file including detailed instructions on how to draw each font. This is called embedding. It makes the file larger but a suitable program can draw the fonts exactly, regardless of whether it has been installed on the machine. To reduce the size of the files, it is possible to include only the details for the glyphs required (only 1 glyph might be used from a font). This is called sub setting.

Type 3 fonts are not as common (offering inferior results to Type 1 and TrueType fonts) and are almost always embedded, so will not be discussed further.

## How to see what fonts are used in a PDF

Adobe Acrobat and Reader have a menu option under File to show font properties. The SimpleViewer sample includes an option, File > fonts, to show what fonts are present and whether they are embedded.

## Java Font handling

Java provides some general font functions which meet some of the requirements for PDF. It provides a set of Standard font families with several weights. Additionally, it provides some support for using the font streams embedded within PDF to generate TrueType (Java 1.3 and above) and Type 1 fonts (Java 1.5). This has several limitations - for example it cannot handle font streams created by FOP (a popular java PDF creation tool). To get around the limitations, JPedal includes its own TrueType and Type1 font renderer.

Fonts available within Java are set at runtime using the `font.properties` file (which is not particularly straight-forward or well-documented) and these are normally not the same as those installed on the system. So while Arial may be available on Windows, it may not be visible in Java.

## Overview of What JPedal offers

JPedal will display embedded fonts automatically and will try to match any fonts against those available to Java. For many situations this will be all that is required.

However, JPedal also provides some very sophisticated font substitution capabilities, allowing it to be tailored to all situations. It makes use of both the standard Java font families and JPedal's own font renderers.

Specifically JPedal offers the following capabilities for fonts which are not embedded:-

### **General**

1. Set any Font as the default to use if a match is not found. If not set, Lucida Sans will be used.

### **Using Java's internal fonts**

2. Map the name used in the PDF to any Java font family (especially useful where the names of common fonts are altered slightly for copyright reasons).

### **Using JPedal's own font renderer**

3. Provide Java with a set of font streams as files in the jar and ask Java to use them for displaying fonts. The fonts are resolved using font name, so `arialMt.ttf` will be used for `arialmt`.

4. Provide a list of directories which may contain TrueType fonts and ask JPedal to use any fonts found. Giving JPedal the windows font directory location (i.e. `C:/windows/fonts/`) therefore makes all Windows TrueType fonts available to use for display.

5. Provide a set of aliases so that fonts can be mapped to the names used in PDF files. JPedal maps fonts using the names (case is not important to `arial` is the same as `Arial`). For example, the names used by Microsoft, Sun or Apple for fonts tend to be slightly different compared to the ones used by Adobe. This works with both fonts in the jar file and in directories.

## Documented code examples

Each of these functions is shown as documented example code in the `SimpleViewer.java` example code. Here is a recent copy (please see the file for reference as this is always the latest version).

```
/**
 * FONT EXAMPLE CODE showing JPedal's functionality to set values for
 * non-embedded * fonts.
 *
 * This allows sophisticated substitution of non-embedded fonts.
 *
 * Most font mapping is done as the fonts are read, so these calls must be
 * made BEFORE the openFile() call.
 */
/**
 * FONT EXAMPLE - Replace global default for non-embedded fonts.
 *
 * You can replace Lucida as the standard font used for all non-embedded
 * and substituted * fonts by using this code.
 * Java fonts are case sensitive, but JPedal resolves this, so you could
 * use Webdings, webdings or webDings for Java font Webdings
 */
    try{
        //choice of example font to stand-out (useful in checking results
        //to ensure no font missed.
        //In general use Helvetica or similar is recommended
        decode_pdf.setDefaultDisplayFont("webdings");
    }catch(PdfFontException e){
        //if its not available catch error and show valid list

        System.out.println(e.getMessage());

        //get list of fonts you can use
        String[] fontList =GraphicsEnvironment.getLocalGraphicsEnvironment()
            .getAvailableFontFamilyNames();
        System.out.println("Fonts available are:-");
        System.out.println("=====\n");
        int count = fontList.length;
        for (int i = 0; i < count; i++) {
            System.out.println(fontList[i]);
        }

    }

}/***/

/**
 * IMPORTANT note on fonts for EXAMPLES
 *
 * USEFUL TIP : The SimpleViewer displays a list of fonts used on the current
 * PDF page with the File > Fonts menu option.
 *
 * PDF allows the use of weights for fonts so Arial,Bold is a weight of Arial.
 * This value is not case sensitive so JPedal would regard arial,bold and
 * aRial,BoLd as the same.
 */
```

```
* Java supports a set of Font families internally (which may have weights),
* while JPedal's substitution facility uses physical True Type fonts so
* it is resolving each font weight separately. So mapping works differently,
* depending on which is being used.
*
* If you are using a font, which is named as arial,bold you can use either
* arial,bold or arial (and JPedal will then try to select the bold weight
* if a Java font is used).
*
* So for a font such as Arial,Bold JPedal will test for an external TrueType
* font substitution (ie arialMT.ttf) mapped to Arial,Bold. BUT if the
* substitute font is a Java font an additional test will be made for a match
* against Arial if there is no match on Arial,Bold.
*
* If you want to map all Arial to equivalents to a Java font such as Times
* New Roman, just map Arial to Times New Roman (only works for inbuilt java
* fonts). Note if you map Arial,Bold to a Java font such as Times New Roman,
* you will get Times New Roman in a bold weight, if available. You cannot
* set a weight for the Java font.
*
* If you wish to substitute Arial but not Arial,Bold you should explicitly
* map Arial,Bold to Arial,Bold as well.
*
* The reason for the difference is that when using Java's inbuilt fonts
* JPedal can resolve the Font Family and will try to work out the weight
* internally. When substituting TrueType fonts, these only contain ONE
* weight so JPedal is resolving the Font and any weight as a separate font.
* Different weights will require separate files.
*/

/**
 * FONT EXAMPLE - Use fonts placed in jar for substitution
 *
 * This allows users to store fonts in the jar and use these for substitution.
 * Please see javadocs for a full description of usage.
 */
    //decode_pdf.addSubstituteFonts(fontPath,enforceMapping)

/**
 * FONT EXAMPLE - Use fonts located on machine for substitution
 *
 * This code explains how to use JPedal to substitute fonts which are not
 * embedded using fonts held in any font directory.
 *
 * It works as follows:-
 *
 * If the -DTTfontDirs="C:/win/fonts/","/mnt/X11/fonts" is set to a
 * comma-separated list of directories, any TrueType fonts (with .ttf file
 * ending) will be logged and added to the substitution table. So arialMT.ttf
 * will be added as arialmt. If arialmt is used in the PDF but not embedded,
 * JPedal will use this font file to render it.
 *
```

```
*
* If the name is not an exact match (ie you have arialMT which you wish
* to use to display arial, you can use the method
* setSubstitutedFontAliases(String[] name, String[] aliases) to convert
* it internally - see sample code at bottom of note.
*
* The Name is not case-sensitive.
*
* Spaces are important so TimesNewRoman and Times New Roman are regarded
* as 2 fonts.
*
* If you have 2 copies of arialMT.ttf in the scanned directories, the last
* one will be used.
*
* If the file was called arialMT,bold.ttf it is resolved as ArialMT,bold
* only.
*/
    String[] aliases={"helvetica","arial","arial,bold"};
    decode_pdf.setSubstitutedFontAliases("arialMT",aliases);

/**
* FONT EXAMPLE - Use Standard Java fonts for substitution
*
* This code tells JPedal to substitute fonts which are not embedded.
*
* The Name is not case-sensitive.
*
* Spaces are important so TimesNewRoman and Times New Roman are
* regarded as 2 fonts.
*
* If you have 2 copies of arialMT.ttf in the scanned directories, the
* last one will be used.
*
* If you wish to use one of Java's fonts for display (for example, Times
* New Roman is a close match for myCompanyFont in the PDF, you can use the
* code below
*
*     String[] aliases={"Times New Roman"};//,"helvetica","arial"};
*     decode_pdf.setSubstitutedFontAliases("myCompanyFont",aliases);
*
* Here it is used to map Java's Times New Roman (and all weights) to
* TimesNewRoman.
*/
    String[] aliases1={"TimesNewRoman"};//,"helvetica","arial"};
    decode_pdf.setSubstitutedFontAliases("Times New Roman",aliases1);
/**/
```