

# Generating Charts in PDF Format with JFreeChart and iText

Written by David Gilbert

May 28, 2002

© 2002, Simba Management Limited. All rights reserved.

*Everyone is permitted to copy and distribute verbatim copies of this document,  
but changing it is not allowed.*

**Please note that if you choose to use this document  
you do so entirely at your own risk.**

## 1 Introduction

This document describes how to generate charts in Adobe Acrobat's *portable document format* (PDF) using JFreeChart and iText, two popular free class libraries for Java.

The author (David Gilbert) is the Project Leader for JFreeChart.

## 2 Background

### 2.1 Acrobat PDF

Acrobat PDF is a widely used electronic document format. Its popularity is due, at least in part, to its ability to reproduce high quality output on a variety of different platforms.

PDF was created by Adobe Systems Incorporated. Adobe provide a free (but closed source) application called *Acrobat Reader* for reading PDF documents. Acrobat Reader is available on most end-user computing platforms, including GNU/Linux, Windows, Unix, Macintosh and others.

If your system doesn't have Acrobat Reader installed, you can download a copy from:

<http://www.adobe.com/products/acrobat/readstep.html>

On some platforms, there are free software packages available for viewing PDF files. Ghostview on Linux is one example.

### 2.2 iText

iText is a free Java class library for creating documents in PDF format. It is developed by Bruno Lowagie, Paulo Soares and others.

The home page for iText is:

<http://www.lowagie.com/iText>

At the time of writing, the latest version of iText is 0.92.

### 2.3 JFreeChart

JFreeChart is a free Java class library for generating charts. It is developed by David Gilbert and others.

The home page for JFreeChart is:

<http://www.object-refinery.com/jfreechart/index.html>

At the time of writing, the latest version of JFreeChart is 0.8.1.

## 3 Generating Charts in PDF Format

### 3.1 Introduction

In this section, I present a small application that creates a chart using JFreeChart and saves it to a PDF file using iText. The resulting file can be viewed using Acrobat Reader, or any other software that is capable of reading and displaying PDF files.

First, though, I will briefly review Java's `Graphics2D` class, since it provides the important link between JFreeChart and iText.

### 3.2 Graphics2D

The `java.awt.Graphics2D` class, part of the standard *Java 2D API*, defines a range of methods for drawing text and graphics in a two dimensional space. Particular subclasses of `Graphics2D` handle all the details of mapping the output (text and graphics) to specific devices.

JFreeChart has been designed to draw charts using only the methods defined by the `Graphics2D` class. This means that JFreeChart can generate output to any target that can provide a `Graphics2D` subclass.

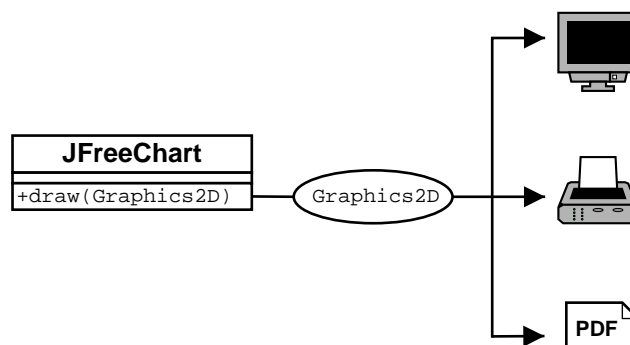


Figure 1: The JFreeChart `draw(...)` method

Recently, a new `PdfGraphics2D` class has been added to iText. This means that iText is now capable of generating PDF content based on calls to the methods defined by the `Graphics2D` class. This makes it easy to produce charts in PDF format, as you will see in the following sections.

### 3.3 Getting Started

To compile and run the sample application, you will need the following jar files:

File:	Description:
<code>jfreechart-0.8.1.jar</code>	The JFreeChart class library.
<code>jcommon-0.6.1.jar</code>	The JCommon class library (used by JFreeChart).
<code>itext-0.92.jar</code>	The iText class library.

The first two files are included with JFreeChart, and the third is included with iText.

### 3.4 The Application

The first thing the sample application needs to do is create a chart. By making use of some of the sample data in the JFreeChart download, I will create a chart in just two lines of code:

```
// create a chart...
XYDataset data = DemoDatasetFactory.createSampleXYDataset();
JFreeChart chart = ChartFactory.createXYChart("PDF Chart 1", "X", "Y", data, true);
```

There is nothing special here—in fact you could replace these two lines of code with any other code that creates a `JFreeChart` object. You are encouraged to experiment.

Next, I will save a copy of the chart in a PDF file:

```
// write the chart to a PDF file...
File fileName = new File("/home/dgilbert/jfreechart1.pdf");
saveChartAsPDF(fileName, chart, 400, 300);
```

There are a couple of things to note here.

First, I have hard-coded the filename used for the PDF file. I've done this to keep the sample code short. You will need to replace the file name with something appropriate for your system. In a real application, you would provide some other means for the user to specify the filename, perhaps by presenting a file chooser dialog.

Second, the `saveChartAsPDF(...)` method hasn't been implemented yet! To create that method, I'll first write another more general method, `writeChartAsPDF(...)`. This method performs most of the work that will be required by the `saveChartAsPDF(...)` method, but it writes data to an *output stream* rather than a file.

Inside `writeChartAsPDF(...)`, you will see some code that sets up and opens an iText document, obtains a `Graphics2D` instance from the document, draws the chart using the `Graphics2D` object, and closes the document.

You will also notice that one of the parameters for this method is a `FontMapper` object. The `FontMapper` interface maps Java `Font` objects to the `BaseFont` objects used by iText.

The `DefaultFontMapper` class is predefined with default mappings for the Java *logical fonts*. If you use only these fonts, then it is enough to create a `DefaultFontMapper` using the default constructor. If you want to use other fonts (for example, a font that supports a particular character set) then you need to do more work. I'll give an example of this later.

Here is the `writeChartAsPDF(...)` method:

```
public static void writeChartAsPDF(OutputStream out,
                                   JFreeChart chart,
                                   int width, int height,
                                   FontMapper mapper) throws IOException {
    Rectangle pagesize = new Rectangle(width, height);
    Document document = new Document(pagesize, 50, 50, 50, 50);
```

```

try {
    PdfWriter writer = PdfWriter.getInstance(document, out);

    document.addAuthor("JFreeChart");
    document.addSubject("Demonstration");
    document.open();

    PdfContentByte cb = writer.getDirectContent();
    PdfTemplate tp = cb.createTemplate(width, height);
    Graphics2D g2 = tp.createGraphics(width, height, mapper);

    Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
    chart.draw(g2, r2D, null);
    g2.dispose();
    cb.addTemplate(tp, 0, 0);
}
catch(DocumentException de) {
    System.err.println(de.getMessage());
}

document.close();
}

```

In the implementation of this method, I've chosen to create a PDF document with a custom page size (matching the requested size of the chart). You can easily adapt the code to use a different page size, alter the size and position of the chart and even draw multiple charts inside one PDF document.

Now that I have a method to send PDF data to an output stream, it is straightforward to implement the `saveChartAsPDF(...)` method. Simply create a `FileOutputStream` and pass it on to the `writeChartAsPDF(...)` method:

```

public static void saveChartAsPDF(File file,
    JFreeChart chart,
    int width, int height,
    FontMapper mapper) throws IOException {

    OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
    writeChartAsPDF(out, chart, width, height, mapper);
    out.close();
}

```

This is all the code that is required. The pieces can be assembled into the following program (reproduced in full here so that you can see all the required import statements and the context in which the code is run):

```

package com.jrefinery.chart.demo;

import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.io.File;
import java.io.OutputStream;
import java.io.BufferedOutputStream;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import com.lowagie.text.Document;
import com.lowagie.text.Rectangle;
import com.lowagie.text.DocumentException;
import com.lowagie.text.pdf.PdfWriter;
import com.lowagie.text.pdf.PdfContentByte;
import com.lowagie.text.pdf.PdfTemplate;
import com.lowagie.text.pdf.FontMapper;
import com.lowagie.text.pdf.DefaultFontMapper;
import com.lowagie.text.pdf.BaseFont;
import com.jrefinery.data.XYDataset;

```

```

import com.jrefinery.chart.JFreeChart;
import com.jrefinery.chart.ChartFactory;
import com.jrefinery.chart.demo.DemoDatasetFactory;

/**
 * A simple demonstration showing how to write a chart to PDF format using
 * JFreeChart and iText.
 * <P>
 * You can download iText from http://www.lowagie.com/iText.
 */
public class ChartToPDFDemo1 {

    /**
     * Saves a chart to a PDF file.
     *
     * @param file The file.
     * @param chart The chart.
     * @param width The chart width.
     * @param height The chart height.
     */
    public static void saveChartAsPDF(File file,
                                     JFreeChart chart,
                                     int width, int height,
                                     FontMapper mapper) throws IOException {

        OutputStream out = new BufferedOutputStream(new FileOutputStream(file));
        writeChartAsPDF(out, chart, width, height, mapper);
        out.close();

    }

    /**
     * Writes a chart to an output stream in PDF format.
     *
     * @param out The output stream.
     * @param chart The chart.
     * @param width The chart width.
     * @param height The chart height.
     */
    public static void writeChartAsPDF(OutputStream out,
                                       JFreeChart chart,
                                       int width, int height,
                                       FontMapper mapper) throws IOException {

        Rectangle pagesize = new Rectangle(width, height);
        Document document = new Document(pagesize, 50, 50, 50, 50);

        try {
            PdfWriter writer = PdfWriter.getInstance(document, out);

            document.addAuthor("JFreeChart");
            document.addSubject("Demonstration");
            document.open();

            PdfContentByte cb = writer.getDirectContent();
            PdfTemplate tp = cb.createTemplate(width, height);
            Graphics2D g2 = tp.createGraphics(width, height, mapper);

            Rectangle2D r2D = new Rectangle2D.Double(0, 0, width, height);
            chart.draw(g2, r2D, null);
            g2.dispose();
            cb.addTemplate(tp, 0, 0);
        }
        catch(DocumentException de) {
            System.err.println(de.getMessage());
        }

        document.close();

    }

    /**
     * Starting point for the demonstration application.
     */
}

```

```

public static void main(String[] args) {

    try {
        // create a chart...
        XYDataset data = DemoDatasetFactory.createSampleXYDataset();
        JFreeChart chart = ChartFactory.createXYChart("PDF Test Chart 1",
            "X", "Y",
            data, true);

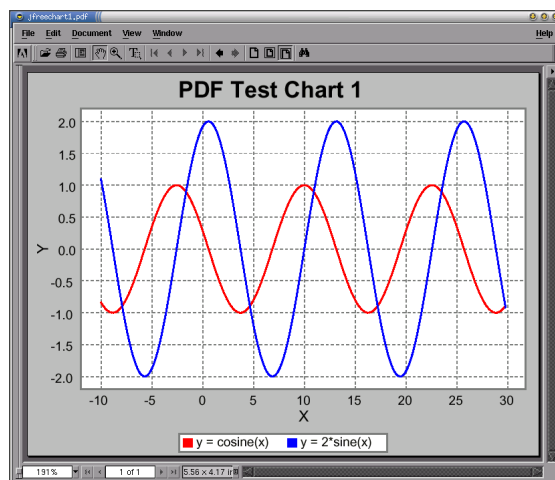
        // write the chart to a PDF file...
        File fileName = new File("/home/dgilbert/jfreechart1.pdf");
        saveChartAsPDF(fileName, chart, 400, 300, new DefaultFontMapper());
    }
    catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
}

```

Before you compile and run the application, remember to change the file name used for the PDF file to something appropriate for your system! And include the jar files listed in section 3.3 on your classpath.

### 3.5 Viewing the PDF File

After compiling and running the sample application, you can view the resulting PDF file using Acrobat Reader:



## 4 Unicode Characters

### 4.1 Introduction

It is possible to use the full range of Unicode characters in JFreeChart and iText, as long as you are careful about which fonts you use. In this section, I present some modifications to the previous example to show how to do this.

## 4.2 Background

Internally, Java uses the Unicode character encoding to represent text strings. This encoding uses sixteen bits per character, which means there are potentially 65,536 different characters available (the Unicode standard defines something like 38,000 characters).

You can use any of these characters in both JFreeChart and iText, subject to one proviso: *the font you use to display the text must define the characters used or you will not be able to see them.*

Many fonts are not designed to display the entire Unicode character set. The following website contains useful information about fonts that do support Unicode (at least to some extent):

`http://www.ccss.de/slovo/unifonts.htm`

I have tried out the **Arial Unicode MS** font with success—in fact, I will use this font in the example that follows. But you should bear in mind that supporting the full Unicode character set means that the font definition file is quite large: the `arialuni.ttf` file weighs in at 24,131,012 bytes on my system.

## 4.3 Fonts, iText and Java

iText has to handle fonts according to the PDF specification. This deals with document portability by allowing fonts to be (optionally) embedded in a PDF file. This requires access to the font definition file.

Java, on the other hand, abstracts away some of the details of particular font formats with the use of the `Font` class.

To support the `Graphics2D` implementation in iText, it is necessary to map `Font` objects from Java to `BaseFont` objects in iText. This is the role of the `FontMapper` interface.

If you create a new `DefaultFontMapper` instance using the default constructor, it will already contain sensible mappings for the logical fonts defined by the Java specification. But if you want to use additional fonts—and you must if you want to use a wide range of Unicode characters—then you need to add extra mappings to the `DefaultFontMapper` object.

## 4.4 Mapping Additional Fonts

I've decided to use the **Arial Unicode MS** font to display a chart title that incorporates some Unicode characters. The font definition file (`arialuni.ttf`) is located, on my system, in the directory:

`/opt/jbuilder5/jdk1.3/jre/lib/fonts`

Here's the code used to create the `FontMapper` for use by iText—I've based this on an example written by Paulo Soares:

```
DefaultFontMapper mapper = new DefaultFontMapper();
mapper.insertDirectory("/opt/jbuilder5/jdk1.3/jre/lib/fonts");
DefaultFontMapper.BaseFontParameters pp =
    mapper.getBaseFontParameters("Arial Unicode MS");
```

```

if (pp!=null) {
    pp.encoding = BaseFont.IDENTITY_H;
}

```

Now I can modify the code that creates the chart, in order to add a custom title to the chart (I've changed the data and chart type also):

```

// create a chart...
XYDataset data = DemoDatasetFactory.createTimeSeriesCollection2();
JFreeChart chart = ChartFactory.createTimeSeriesChart("PDF Test",
    "Time", "Price",
    data, true);

String text = "\u278A\u20A0\u20A1\u20A2\u20A3\u20A4\u20A5\u20A6\u20A7\u20A8\u20A9";
Font font = new Font("Arial Unicode MS", Font.PLAIN, 12);
TextTitle subtitle = new TextTitle(text, font);
chart.addTitle(subtitle);

```

Notice that the subtitle (which mostly consists of a meaningless collection of currency symbols) is defined using escape sequences to specify each Unicode character. This avoids any problems with encoding conversions when I save the Java source file.

The output from the modified sample program is shown here:

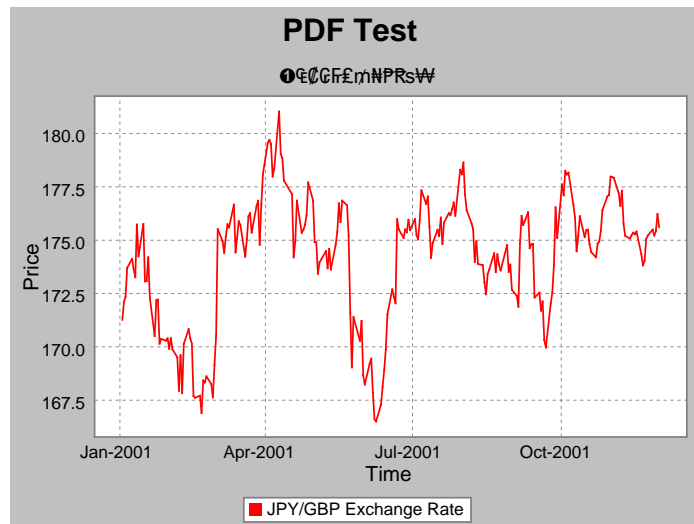


Figure 2: A Unicode subtitle

Note that the chart in Figure 2 has been embedded in this document in PDF format, so it is a good example of the type of output you can expect by following the instructions in this document.

## 5 Comments

If you have any comments or questions relating to this document, please send an e-mail to:

david.gilbert@object-refinery.com

Your feedback is appreciated.