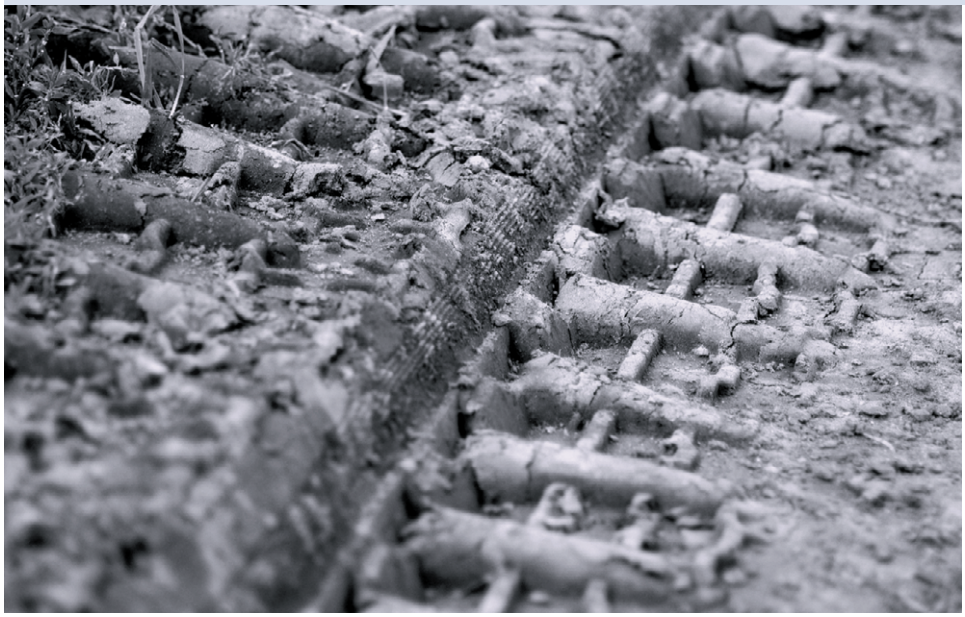
Java-Print-API

Sauberer Abdruck

Javas Druckfunktionen änderten sich im Lauf der Jahre mehrfach, nicht immer zum Besten. Der neue Java Print Service ist endlich so ausgereift, dass man ernsthaft mit ihm arbeiten kann. Mit einem Zusatzpaket findet er unter Linux auch Cups-Drucker. Bernhard Bablok



Richtig toll konnte Java bisher nicht drucken. Viele verschiedene, auch plattformabhängige Komponenten mussten zusammenspielen, was nicht immer die besten Ergebnisse brachte. Treue Leser des Coffee-Shops wissen, dass solche Probleme und ihre Lösung schon einmal Thema waren [1], damals mit dem JDK 1.2. Manches ist heute noch aktuell, aber die Druckschnittstelle hat sich wesentlich verändert.

Version 1.3 enthielt neue Klassen, um Job- und Seitenattribute zu steuern, JDK 1.4 ein komplettes API, das weitere Aspekte des Druckens umfasst. Im Folgenden geht es um Details zur Architektur des Java Print Service (JPS).

Die drei Teile des API

Das Java-Print-Service-API besteht aus drei Teilen, verteilt auf vier Packages. Die wichtigsten Klassen und Methoden finden sich im Package »javax.print«. Mit den Klassen und Interfaces dieses Package kann man Drucker (vom JPS Print Service genannt) finden, das Datenformat des Druckinhalts beschreiben, Druckjobs erstellen und an einen verfügbaren Drucker schicken.

Der zweite Teil umfasst die beiden Packages »javax.print.attribute« und »javax.print.attribute.standard«. Die Attribute sind wichtig, weil damit alle Aspekte des Druckens steuerbar sind. Attribute beschreiben statische Eigenschaften der Druckdienste (Unterstützt der Drucker beidseitiges Drucken?) und Dokumente (Welches Format haben die Seiten?) und transiente Eigenschaften, etwa die Anzahl der Kopien oder den aktuellen Status eines Druckauftrags.

Der letzte Teil des API im Package »javax.print.event« kümmert sich zum Beispiel um die Verfügbarkeit von Druckern und den Status von Druckaufträgen. Das Event-Modell folgt dem bekannten AWT-Modell mit Listenern, die für die entsprechenden Events zu registrieren sind. Das ist nichts Neues, weshalb dieser Teil des JPS an dieser Stelle nicht weiter behandelt wird.

Wie so häufig bei Java gibt es noch weitere Packages und Klassen aus früheren Versionen, die sich zwar nicht in das aktuelle API einfügen, aber trotzdem noch nutzbar sind. Beim Drucken sind dies die Klassen im Package »java.awt.print«. Schon der Name des Package zeigt den eingeengten Blick der Schöpfer des alten API, obwohl schon die damalige Version als generelles Printing-API gedacht war. Auch wenn die Klassen dieses Package noch nicht deprecated (missbilligt) sind, sollten sie nicht mehr zum Einsatz kommen – die neuen Klassen sind viel konsistenter und mächtiger.

Vom Dokument zum Druck

Vor dem Druck steht die Druckaufbereitung, doch dieser Teil des Druckvorgangs ist nicht Teil des JPS. Der Printservice geht also davon aus, dass das Dokument schon in einem für den Drucker (genauer: für das ganze Drucksystem) verständlichen Format vorliegt. Unter Linux mit seinem leistungsfähigen Cups ist diese Annahme für viele Standardformate erfüllt.

Dann läuft das Drucken mit Java nach einem einfachen Schema ab. Zuerst ist ein Objekt der Klasse »DocFlavor« beziehungsweise einer Unterklasse, etwa »DocFlavor.READER« oder »DocFlavor.URL« zu erzeugen. Diese inneren Klassen von »DocFlavor« besitzen eine Reihe von Konstanten, die den Dokumenttyp angeben, zum Beispiel »DocFlavor.READER.TEXT_HTML«. Ein besonders nützliches »DocFlavor« ist »DocFlavor

.INPUT_STREAM.AUTOSENSE«, denn es versucht, das Format selbstständig zu erkennen.
Im zweiten Schritt entsteht ein »AttributeSet«, das den Druckauftrag beschreibt. Mit diesen Informationen – »DocFlavor« und »AttributeSet« – beginnt die Suche nach einem passenden Print Service, also einem Drucker. Zuletzt erzeugt man einen Druckauftrag und ruft dessen »print«-Methode auf.
Aus API-Sicht ist »PrintService« ein Interface. Die Klasse »PrintServiceLookup« ist eine Factory für konkrete Implementationen von Print Services. »PrintService« wiederum ist eine Factory für Klassen, die das Interface »DocPrintJob« implementieren.

Job an Drucker

Ein kleines Beispiel soll das soeben beschriebene Verfahren verdeutlichen. In Listing 1 ist ein Java-Programm abgedruckt, das die übergebene Datei zum ersten verfügbaren Drucker schickt. Die Methode »printDocument()« (Zeilen 60 bis 76) zeigt alle oben beschriebenen Schritte. Das »DocFlavor« wird automatisch bestimmt, das gewünschte Papierformat ist A4. Die Zeilen ab 66 suchen den passenden Drucker. Wer die Auswahl offen lassen will, kann als Argumente für »DocFlavor« und die Attribute jeweils »null« übergeben.
Zeile 71 führt eine bisher nicht erwähnte Klasse ein. Der JPS kapselt Dokumente in Klassen, die das Interface »Doc« implementieren, zum Beispiel »SimpleDoc«. In der Dokumentation von »SimpleDoc« weist Sun explizit darauf hin, dass die Klasse nicht gewährleistet, dass ein Stream wie im Beispiel automatisch geschlossen wird. Bei lange laufenden Programmen muss eigene Software deshalb den Druckvorgang überwachen und den Stream gegebenenfalls selbst schließen.

Drucker finden

Die Methode »printDocument()« alleine findet auf Linux-Systemen keine Drucker. Doch die aktive Open-Source-Community hat das Problem mittlerweile gelöst. Auf [2] ist eine »PrintServiceLookup«-Implementation für das Internet-Printing-Protocol IPP zu finden, das von Cups verwendet wird, dem Drucksystem der gängigen Distributionen.
Download, Installation und Nutzung dieser Erweiterung sind einfach. Es genügt, »jipsi.jar« im »CLASSPATH« zu haben. In Listing 1 registriert das Programm in den Zeilen 48 bis 52 den entsprechenden Provider. Alternativ ließe sich die Klasse »IppPrintServiceLookup« auch direkt nutzen.
Der IPP-Print-Service verhielt sich beim Test eigenartig: Waren bei der Druckersuche »DocFlavor« und Attribute belegt, gehörten die erstellten Druckjobs dem Besitzer »anonymous«. Ohne diese Vorgaben gehörten sie dem aktuell angemeldeten Benutzer.

Druckdialoge

Das Beispielprogramm schickt den Druckauftrag an den ersten Drucker. Das ist natürlich nicht besonders elegant, grafische Anwendungen sollten die Wahl lieber dem Nutzer überlassen. Batch- und Kommandozeilen-Anwendungen erlauben es, den Drucker über das Attribut »PrinterName« auszuwählen.
Inzwischen gibt es im JPS einen recht ausgefeilten Druckdialog, siehe die Abbildungen 1 bis 3. Der unmittelbare Vergleich zum alten AWT-basierten Dialog

Listing 1: »PrintDemo.java«

```
022 import java.io.*;
023 import java.net.*;
024 import javax.print.*;
025 import javax.print.attribute.*;
026 import javax.print.attribute.standard.*;
027
028 import de.lohndirekt.print.*;
029
037 public class PrintDemo {
038
039   private static final String
040     CUPS_URI = "ipp://localhost:631";
041
048   private void registerService() throws Exception {
049     PrintServiceLookup psl =
050       new IppPrintServiceLookup(new URI(CUPS_URI),null,null);
051     PrintServiceLookup.registerServiceProvider(psl);
052   }
053
060   private void printDocument(String filename) throws Exception {
061     DocFlavor flavor = DocFlavor.INPUT_STREAM.AUTOSENSE;
062     PrintRequestAttributeSet attributes =
063       new HashPrintRequestAttributeSet();
064     attributes.add(MediaSizeName.ISO_A4);
065
066     PrintService[] pservices =
067       PrintServiceLookup.lookupPrintServices(flavor,attributes);
068     if (pservices.length > 0) {
069       DocPrintJob pj = pservices[0].createPrintJob();
070       FileInputStream stream = new FileInputStream(filename);
071       Doc doc = new SimpleDoc(stream,flavor,null);
072       pj.print(doc,attributes);
073     } else {
074       System.out.println("no printers available");
075     }
076   }
077
080   public static void main(String[] args) {
081     if (args.length != 1)
082       usage();
083     PrintDemo p = new PrintDemo();
084     try {
085       p.registerService();
086       p.printDocument(args[0]);
087     } catch (Exception e) {
088       e.printStackTrace();
089     }
090   }
091
098   private static void usage() {
099     System.out.println("usage: java " +
100       PrintDemo.class.getName() + " filename");
101     System.exit(3);
102   }
103 }
```

(siehe Abbildung 4) macht den Fortschritt des Print-API deutlich. Der Aufruf des Dialogs ist einfach (Listing 2). Die Methode ist ein Ausschnitt aus einer erweiterten »PrintDemoGUI«-Klasse, die hier nicht abgedruckt ist. Die Quellen sind wie üblich auf dem Linux-Magazin-Server unter [3] verfügbar.

Die einzige Methode der »ServiceUI«-Klasse heißt »printDialog()«. Sie erwartet eine Reihe von Parametern, die auch »null« sein dürfen. Wichtig sind ein Array mit »PrintServices« und zudem ein »AttributeSet«, das vor dem Aufruf Defaultwerte enthält und danach die veränderten Werte.

Die grafische Schnittstelle der druckerspezifischen Einstellungen bietet Raum für zukünftige Erweiterungen des JPS. Dafür gibt es jetzt schon eine Klasse »ServiceUIFactory«. Der aktuelle Druckdialog bietet aber kein API, um eigene Tabs in den Dialog einzubinden.

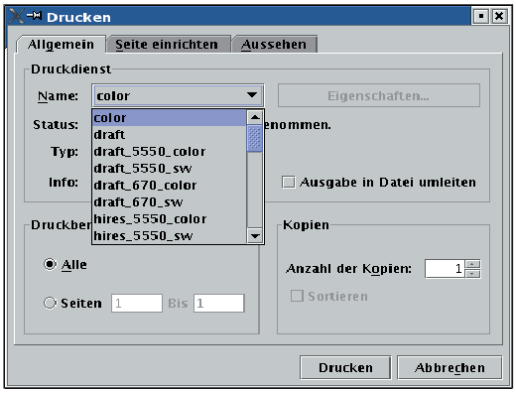


Abbildung 1: Print-Dialog des JPS (Reiter 1).

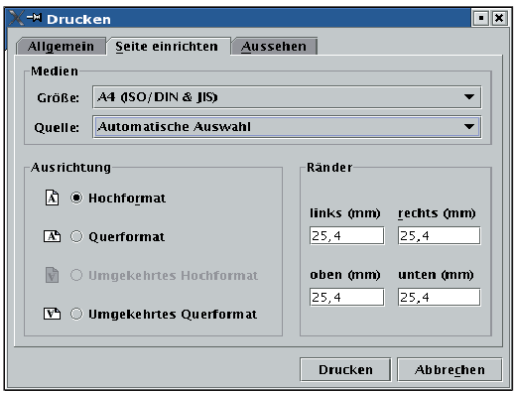


Abbildung 2: Print-Dialog des JPS (Reiter 2).

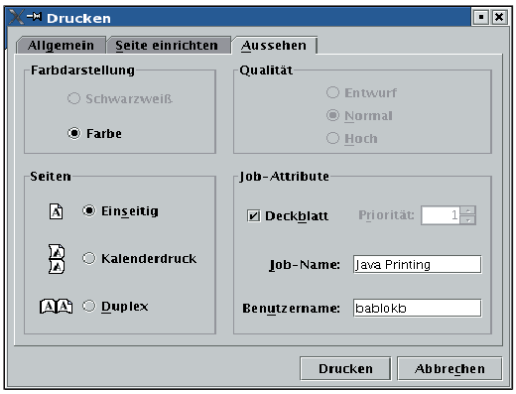


Abbildung 3: Print-Dialog des JPS (Reiter 3).

Listing 2: Grafische »getService()«-Methode

```
071 private PrintService getService() {
072     PrintService[] pservices =
073        PrintServiceLookup.lookupPrintServices(null,null);
074     iAttributes =  new HashPrintRequestAttributeSet();
075     iAttributes.add(MediaSizeName.ISO_A4);
076     return ServiceUI.printDialog(null,50,50,pservices,null,null,
077                                                  iAttributes);
078 }
```

Dateien als Druckziel

Der JPS unterstützt auch Dateien, die als Ziel für Druckaufträge dienen. Realisiert ist dies durch die abstrakte Klasse »StreamPrintService«, die das Interface »PrintService« implementiert und deshalb überall dort verwendet werden kann, wo normale Print-Services vorkommen. Als abstrakte Klasse wird »StreamPrintService« nicht direkt instanziert, sondern über eine Factory (eine Unterklasse von »StreamPrintServiceFactory«).

Das Drucken in eine Datei ist eigentlich eine Formatumwandlung. Das Ausgangsformat ist wie üblich über das »DocFlavor« beschreibbar, das Ergebnisformat über einen Mime-Typ. Das Vorgehen ist zweistufig: Zuerst fragt man über eine statische Methode von »StreamPrintServiceFactory« alle Factories ab, die ein vorgegebenes »DocFlavor« und den Ausgabe-Mime-Typ unterstützen. Gibt es eine solche Factory, dann erzeugt die »getPrintService()«-Methode der »StreamPrintServiceFactory« einen passenden »PrintService«.

Mit dem JDK liefert Sun eine Factory mit, die Grafikaufrufe aus Java2D in Postscript umwandeln kann. Damit lassen sich grafische Objekte als Dateien ausgeben, ohne auf die alten Klassen aus »java.awt.print« zurückgreifen zu müssen. Weitere Factories sind über das Service-Provider-Interface (SPI) selbst zu implementieren.

finally{}

Mit den hier beschriebenen Verfahren ist die Druckausgabe mit Java kein großes Problem mehr. Die Bibliothek JIPSI hilft bei Cups-Funktionen. Wer in Details einsteigen will, sollte unbedingt den „Java Print Service API User Guide“ (Teil der JDK-Dokumentation) durchlesen. Dort findet sich noch viel Interessantes, insbesondere zu den Attribute, die hier nur eine Nebenrolle spielten. *(ofr)*

Infos

[1] Bernhard Bablok, „Druck machen“: Linux-Magazin 04/00, S. 160, [http://www.linux-magazin.de/Artikel/ausgabe/2000/04/Print2D/print2d.html]

[2] JIPSI-Homepage: [http://sourceforge.net/projects/jipsi]

[3] Listings: [http://www.linux-magazin.de/Service/Listings/2004/12/Coffeeshop]

Der Autor

Bernhard Bablok arbeitet bei der AGIS mbH als Anwendungsentwickler. Wenn er nicht Musik hört, mit dem Radl oder zu Fuß unterwegs ist, beschäftigt er sich mit Themen rund um Objektorientierung. Er ist unter [coffee-shop@bablokb.de] zu erreichen.

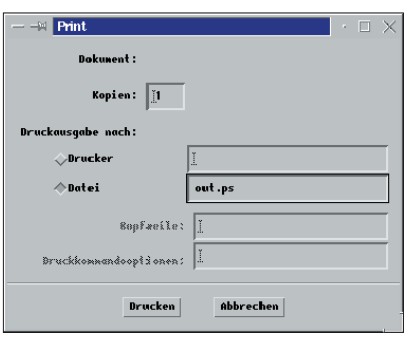


Abbildung 4: Der alte, AWT-basierte Druckdialog.