PDFlib GmbH

PDFlib, PDFlib+PDI, PPS

Eine Bibliothek für dynamisches PDF

PDFlib 9.0.1

Tutorial

Ausgabe für C, C++, Cobol, COM, Java, .NET, Objective-C,
Perl, PHP, Python, REALbasic, RPG, Ruby

PDFlib GmbH
Franziska-Bilek-Weg 9, D-80339 München
www.pdflib.com
Tel. +49 • 89 • 452 33 84-0
Fax +49 • 89 • 452 33 84-99

Bei Fragen können Sie die PDFlib-Mailing-Liste abonnieren und sich deren Archiv ansehen unter: tech.groups.yahoo.com/group/pdflib.

Vertriebsinformationen: sales@pdflib.com
Support für Inhaber einer kommerziellen PDFlib-Lizenz: support@pdflib.com (geben Sie bitte immer Ihre Lizenznummer an)

Inhaltsverzeichnis

0 Anwendung des PDFlib-Lizenzschlüssels

Einschränkungen der Evaluierungsversion. Alle Binärversionen von PDFlib, PDFlib+PDI und PPS, die PDFlib GmbH anbietet, sind als Evaluierungsversionen in vollem Funktionsumfang verwendbar, unabhängig davon, ob Sie eine kommerzielle Lizenz erworben haben oder nicht. Bei nicht lizenzierten Versionen wird jedoch quer über allen generierten Seiten der Demostempel *www.pdflib.com* ausgegeben Seiten, und die integrierte pCOS-Schnittstelle ist auf kleine Dokumente (bis zu 10 Seiten und 1 MB Dateigröße) begrenzt. Nicht lizenzierte Binärdateien dürfen nicht im produktiven Einsatz, sondern nur für die Evaluierung des Produkts verwendet werden. Der produktive Einsatz eines PDFlib GmbH Produkts erfordert eine gültige Lizenz.

Unternehmen, die an einer Lizenzierung von PDFlib interessiert sind und die Einschränkungen in der Evaluierungsphase oder ersten Demos vermeiden möchten, können ihre Firmen- und Projektdaten mit einer kurzen Erläuterung an *sales@pdflib.com* senden und einen temporären Lizenzschlüssel anfordern. (Wir behalten uns das Recht vor, Anforderungen von Evaluierungslizenzen abzulehnen, z.B. bei anonymen Anfragen).

Beachten Sie, dass es sich bei PDFlib, PDFlib+PDI und PDFlib Personalization Server (PPS) um verschiedene Produkte handelt, die unterschiedliche Lizenzschlüssel erfordern, selbst wenn sie in einem einzigen Paket ausgeliefert werden. Ein Lizenzschlüssel für PPS gilt auch für PDFlib+PDI und PDFlib, ein Schlüssel für PDFlib+PDI gilt auch für PDFlib, aber nicht umgekehrt. Alle Lizenzschlüssel sind plattformabhängig und können nur auf der Plattform eingesetzt werden, für die sie erworben wurden.

Nachdem Sie einen Lizenzschlüssel erworben haben, müssen Sie ihn anwenden, damit der Demostempel unterdrückt wird. Zur Anwendung des Lizenzschlüssels gibt es verschiedene Methoden, die im folgenden erläutert werden.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/license_key`.

Windows-Installationsroutine. Wenn Sie die Windows-Installationsroutine verwenden, können Sie den Lizenzschlüssel bei der Produktinstallation eingeben. Dieser wird dann zur Registry hinzugefügt (siehe unten).

Anwenden eines Lizenzschlüssels mit einem API-Aufruf zur Laufzeit. Fügen Sie in Ihrem Skript oder Programm eine Zeile ein, die den Lizenzschlüssel zur Laufzeit setzt. Die Option *license* muss dabei unmittelbar nach der Instantiierung des PDFlib-Objekts gesetzt werden (in C: nach *PDF_new( )*). Die Syntax hängt von der jeweiligen Programmiersprache ab:

- In COM/VBScript und REALbasic:

```
oPDF.set_option "license", "...Ihr Lizenzschlüssel..."
```

- In C++, Java, .NET/C#, Python und Ruby:

```
PDF_set_option("license=...Ihr Lizenzschlüssel...")
```

- In C:

```
PDF_set_option(p, "license", "...Ihr Lizenzschlüssel...")
```

- In Objective-C:

```
[pdflib set_option: @"license=...Ihr Lizenzschlüssel..."];
```

- In Perl und PHP:

```
$p->set_option("license=...Ihr Lizenzschlüssel...")
```

- In RPG:

```
c            callp      PDF_set_option(p:%ucs2('license=...Ihr Lizenz-schlüssel...')
```

Verwendung einer Lizenzdatei. Statt den Lizenzschlüssel mit einem Aufruf zur Laufzeit zu übergeben, können Sie ihn folgendermaßen in eine Textdatei eintragen (alle PDFlib-Distributionen enthalten dafür die Vorlage *licensekeys.txt*). Mit einem '#'-Zeichen beginnende Zeilen enthalten Kommentare und werden ignoriert. Die zweite Zeile enthält Informationen zur Version der Lizenzdatei selbst:

```
# Lizenzinformation für Produkte der PDFlib GmbH
PDFlib license file 1.0
PDFlib      9.0.1           ...Ihr Lizenzschlüssel...
```

Sie können Lizenzschlüssel für verschiedene Produkte der PDFlib GmbH in die Lizenzdatei aufnehmen, wobei jeder Schlüssel in einer eigenen Zeile stehen muss. Sie können auch Lizenzschlüssel für verschiedene Plattformen aufnehmen, so dass die Lizenzdatei für mehrere Plattformen gemeinsam benutzt werden kann. Sie können Lizenzdateien folgendermaßen konfigurieren:

- Eine Datei namens *licensekeys.txt* wird an allen vorgegebenen Stellen gesucht (siehe »Standard-Dateisuchpfade«, Seite 13).
- Sie können die Option *licensefile* mit der *set_option ()* API-Funktion angeben:

```
p.set_option("licensefile={/path/to/licensekeys.txt}");
```

- Sie können eine Umgebungsvariable (Shell) anlegen, die auf die Lizenzdatei verweist. Unter Windows öffnen Sie die Systemsteuerung und wählen *System, Erweiterte Systemeinstellungen, Erweitert, Umgebungsvariablen*. Unter Unix verwenden Sie folgenden Befehl:

```
export PDFLIBLICENSEFILE=/path/to/license/file
```

- Auf i5/iSeries-Systemen muss die Lizenzdatei in ASCII kodiert sein (siehe Option *asciifile)*. Die Lizenz-Datei kann wie folgt angegeben werden. (Dieser Befehl kann im Startup-Programm QSTRUP angegeben werden und funktioniert für alle PDFlib GmbH Produkte.)

```
ADDENVVAR ENVVAR(PDFLIBLICENSEFILE) VALUE('/PDFlib/9.0/licensefile.txt') LEVEL(*SYS)
```

Lizenzschlüssels in der Registry. Unter Windows können Sie den Namen der Lizenzdatei auch unter folgendem Registry-Schlüssel eintragen:

```
HKLM\Software\PDFlib\PDFLIBLICENSEFILE
```

Alternativ können Sie den Lizenzschlüssel auch direkt unter einem der folgenden Registry-Schlüssel eintragen:

```
HKLM\SOFTWARE\PDFlib\PDFlib9\license
HKLM\SOFTWARE\PDFlib\PDFlib9\9.0.1\license
```

Der MSI-Installer schreibt den Lizenzschlüssel in den letzten dieser Einträge.

Hinweis *Seien Sie vorsichtig beim manuellen Zugriff auf die Registry von 64-Bit-Windows-Systemen: wie üblich funktionieren 64-Bit PDFlib-Binärdateien mit der 64-Bit-Ansicht der Windows-Registry, während 32-Bit PDFlib-Binärdateien auf einem 64-Bit-System mit der 32-Bit-Ansicht der Registry funktionieren. Wenn Sie Registry-Schlüssel für einen 32-Bit-Produkt manuell hinzufügen wollen, stellen Sie sicher, dass Sie die 32-Bit-Version des* `regedit`*-Tools verwenden. Rufen Sie es über den Dialog* `Start` *wie folgt auf:*

```
%systemroot%\syswow64\regedit
```

Standard-Dateisuchpfade. Unter Unix, Linux, OS X und i5/iSeries werden einige Verzeichnisse standardmäßig nach Dateien durchsucht, sogar ohne Angabe von Pfad- und Verzeichnisnamen. Bevor die UPR-Datei (welche zusätzliche Suchpfade enthalten kann) durchsucht und gelesen wird, werden die folgenden Verzeichnisse durchsucht:

```
<rootpath>/PDFlib/PDFlib/9.0/resource/cmap
<rootpath>/PDFlib/PDFlib/9.0/resource/codelist
<rootpath>/PDFlib/PDFlib/9.0/resource/glyphlst
<rootpath>/PDFlib/PDFlib/9.0/resource/fonts
<rootpath>/PDFlib/PDFlib/9.0/resource/icc
<rootpath>/PDFlib/PDFlib/9.0
<rootpath>/PDFlib/PDFlib
<rootpath>/PDFlib
```

Unter Unix, Linux und OS X wird *<roothpath>* zuerst durch */usr/local* ersetzt und dann durch das HOME-Verzeichnis. Unter i5/iSeries ist *<roothpath>* leer.

Standardnamen für Lizenz- und Ressource-Dateien. Standardmäßig werden die folgenden Dateinamen in den Standard-Verzeichnissuchpfaden gesucht:

```
licensekeys.txt            (Lizenzdatei)
pdflib.upr                 (Ressource-Datei)
```

Mit dieser Funktion können Sie eine Lizenzdatei verwenden, ohne eine Umgebungsvariable oder Laufzeit-Option anzugeben.

Updates und Upgrades. Wenn Sie ein Update erworben haben, d.h. die ältere Version eines Produkts durch eine neuere Version desselben Produkts ersetzen, oder ein Upgrade durchführen, d.h. von PDFlib auf PDFlib+PDI bzw. PPS oder von PDFlib+PDI auf PPS umsteigen oder als Teil Ihres Supportvertrags einen neuen Lizenzschlüssel erhalten haben, müssen Sie den neuen Lizenzschlüssel anwenden, den Sie mit dem Update oder Upgrade erhalten haben. Der alte Lizenzschlüssel darf dann nicht mehr zum Einsatz kommen.

Testen unlizenzierter Funktionen. Ohne Lizenzschlüssel lassen sich alle Funktionen ohne Einschränkungen testen. Sobald Sie auf die Software einen für ein bestimmtes Produkt gültigen Lizenzschlüssel anwenden, sind Funktionen höherer Produktstufen

nicht mehr verfügbar. Nach der Installation eines gültigen PDFlib-Lizenzschlüssels beispielsweise stehen die PDI-Funktionen nicht mehr zum Test zur Verfügung. Genauso haben Sie keinen Zugang zu den Personalisierungsfunktionen (Blockfunktionen), nachdem der PDFlib+PDI-Lizenzschlüssel installiert ist.

Wurde bereits ein Lizenzschlüssel für ein Produkt installiert, können Sie ihn durch den Pseudo-Lizenzstring »0« (Ziffer Null) ersetzen, um die Funktionalität einer höheren Produktstufe zu testen. Damit werden alle deaktivierten Funktionen wieder aktiviert, und der Demostempel erscheint wieder auf allen Seiten.

Lizenzvarianten. Es gibt verschiedene Lizenzierungsmöglichkeiten für die Verwendung von PDFlib auf einem oder mehreren Servern und für die Weitergabe von PDFlib in eigenen Produkten. Wir bieten außerdem Support- und Wartungsverträge an. Einzelheiten zur Lizenzierung und das PDFlib-Bestellformular finden Sie in der PDFlib-Distribution. Bitte wenden Sie sich an uns, wenn Sie Fragen haben oder eine kommerzielle PDFlib-Lizenz beziehen möchten:

PDFlib GmbH, Lizenzabteilung
Franziska-Bilek-Weg 9, D–80339 München
www.pdflib.com
Tel. +49 • 89 • 452 33 84-0
Fax +49 • 89 • 452 33 84-99
Bestellung: *sales@pdflib.com*
Support für PDFlib-Lizenznehmer: *support@pdflib.com*

1 Einführung

1.1 Roadmap für Dokumentation und Beispiele

Zum erfolgreichen Einsatz von PDFlib-Produkten stehen Ihnen unten aufgeführte Materialien zur Verfügung.

Minibeispiele für alle Sprachbindungen. Die *Minibeispiele* (hello, image, pdfclock usw.) werden in allen Paketen und für alle Sprachbindungen mitgeliefert. Sie enthalten kurzen Beispielcode für die Textausgabe sowie für Rasterbilder und Vektorgrafiken. Die Minibeispiele dienen in erster Linie zum Testen Ihrer PDFlib-Installation und für eine kurze Übersicht über die Programmierung von PDFlib-Anwendungen.

Starter-Beispiele für alle Sprachbindungen. Die Starter-Beispiele sind in allen Paketen für verschiedene Sprachbindungen enthalten. Sie bieten eine gute Grundlage für wichtige Themen und befassen sich mit einfacher Text- und Rasterbildausgabe, Textflow- und Tabellenformatierung, sowie der Erstellung von PDF/A, PDF/X, PDF/VT und PDF/UA. Die Starter-Beispiele zeigen elementare Techniken zur Durchführung verschiedener Aufgaben mit PDFlib-Produkten. Diese Beispiele sollten Sie sich auf jeden Fall ansehen.

PDFlib-Tutorial. Das vorliegende PDFlib-Tutorial, das in allen Paketen als PDF-Dokument enthalten ist, liefert eine detaillierte Beschreibung wichtiger Programmierkonzepte einschließlich kurzer Codebeispiele. Sobald Sie über die Starter-Beispiele hinausgekommen sind, sollten Sie die jeweils relevanten Themen im PDFlib-Tutorial nachlesen.

Hinweis *Im vorliegenden PDFlib-Tutorial sind die meisten Beispiele in Java kodiert (mit Ausnahme der sprachspezifischen Beispiele in Kapitel 2, »Sprachbindungen von PDFlib«, Seite 27, und einigen C-spezifischen Beispielen, die entsprechend gekennzeichnet sind). Die Syntax unterscheidet sich in den verschiedenen Sprachen zwar in den Details, die grundlegenden Konzepte der PDFlib-Programmierung sind jedoch für alle Programmiersprachen dieselben.*

PDFlib-Referenz. Die PDFlib-Referenz, die in allen Paketen als PDF-Dokument verfügbar ist, enthält eine umfassende Beschreibung aller Funktionen und Optionen, aus denen sich das PDFlib Application Programming Interface (API) zusammensetzt. Die PDFlib-Referenz ist die ultimative Quelle zum Nachschlagen von unterstützten Optionen, Eingabebedingungen und anderen einzuhaltenden Programmiervorschriften. Beachten Sie bitte, dass manche andere Referenzdokumente unvollständig sind, z.B. das Javadoc API-Listing für PDFlib oder die PDFlib-Funktionsliste von *php.net*. Achten Sie deshalb unbedingt darauf, bei der PDFlib-Programmierung immer die vollständige PDFlib-Referenz zu verwenden.

pCOS-Pfadreferenz. Die pCOS-Schnittstelle kann verwendet werden, um zahlreiche Eigenschaften von PDF-Dokumenten abzufragen. pCOS ist in PDFlib+PDI und PPS enthalten. Die pCOS-Pfadreferenz enthält eine Beschreibung der Pfad-Syntax zur Adressierung einzelner Objekte innerhalb eines PDF-Dokuments, um deren Werte abrufen zu können.

PDFlib Cookbook. Das *PDFlib Cookbook* ist eine Sammlung von PDFlib-Codefragmenten, die sich der Lösung spezieller Probleme widmen. Die meisten Cookbook-Beispiele sind für Java und PHP implementiert, lassen sich aber leicht an andere Programmiersprachen anpassen, da sich das PDFlib-API in den verschiedenen unterstützten Sprachbindungen kaum unterscheidet. Das *PDFlib Cookbook* ist eine Sammlung von Programmierbeispielen und unter folgender Adresse verfügbar:

```
www.pdflib.com/pdflib-cookbook/
```

pCOS Cookbook. Das *pCOS Cookbook* ist eine Sammlung von Codefragmenten für die pCOS-Schnittstelle, die in PDFlib+PDI und PPS enthalten ist. Die pCOS-Schnittstelle kann verwendet werden, um zahlreiche Eigenschaften von PDF-Dokumenten abzufragen. Das *pCOS Cookbook* ist unter der folgenden Adresse verfügbar:

```
www.pdflib.com/pcos-cookbook/
```

TET Cookbook. PDFlib TET (Text Extraction Toolkit) ist ein separates Produkt zum Extrahieren von Text und Rasterbildern aus PDF-Dokumenten. Es kann mit PDFlib+PDI kombiniert werden, um PDF-Dokumente auf der Basis ihres Inhalts zu verarbeiten. Das *TET Cookbook* ist eine Sammlung von Codefragmenten für TET. Es enthält eine Reihe von Beispielen, die die Kombination von TET und PDFlib+PDI demonstrieren, z.B. Einfügen von Weblinks oder Lesezeichen basierend auf dem Text einer Seite, Markieren von Suchbegriffen, textbasiertes Teilen eines Dokuments, Erstellen eines Inhaltsverzeichnisses usw. Das *TET Cookbook* ist unter der folgenden Adresse verfügbar:

```
www.pdflib.com/tet-cookbook/
```

1.2 Programmierung mit PDFlib

Was ist PDFlib? PDFlib ist eine Entwicklungskomponente, mit der Sie Dateien im Portable Document Format (PDF) von Adobe erstellen können. PDFlib fungiert als Backend für Ihre Programme. Sie als Programmierer sind verantwortlich für die Verwaltung der zu verarbeitenden Daten, und PDFlib übernimmt die Generierung des PDF-Codes, der die Daten grafisch darstellt. Dank PDFlib brauchen Sie sich nicht um die Details des komplexen PDF-Formats zu kümmern. PDFlib bietet zahlreiche Methoden, die Sie bei der Formatierung der PDF-Ausgabe unterstützen. Die PDFlib-Pakete enthalten mehrere Produkte in einer einzigen Binärdatei:

- PDFlib bietet alle Funktionen zur Erstellung von PDF-Ausgabe inklusive Text, Vektorgrafik, Rasterbildern und Hypertext-Elementen. PDFlib bietet leistungsfähige Formatierungsfunktionen zur Platzierung von ein- oder mehrzeiligem Text, von Bildern und zur Tabellenerstellung.
- PDFlib+PDI enthält neben allen PDFlib-Funktionen zusätzlich die PDF-Importbibliothek PDI, mit der sich Seiten aus vorhandenen PDF-Dokumenten in die generierte Ausgabe integrieren lassen. Außerdem enthalten ist die pCOS-Schnittstelle zur Abfrage beliebiger PDF-Objekte aus einem importierten Dokument (z.B. alle auf der Seite verwendeten Fonts, Metadaten und vieles mehr).
- Der PDFlib Personalization Server (PPS) enthält neben PDFlib+PDI zusätzliche Funktionen zum automatischen Füllen von PDFlib-Blöcken. Blöcke stellen Platzhalter auf der Seite dar, die sich mit Text, Bildern oder PDF-Seiten füllen lassen. Sie können interaktiv mit dem PDFlib-Block-Plugin für Adobe Acrobat (OS X oder Windows) erzeugt und automatisch mit PPS gefüllt werden. Das Plugin gehört zum Lieferumfang von PPS.

Wie wird PDFlib eingesetzt? PDFlib ist auf verschiedensten Plattformen einsetzbar, unter anderem auf Unix, Windows, OS X und EBCDIC-basierten Systemen wie IBM i5/iSeries und zSeries. PDFlib ist in der Programmiersprache C geschrieben, unterstützt aber den Zugriff von verschiedensten anderen Programmiersprachen und -umgebungen – den so genannten Sprachbindungen. Dazu gehören alle für eigenständige Applikationen und Webanwendungen gängige Sprachen. Das *Application Programming Interface (API)* ist einfach zu erlernen und für alle Sprachbindungen gleich. Derzeit werden folgende Sprachbindungen unterstützt:

- COM für den Einsatz mit VB, ASP mit VBScript oder JScript, Windows Script Host usw.
- ANSI C und C++
- Cobol (IBM zSeries)
- Java inklusive J2EE Servlets und JSP
- .NET für den Einsatz mit C#, VB.NET, ASP.NET usw.
- Objective-C (OS X, iOS)
- PHP
- Perl
- Python
- REALbasic
- RPG (IBM i5/iSeries)
- Ruby einschließlich Ruby on Rails

Wozu kann man PDFlib verwenden? PDFlib wird hauptsächlich dazu verwendet, innerhalb eigener Software oder auf einem Webserver dynamisch PDF zu generieren. Genauso wie sich auf dem Webserver dynamisch HTML-Seiten erzeugen lassen, können Sie ein PDFlib-Programm schreiben, das dynamisch PDF erzeugt, zum Beispiel in Reaktion auf Benutzereingaben oder auf Basis von Daten, die dynamisch aus der Datenbank des Webservers abgerufen werden. Die Arbeitsweise von PDFlib bietet mehrere Vorteile:

- PDFlib kann unmittelbar in die Anwendung, die für die Datengenerierung zuständig ist, integriert werden. Damit wird der umständliche Weg über »Anwendung – PostScript – Acrobat Distiller – PDF« überflüssig.
- Aufgrund dieser Vereinfachung bietet PDFlib die schnellstmögliche Methode zur Generierung von PDF, die sich ideal für den Einsatz im Web eignet.
- Thread-Sicherheit, stabile Speicherverwaltung und Fehlerbehandlung erlauben die Implementierung von Server-Anwendungen mit höchsten Performance-Anforderungen.
- PDFlib ist für eine Vielzahl von Betriebssystemen und Entwicklungsumgebungen verfügbar.

Anforderungen an den Einsatz von PDFlib. Mit PDFlib können Sie PDF generieren, ohne sich vorher durch die PDF-Spezifikation zu quälen. Der Einsatz von PDFlib setzt zwar kein Wissen über die technischen Einzelheiten des PDF-Formats voraus, ein allgemeines Verständnis von PDF kann jedoch nicht schaden. Für einen optimalen Einsatz von PDFlib sollte der Anwendungsprogrammierer mit dem zugrunde liegenden Grafikmodell von PDF vertraut sein. Aber auch ein Anwendungsprogrammierer, der bereits mit einem Grafik-API für Bildschirmanzeige oder Ausdruck gearbeitet hat, sollte sich ohne große Probleme auf das PDFlib-API umstellen können.

1.3 Was ist neu in PDFlib/PDFlib+PDI/PPS 9?

Die folgende Liste gibt eine Übersicht über die wichtigsten neuen oder erweiterten Funktionen von PDFlib, PDFlib+PDI/PPS 9.0 sowie dem Block-Plugin 5.0. Für eine detaillierte Liste aller neuen Features siehe Tabelle 1.1 sowie die PDFlib-Referenz.

Erstellen von PDF/A-2 und PDF/A-3. PDFlib unterstützt zwei neue Teile des PDF/A-Standards für die Archivierung. PDF/A-2 basiert auf PDF 1.7 und unterstützt Transparenz, JPEG-2000-Kompression, Ebenen und viele weitere Funktionen. Während PDF/A-2 die Einbettung von PDF/A-1- und PDF/A-2-Dokumenten ermöglicht, erlaubt PDF/A-3 die Einbettung beliebiger Dateitypen.

Erstellen von Tagged PDF und PDF/UA. Die Erstellung von Tagged PDF wurde durch viele praktische Funktionen vereinfacht, etwa vereinfachtes Anbringen von Tags und automatische Erstellung von Tags für Artefakte. Der Tabellenformatierer von PDFlib erstellt automatisch alle Tabellen-Tags. Tagged PDF einschließlich Strukturelementen kann mit PDI importiert werden.

Gemäß dem PDF/UA-Standard (Universal Accessibility) können barrierefreie PDF-Dokumente erstellt werden. PDF/UA basiert auf PDF 1.7 und verbessert Tagged PDF für die Barrierefreiheit, vergleichbar mit WCAG 2.0 (*Web Content Accessibility Guidelines)* für das Web.

Erstellen von PDF/VT. PPDF/VT ist ein Standard für optimiertes PDF für den Druck von variablen Daten und Transaktionsdokumenten. Mit PDFlib können Dokumente erstellt werden, die den Standards PDF/VT-1, PDF/VT-2 oder PDF/VT-2s nach ISO 16612-2 für den Druck von variablen Daten *(Variable Document Printing, VDP)* entsprechen. Document Part Metadata (DPM) kann gemäß dem PDF/VT-Standard angehängt werden.

Importieren von SVG-Grafiken. Mit PDFlib können Sie Vektorgrafiken im SVG-Format importieren. SVG *(Scalable Vector Graphics)* ist das Standardformat für Vektorgrafik im Web und wird von allen gängigen Browsern unterstützt.

Fontverarbeitung und Textausgabe. Die Font- und Textverarbeitung von PDFlib wurde wie folgt erweitert:

- Ideographic Variation Sequences (IVS) zur Auswahl von CJK-Glyphvarianten
- WOFF-Fonts *(Web Open Font Format)*, ein neues Containerformat für TrueType- und OpenType-Fonts, das vom W3C spezifiziert wurde
- SVG-Fonts, d.h. Vektor-Fonts im SVG-Format
- CEF-Fonts *(Compact Embedded Font)*, eine Variante von OpenType zum Einbetten von Fonts in SVG-Grafiken
- Unterstützung für alle Unicode-Normalisierungsformen (NFC, NFKC usw.)
- automatische Erstellung von UPR-Font-Konfigurationsdateien mit allen in beliebig vielen Verzeichnissen verfügbaren Schriften

PDF-Import mit PDFlib+PDI. Die folgenden Funktionen sind neu in der PDF-Importbibliothek PDI:

- Tagged PDF kann einschließlich Strukturelementen importiert werden.
- Definitionen von Ebenen können importiert werden.

PDFlib Personalization Server (PPS) und Block-Plugin. Die folgenden Funktionen sind neu in PPS:

- Den neuen Blocktyp G*raphics* können Sie verwenden, um PDFlib-Blöcke mit SVG-Grafiken zu füllen.
- PDFlib-Blöcke können Sie nicht nur mit PPS füllen, sondern auch ins Ausgabe-PDF importieren.
- Einige neue Blockeigenschaften wurden eingeführt.

PDFlib-Blöcke per Programm erstellen. Neben der interaktiven Erstellung von PDFlib-Blöcken mit dem PDFlib Block-Plugin können Sie die Erstellung von PDFlib-Blöcken auch mit PPS programmieren. Bestehende PDFlib-Blöcke aus importierten Dokumenten können in die generierte PDF-Ausgabe kopiert werden. Diese Funktionen ermöglichen anspruchsvolle Workflows für die Dokumentzusammensetzung, bei denen Sie selbst Vorlagen für PPS programmieren können.

PDF Object Creation API (POCA). POCA bietet eine Reihe von Methoden für die Erstellung von Low-Level PDF-Objekten, die in die generierte PDF-Ausgabe übernommen werden. POCA kann für folgende Zwecke verwendet werden:

- Erstellen von Document Part Metadata (DPM) für PDF/VT
- Programmatische Erstellung von PDFlib-Blöcken für die Verwendung mit PPS
- Erstellen von Argumentlisten für Rich-Media-Annotationen (z.B. Flash)

Einbettung von Multimedia-Inhalten. Mit PDFlib können Sie Rich-Media-Annotationen mit Flash-, Audio-, Movie- oder 3D-Inhalten erstellen. Die Multimedia-Inhalte können mit JavaScript und PDF-Aktionen gesteuert werden. Die folgenden neuen Multimedia-Funktionen stehen zur Verfügung:

- Rich-Media-Annotationen
- PDF-Aktionen zum Ansteuern von Rich-Media-Objekten
- Flash-basierte Navigatoren für individuelle Präsentation von PDF-Portfolios

Verbesserter Verschlüsselungsalgorithmus. PDFlib unterstützt die PDF-Dokumentverschlüsselung gemäß Acrobat X/XI. Dieses Verschlüsselungsverfahren basiert auf AES-256 und ist in PDF 1.7 Extension Level 8 sowie PDF 2.0 gemäß ISO 32000-2 spezifiziert.

Weitere Verbesserungen. Folgende Verbesserungen wurden implementiert:

- Verbesserung der Tabellen- und Textflow-Formatierer
- Mehrere Komfortfunktionen für die Erstellung von Pfadobjekten mit geometrischen Formen
- Verbesserung des Imports von Bildern im Format JPEG 2000
- Abfrage der Eigenschaften von Dateien im PDFlib Virtual Filesystem (PVF)
- Die meisten Beschränkungen für die Gültigkeitsbereiche von Funktionsaufrufen wurden aufgehoben; z. B. können Seiten, Pattern und Templates nun beliebig verschachtelt werden.

1.4 Funktionalität von PDFlib

Tabelle 1.1 listet die Funktionen zum Erzeugen von PDF auf. Neue oder erweiterte Funktionen sind gekennzeichnet.

Tabelle 1.1 Funktionsübersicht von PDFlib

Kategorie	Funktionalität
PDF-Varianten	PDF 1.4 – PDF 1.7 Extension Level 8[1] und PDF 2.0 (Acrobat 5 – XI)
	Linearisiertes (web-optimiertes) PDF für Byteserving über das Web
	Ausgabe mit großem Volumen und beliebiger PDF-Dateigröße (über 10 GB)
ISO-Standards für PDF	ISO 32000-1: standardisierte Version von PDF 1.7
	ISO 32000-2 (Entwurf): PDF 2.0[1]
	ISO 15930: PDF/X-1/3/4/5 für die Grafikindustrie
	ISO 19005-1/2/3: PDF/A-1/2/3 für die Archivierung
	ISO 16612-2: PDF/VT-1/2 für den Druck von variablen Daten und Transaktionsdokumenten
	ISO 14289-1: PDF/UA-1 für Barrierefreiheit[1]
Fonts	TrueType- (TTF und TTC) und PostScript-Type-1-Fonts
	OpenType-Fonts mit PostScript- oder TrueType-Zeichenbeschreibungen (TTF, OTF)
	WOFF-Fonts (Web Open Font Format), ein vom W3C spezifiziertes Containerformat für Fonts im Web[1]
	CEF-Fonts (Compact Embedded Font), eine OpenType-Variante für die Fonteinbettung in SVG[1]
	SVG-Fonts, also Fonts, die das SVG-Format zur Beschreibung der Glyph-Umrisslinien benutzen[1]
	Unterstützung für zahlreiche OpenType Layout-Features für westliche und CJK-Textausgabe, z.B. Ligaturen, Kapitälchen, Mediävalziffern, Swash-Zeichen, vereinfachte/traditionelle asiatische Formen, vertikale Alternativformen
	Verwendung der im System installierten Schriften unter Windows und OS X (»host fonts«)
	Fonteinbettung für alle Fonttypen; Untergruppenbildung (Subsetting) für TrueType-, OpenType- und Type-3-Fonts
	Benutzerdefinierte (Type 3) Fonts für Bitmap-Schriften oder Kundenlogos
	EUDC- und SING-Fonts (Glyphlets) für asiatische Gaiji-Zeichen
	Fallback-Fonts (fehlende Glyphen werden aus einem Ersatz-Font entnommen)
Textausgabe	Textausgabe in beliebigen Fonts; unterstrichener, überstrichener, durchgestrichener Text
	Glyphen eines Fonts können über numerische Werte, Unicode-Werte oder Glyphnamen adressiert werden.
	Unterschneidung (Kerning) für optimalen Zeichenabstand
	Künstliche Fett-, Kursiv- und Schattenschrift
	Text auf einem Pfad positionieren
	Konfigurierbare Ersatzdarstellung für fehlende Glyphen
Barrierefreiheit	Erzeugung von Tagged PDF für Barrierefreiheit (Accessibility), Umfließen von Text und verbesserte Weiterverwendung des Seiteninhalts
	Vereinfachtes Anbringen von Tags in allen Funktionen zur Erstellung von Seiteninhalten[1]
	Automatische Erstellung von Tags für Tabellen und Artefakte[1]
	PDF/UA-1 für Barrierefreiheit (Universal Accessibility), WCAG 2.0 (Web Content Accessibility Guidelines)[1]
	Zusätzliche Strukturelement-Typen und -Attribute[1]
Internationalisierung	Unicode für Seitenbeschreibungen, interaktive Elemente und Dateinamen; UTF-8-, UTF-16- und UTF-32-Kodierung, Unicode-Normalisierungsformen[1]
	CJK-Fonts und CMaps für Chinesisch, Japanisch und Koreanisch

Tabelle 1.1 Funktionsübersicht von PDFlib

Kategorie	Funktionalität
	Unterstützung zahlreicher 8-Bit- und traditioneller CJK-Encodings (z.B. SJIS; Big5)
	Ideographic variation sequences (IVS)[1] zur Auswahl von Glyphvarianten
	Vertikale Schreibrichtung für Chinesisch, Japanisch und Koreanisch
	Shaping für komplexe Schriftsysteme, z.B. Arabisch, Thai, Devanagari
	Bidirektionale Textformatierung für linksläufige Schriften, z.B. Arabisch und Hebräisch
SVG-Vektorgrafik	Import von Vektorgrafiken im SVG-Format[1]
Rasterbilder	Rasterbilder in den Formaten BMP, GIF, PNG, TIFF, JBIG2, JPEG, JPEG 2000 und CCITT
	Abfragen von Rasterbildinformationen (Pixelgröße, Auflösung, ICC-Profile, Beschneidungspfade usw.)
	Auswertung von Beschneidungspfaden in TIFF- und JPEG-Rasterbildern
	Auswertung des Alphakanals (Transparenz) in TIFF- und PNG-Rasterbildern
	Bildmasken (eingefärbte Bilder), Einfärben von Bildern mit einer Schmuckfarbe
Farbe	Graustufen, RGB (numerische, hexadezimale Strings, HTML-Farbnamen), CMYK und CIE L*a*b*
	Integrierte Schmuckfarbtabellen für PANTONE® (incl. PANTONE® Goe™ und PANTONE+)[1] und HKS®
	Benutzerdefinierte Schmuckfarben
Farbmanagement	ICC-basierte Farbe mit ICC-Farbprofilen; Unterstützung von ICC 5 Profilen
	Rendering-Intent für Text, Vektorgrafiken und Rasterbilder
	ICC-Profile als Druckausgabe-Bedingung für PDF/A und PDF/X
Archivierung	PDF/A-1a/1b, PDF/A-2a/b/u und PDF/A-3a/b/u
	XMP Extension-Schemas für PDF/A
Grafische Industrie	PDF/X-1a, PDF/X-3, PDF/X-4, PDF/X-4p, PDF/X-5p, PDF/X-5pg
	Einbettung eines ICC-Profils für die Druckausgabe-Bedingung oder Angabe einer extern referenzierten Druckausgabe-Bedingung
	Externer Grafikinhalt (referenzierte Seiten) für PDF/X-5p und PDF/X-5pg
	Einstellungen für Überdrucken, Aussparen usw.
Variable Document Printing (VDP)	PDF/VT-1, PDF/VT-2 und PDF/VT-2s für den Druck von variablen Daten und Transaktionsdokumenten[1]
Textflow-Formatierung	Formatierung von Text in einen oder mehrere rechteckige oder beliebig geformte Bereiche unter Anwendung von Silbentrennung (benutzerdefinierte Trennstellen erforderlich), Schrift- und Farbwechsel, verschiedenen Ausrichtungsverfahren, Tabulatoren, Führungszeichen, Steueranweisungen; fortgeschrittener Zeilenumbruch mit sprachspezifischer Verarbeitung
	Flexible Platzierung und Formatierung von Rasterbildern
	Text kann Bild oder Bild-Beschneidungspfad umfließen
Tabellen-formatierung	Tabellenformatierer platziert Zeilen und Spalten und berechnet automatisch deren Größe, wobei zahlreiche Optionen berücksichtigt werden. Tabellen können sich über mehrere Seiten erstrecken.
	Zellen können ein- oder mehrzeiligen Text, Rasterbilder, Vektorgrafik[1], PDF-Seiten, Pfadobjekte, Annotationen und Formularfelder enthalten.
	Tabellenzellen können umrandet oder gefüllt werden.
	Flexible Stempelfunktion
	Matchbox-Konzept zur Referenzierung der Koordinaten platzierter Bilder oder anderer Objekte[1]
Vektorgrafik	Basisfunktionen für Vektorgrafik: Linienzüge, Kurvenzüge, Kreise, Ellipsen, Rechtecke usw.
	Farbverläufe, Füllen von Flächen und Durchziehen von Linien mit Mustern
	Transparenz und Farbmischmodus

Tabelle 1.1 Funktionsübersicht von PDFlib

Kategorie	Funktionalität
	Wiederverwendbare Pfadobjekte und aus Rasterbildern importierte Beschneidungspfade
Ebenen	Optionaler, selektiv anzeigbarer Seiteninhalt
	Anmerkungen, Kommentare und Formularfelder können auf Ebenen platziert werden
Sicherheit	Verschlüsselung von PDF-Dokumenten oder -Anhängen mit 128/256-Bit AES[1]- oder RC4-Verschlüsselungsalgorithmus
	Unicode-Passwörter
	Festlegen von Dokumentberechtigungen (z.B. Drucken oder Kopieren nicht zulässig)
Interaktive Elemente	Erzeugung von Formularfeldern mit allen Feldoptionen und JavaScript
	Erzeugung von Formularfeldern für Barcodes
	Erzeugung von Aktionen für Lesezeichen, Anmerkungen, Öffnen/Schließen der Seite und andere Ereignisse
	Erzeugung von Lesezeichen mit einer Vielzahl von Optionen und Steuermöglichkeiten
	Seitenübergänge für die Vollbildanzeige, z.B. Verwischen oder Schachbrettmuster
	Erzeugung aller PDF-Anmerkungstypen wie PDF-Verknüpfungen, Links auf andere Dokumenttypen und Weblinks
	Benannte Ziele für Verknüpfungen, Lesezeichen und Datei-Öffnen-Aktion
	Erzeugung symbolischer Namen für die Seiten (page labels)
Multimedia	Einbettung von 3D-Animationen in PDF
	Einbettung von Flash-, Audio-, Movie- und 3D-Inhalten in PDF und Steuerung mit JavaScript[1]
	Flash-basierte Navigatoren für die individuelle Präsentation von PDF-Portfolios[1]
Georeferenziertes PDF	Erzeugung von PDF mit Geo-Referenzdaten
Metadaten	Dokumentinformation: Standardfelder (Titel, Thema, Verfasser, Stichwörter) und benutzerdefinierte Felder
	Erzeugung von XMP-Metadaten aus konventionellen Dokument-Infofeldern oder aus benutzerdefinierten XMP-Daten
	Verarbeitung von XMP-Metadaten in Rasterbildern (TIFF, JPEG, JPEG 2000) und SVG-Grafiken[1]
Programmierung	Sprachbindungen für Cobol, COM, C, C++, Java, .NET, Objective-C, Perl, PHP, Python, REALbasic, RPG, Ruby
	Virtuelles Dateisystem zur Datenübergabe im Speicher, zum Beispiel für Bilder aus einer Datenbank
	Erzeugung von PDF-Dokumenten auf Datenträger oder direkt im Arbeitsspeicher (für Webserver)

1. Neu oder erheblich verbessert in PDFlib 9.0

1.5 Zusätzliche Funktionalität von PDFlib+PDI

Tabelle 1.2 listet die Funktionen von PDFlib+PDI und PPS in Ergänzung zu den Grundfunktionen der PDF-Erzeugung aus Tabelle 1.1 auf. Neue oder erweiterte Funktionen sind gekennzeichnet.

Tabelle 1.2 Zusatzfunktionen von PDFlib+PDI

Kategorie	Funktionalität
PDF input (PDI)	*Import von Seiten aus vorhandenen PDF-Dokumenten*
	Import aller PDF-Versionen bis PDF 1.7 Extension Level 8 (Acrobat X/XI)[1] und PDF 2.0
	Import von Dokumenten, die mit einem PDF-Standardalgorithmus verschlüsselt sind[1]
	Abfrage von Informationen aus importierten Seiten
	Klonen der Seitengeometrie importierter Seiten (z.B. BleedBox, TrimBox, CropBox)
	Löschen redundanter Objekte (z.B. identische Fonts) über mehrere importierte PDF-Dokumente[1]
	Reparatur beschädigter importierter PDF-Dokumente[1]
	Kopieren von PDF/A- oder PDF/X-Druckausgabe-Bedingungen aus importierten PDF-Dokumenten
	Import von Tagged PDF einschließlich der Strukturhierarchie[1]
	Import von Ebenen-Definitionen (optionaler Inhalt)[1]
pCOS-Schnittstelle	*pCOS-Schnittstelle zur Abfrage von Detailinformationen über importierte PDF-Dokumente*

1. Neu oder erheblich verbessert in PDFlib+PDI 9.0

1.6 Zusätzliche Funktionalität von PPS

Tabelle 1.3 listet die Funktionen auf, die ausschließlich von PDFlib Personalization Server (PPS) unterstützt werden (in Ergänzung zu den Grundfunktionen der PDF-Erzeugung aus Tabelle 1.1 und den PDF-Importfunktionen aus Tabelle 1.2). Neue oder erweiterte Funktionen sind gekennzeichnet.

Tabelle 1.3 Zusatzfunktionalität von PDFlib Personalization Server (PPS)

Kategorie	***Funktionalität***
Variable Document Printing (VDP)	*PDF-Personalisierung mit PDFlib-Blöcken für Text, Rasterbilder, PDF oder SVG-Vektorgrafik*[1]
	Erstellung von PDFlib-Blöcken mit PPS per Programm[1]
	Kopieren von PDFlib-Blöcken aus importierten Dokumenten[1]
PDFlib Block-Plugin	*PDFlib Block-Plugin zur interaktiven Erstellung von PDFlib-Blöcken in Adobe Acrobat auf Windows und OS X*
	PPS-Blockvorschau in Acrobat
	Kopieren von Blöcken in die Vorschaudatei[1]
	Am Raster ausrichten zur Erzeugung oder Bearbeitung von Blöcken in Acrobat
	Klonen der PDF/X- oder PDF/A-Properties eines Block-Containers[1]
	Konvertierung von PDF-Formularfeldern in PDFlib-Blöcke zur automatischen Befüllung
	Textflow-Blöcke können so verknüpft werden, dass ein Block den übrigen Text des Vorgängerblocks aufnimmt.
	Liste der PANTONE®- und HKS®-Schmuckfarbnamen in das Block-Plugin integriert[1]

1. Neu oder erheblich verbessert in PDFlib Personalization Server 9.0

1.7 Verfügbarkeit der Funktionen in den Produkten

Tabelle 1.4 zeigt die Verfügbarkeit von Funktionen in den unterschiedlichen Produkten der PDFlib-Familie.

Tabelle 1.4 Verfügbarkeit von Funktionen in verschiedenen Produkten

Funktion	***API-Funktionen und Optionen***	***PDFlib***	***PDFlib+PDI***	***PPS***
elementare PDF-Erstellung	*alle außer den unten angeführten*	*X*	*X*	*X*
linearisiertes (web-optimiertes) PDF	*PDF_end_document( ) mit Option* `linearize`	*X*[1]	*X*	*X*
PDF-Optimierung (nur relevant für ineffizienten Clientcode und nicht optimierte, importierte PDF-Dokumente)	*PDF_begin_document( ) mit Option* `optimize`	*X*[1]	*X*	*X*
Referenziertes PDF, PDF/X-5g und PDF/X-5pg	*PDF_begin_template_ext( ), PDF_open_pdi_page( ), und PDF_load_graphics( ) mit Option* `reference`	*X*[1]	*X*	*X*
Parsen von PDF-Dokumenten für Portfolio-Erstellung	*PDF_add_portfolio_file( ) mit Option* `password`	*X*[1]	*X*	*X*
PDF-Import (PDI)	*alle PDI-Funktionen*	–	*X*	*X*
Abfrage von Informationen aus PDFs mit pCOS	*alle pCOS-Funktionen*	–	*X*	*X*
Blockbefüllung mit variablen Daten	*PDF_fill_*block( )*	–	–	*X*
Block-Erstellung per Programm	*PDF_poca_new( ) mit Option* `usage=blocks` *PDF_begin/end_page_ext( ) mit Option* `blocks`	–	–	*X*
Kopieren von Blöcken in bereits erzeugte Ausgaben	*PDF_process_pdi( ) mit Option* `action=copyblock` *oder* `action=copyallblocks`	–	–	*X*
interaktive Erstellung von PDFlib-Blöcken zur Verwendung mit PPS	*PDFlib Block-Plugin für Acrobat*	–	–	*X*

1. In PDFlib-Quellcode-Paketen nicht verfügbar, da für diese Funktion PDI intern erforderlich ist.

2 Sprachbindungen von PDFlib

Hinweis *Sie sollten sich unbedingt die Starter-Beispiele ansehen, die in allen PDFlib-Paketen enthalten sind. Diese befassen sich mit vielen wichtigen Aspekten der PDFlib-Programmierung und bieten damit eine gute Basis zur Entwicklung eigener Anwendungen.*

2.1 C-Sprachbindung

PDFlib ist in C geschrieben, mit einigen C++-Modulen. Um die C-Sprachbindung von PDFlib zu nutzen, können Sie eine statische oder eine dynamisch ladbare Bibliothek (DLL unter Windows und MVS) verwenden. Außerdem benötigen Sie die zentrale PDFlib-Include-Datei *pdflib.h* zur Einbindung in die Quellmodule Ihrer PDFlib-Anwendung. Alternativ dazu kann *pdflibdl.h* eingesetzt werden, um die PDFlib-DLL dynamisch zur Laufzeit zu laden (siehe Abschnitt »Einsatz von PDFlib als DLL, die zur Laufzeit geladen wird«, Seite 29).

Hinweis *Anwendungen, die die C-Sprachbindung von PDFlib verwenden, müssen mit einem C++-Linker gebunden werden, da PDFlib einige in C++ implementierten Teile enthält. Die Verwendung eines C-Linkers kann zu ungelösten externen Verweisen führen, sofern die Anwendung nicht explizit mit C++-Bibliotheken gebunden wird.*

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.1 aufgeführten Datentypen übergeben werden.

Tabelle 2.1 Datentypen der C-Sprachbindung

API-Datentyp	*Datentypen der C-Sprachbindung*
Strings	`const char *` *(NULL-Stringwerte von C und leere Strings werden gleich behandelt.)*
Binärdaten	`const char *`

Fehlerbehandlung in C. PDFlib unterstützt die strukturierte Ausnahmebehandlung mit *try/catch*-Klauseln. Damit können C- und C++-Clients von PDFlib ausgelöste Exceptions abfangen und angemessen reagieren. In der *catch*-Klausel hat der Client Zugriff auf einen String mit einer exakten Problembeschreibung, einer eindeutigen Exception-Nummer und dem Namen der PDFlib-API-Funktion, die die Ausnahme ausgelöst hat. Ein PDFlib-C-Clientprogramm mit Ausnahmebehandlung besitzt in etwa folgenden Aufbau:

```
PDF_TRY(p)
{
        ...PDFlib-Anweisungen...
}
PDF_CATCH(p)
{
        printf("PDFlib-Exception im Beispiel Hello:\n");
        printf("[%d] %s: %s\n",
            PDF_get_errnum(p), PDF_get_apiname(p), PDF_get_errmsg(p));
        PDF_delete(p);
        return(2);
}
```

```
PDF_delete(p);
```

PDF_TRY/PDF_CATCH sind recht trickreich als Präprozessor-Makros implementiert. Vergessen Sie eines davon, so erhalten Sie eine vermutlich schwierig zu verstehende Compiler-Fehlermeldung. Benutzen Sie die Makros deshalb genau wie oben angegeben, ohne zusätzlichen Code zwischen die *TRY*- und *CATCH*-Klauseln einzufügen (außer *PDF_CATCH( )*).

Eine wesentliche Aufgabe der *catch*-Klausel besteht darin, den internen Speicher von PDFlib mit *PDF_delete( )* und dem Zeiger auf das PDFlib-Objekt zu bereinigen. *PDF_delete( )* schließt gegebenenfalls auch die Ausgabedatei. Nach einer Exception ist das PDF-Dokument unbrauchbar und wird unvollständig und inkonsistent hinterlassen. Wie auf eine Exception zu reagieren ist, hängt natürlich von der jeweiligen Anwendung ab.

Bei C- und C++-Clients, die keine Exceptions abfangen, wird auf Exceptions standardmäßig mit einer entsprechenden Meldung auf die Standard-Fehlerausgabe sowie einem Abbruch reagiert. Beachten Sie, dass die PDF-Ausgabedatei dabei in einem inkonsistenten Zustand hinterlassen wird! Da ein Programmabbruch für eine Bibliotheksroutine nicht akzeptabel ist, sollten bei ernsthaften PDFlib-Projekten unbedingt die Fehlerbehandlungsmöglichkeiten von PDFlib genutzt werden. Eine benutzerdefinierte *catch*-Klausel könnte die Fehlermeldung beispielsweise in einem GUI-Dialogfeld präsentieren und danach nicht abbrechen, sondern auf andere Art fortfahren.

Volatile-Variablen. Besondere Aufmerksamkeit ist beim Umgang mit Variablen erforderlich, die sowohl im *PDF_TRY( )*-Block als auch im *PDF_CATCH( )*-Block verwendet werden. Da der Compiler nichts vom Sprung zwischen den Blöcken weiß, erzeugt er in einer solchen Situation unter Umständen fehleranfälligen Code (z.B. durch Optimierung des Variablenzugriffs über Register). Dieses Problem lässt sich jedoch mit einer einfachen Regel vermeiden:

Hinweis *Variablen, die sowohl im PDF_TRY( )-Block als auch im PDF_CATCH( )-Block verwendet werden, müssen als »volatile« deklariert werden.*

Das Schlüsselwort *volatile* signalisiert dem Compiler, dass er die Variable keiner (eventuell riskanten) Optimierung unterziehen darf.

Verschachtelte try/catch-Blöcke und wiederholtes Auslösen von Exceptions. *PDF_TRY( )*-Blöcke lassen sich beliebig verschachteln. Im Falle einer verschachtelten Fehlerbehandlung kann der innere *catch*-Block den äußeren *catch*-Block durch erneutes Auslösen der Exception aktivieren:

```
PDF_TRY(p)                          /* äußerer try-Block */
{
        /* ... */

        PDF_TRY(p)                  /* innerer try-Block */
        {
            /* ... */
        }
        PDF_CATCH(p)                /* innerer catch-Block */
        {
            /* Fehlerbereinigung */
            PDF_RETHROW(p);
```

```
        }
        /* ... */
}
PDF_CATCH(p)                        /* äußerer catch-Block */
{
        /* weitere Fehlerbereinigung */
        PDF_delete(p);
}
```

Der Aufruf von *PDF_RETHROW( )* in der inneren Fehlerbehandlung übergibt die Programmausführung unmittelbar an die erste Anweisung des äußeren *PDF_CATCH( )*-Blocks.

Vorzeitiges Verlassen eines try-Blocks. Wird ein *PDF_TRY( )*-Block verlassen – zum Beispiel mit einer return-Anweisung –, ohne dass das entsprechende *PDF_CATCH( )*-Makro zur Ausführung kommt, dann muss die Exception-Maschinerie mit dem Makro *PDF_EXIT_TRY( )* darüber informiert werden. Zwischen diesem Makro und dem Ende des *try*-Blocks darf keine andere Bibliotheksfunktion aufgerufen werden:

```
PDF_TRY(p)
{
        /* ... */

        if (error_condition)
        {
            PDF_EXIT_TRY(p);
            return -1;
        }
}
PDF_CATCH(p)
{
        /* Fehlerbereinigung */
        PDF_RETHROW(p);
}
```

Einsatz von PDFlib als DLL, die zur Laufzeit geladen wird. Die meisten Clients werden PDFlib als statisch gebundene oder dynamische Bibliothek einsetzen, die beim Linken gebunden wird. Sie können die PDFlib-DLL aber auch zur Laufzeit laden und sich dynamisch Zeiger auf alle API-Funktionen besorgen. Dies ist insbesondere unter MVS sinnvoll, wo es üblich ist, die Bibliothek als DLL zur Laufzeit zu laden, ohne die Applikation überhaupt mit der PDFlib-Bibliothek zu linken. Zur einfacheren Verwendung dieser Methode gehen Sie wie folgt vor:

- Inkludieren Sie *pdflibdl.h* statt *pdflib.h*.
- Verwenden Sie *PDF_new_dl( )* und *PDF_delete_dl( )* statt *PDF_new( )* und *PDF_delete( )*.
- Verwenden Sie *PDF_TRY_DL( )* und *PDF_CATCH_DL( )* statt *PDF_TRY( )* und *PDF_CATCH( )*.
- Arbeiten Sie bei allen anderen PDFlib-Aufrufen mit Funktionszeigern.
- *PDF_get_opaque( )* darf nicht verwendet werden.
- Kompilieren Sie das Hilfsmodul *pdflibdl.c* und binden Sie es mit Ihrer Anwendung.

Hinweis Das Laden der PDFlib-DLL zur Laufzeit wird nicht auf allen Plattformen unterstützt.

2.2 C++-Sprachbindung

Hinweis *Für in C++ geschriebene .NET-Anwendungen empfehlen wir, auf die .NET-DLL von PDFlib direkt zuzugreifen, anstatt über die C++-Sprachbindung (plattformübergreifende Anwendungen sollten allerdings die C++-Sprachbindung verwenden). Das PDFlib-Paket enthält C++-Beispielcode für .NET CLI (Common Language Interface), der diese Kombination veranschaulicht.*

Neben der C-Include-Datei *pdflib.h* wird für PDFlib-Clients ein objektorientierter Wrapper für C++ mitgeliefert. Dieser erfordert die Include-Datei *pdflib.hpp*, die wiederum *pdflib.h* inkludiert. Da die Implementierung von *pdflib.hpp* auf Templates basiert, wird kein entsprechendes *.cpp*-Modul benötigt. Der C++-Wrapper ersetzt das Präfix *PDF_* in allen PDFlib-Funktionen durch einen objektorientierten Ansatz mit einem PDFlib-Objekt und zugehörigen Methoden.

DatentypenParameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.2 aufgeführten Datentypen übergeben werden.

Tabelle 2.2 Datentypen der C++-Sprachbindung

API-Datentyp	*Datentypen der C++-Sprachbindung*
Strings	`std::wstring` *als Vorgabewert, der aber angepasst werden kann (siehe unten)*
Binärdaten	`const char *`

String-Behandlung in C++. Mit PDFlib 8 wurde eine neue Unicode-fähige C++-Sprachbindung eingeführt. Mit dem neuen Template-basierten Ansatz wird die String-Behandlung folgendermaßen unterstützt:

- Strings vom Typ *std::wstring* der C++-Standard-Bibliothek werden als grundlegender String-Typ verwendet. Sie können UTF-16- oder UTF-32-kodierte Unicode-Zeichen enthalten. Dies ist das voreingestellte Verhalten ab PDFlib 8 und die empfohlene Vorgehensweise für neue Anwendungen, es sei denn, benutzerdefinierte Datentypen bieten einen erheblichen Vorteil gegenüber *wstrings* (siehe nächster Punkt).
- Benutzerdefinierte Datentypen können für die String-Behandlung verwendet werden, sofern der benutzerdefinierte Datentyp eine Instanziierung des Klassen-Templates *basic_string* ist und mit vom Client übergebenen Konvertierungsmethoden nach und aus Unicode konvertiert werden kann. Als Beispiel ist eine benutzerdefinierte String-Implementierung für UTF-8-Strings im PDFlib-Paket enthalten.
- Aus Gründen der Kompatibilität mit bestehenden C++-Anwendungen, die mit PDFlib 7 oder früheren Versionen entwickelt wurden. können normale C++-Strings verwendet werden (Datentyp *string*). Diese Variante wird nur zur Kompatibilität mit bestehenden Anwendungen empfohlen, nicht aber für neue Projekte (siehe Abschnitt »Vollständige Quellcode-Kompatibilität für ältere Anwendungen«, Seite 31).

Die neue Schnittstelle setzt voraus, dass alle an PDFlib-Methoden übergebenen und von PDFlib-Methoden erhaltenen Strings native *wstrings* sind. Abhängig von der Größe des Datentyps *wchar_t* sollten *wstrings* Unicode-Strings enthalten, die als UTF-16 (2-Byte-Zeichen) oder UTF-32 (4-Byte-Zeichen) kodiert sind. Literalen Strings im Quellcode muss ein *L* vorangestellt werden, um sie als Wide Strings zu kennzeichnen. Unicode-Zeichen in Literalen können mit der Syntax *\u* und *\U* erstellt werden. Obwohl diese Syntax Teil der ISO-Norm von C++ ist, wird sie von einigen Compilern nicht unterstützt. In diesem Fall müssen literale Unicode-Zeichen mit Hex-Zeichen erstellt werden.

Hinweis *Auf EBCDIC-basierten Systemen erfordert das Formatieren der Optionslisten-Strings für die auf wstring basierende Schnittstelle eine zusätzliche Konvertierung, um eine Mischung aus EBCDIC- und UTF-16-wstrings in Optionslisten zu vermeiden. Beispiel-Code sowie Anweisungen für diese Konvertierung finden Sie im Hilfsmodul* `utf16num_ebcdic.hpp`.

Anpassen von Anwendungen an die neue C++-Sprachbindung. Bestehende C++-Anwendungen, die mit PDFlib 7 oder früheren Versionen entwickelt wurden, können folgendermaßen angepasst werden:

- Da die C++-Klasse von PDFlib nun im Namensraum *pdflib* steht, muss der Name der Klasse qualifiziert werden. Um das Konstrukt *pdflib::PDFlib* zu vermeiden, sollte bei Client-Anwendungen die folgende Anweisung hinzugefügt werden, bevor PDFlib-Methoden verwendet werden:

```
using namespace pdflib;
```

- Stellen Sie die String-Behandlung der Anwendung auf *wstrings* um. Dies betrifft hauptsächlich Daten aus externen Quellen. Allerdings müssen String-Literale im Quellcode einschließlich Optionslisten auch durch das Voranstellen des Präfix *L* angepasst werden, z.B.:

```
const wstring imagefile = L"nesrin.jpg";
image = p.load_image(L"auto", imagefile, L"");
```

- Geeignete *wstring*-fähige Methoden *(wcerr* usw.) müssen verwendet werden, um PDFlib-Fehlermeldungen und Exception-Strings (Methode *get_errmsg( )* in den Klassen *PDFlib* und *PDFlibException*) zu verwenden.
- Entfernen Sie Aufrufe von PDFlib-Methoden, die nur für nicht Unicode-fähige Sprachen erforderlich sind, besonders den Folgenden:

```
p.set_parameter("hypertextencoding", "host");
```

- Das Modul *pdflib.cpp* ist für die C++-Sprachbindung nicht mehr erforderlich. Obwohl das PDFlib-Paket eine Dummy-Implementierung dieses Moduls enthält, sollten Sie es aus dem Build-Prozess für PDFlib-Anwendungen entfernen.

Vollständige Quellcode-Kompatibilität für ältere Anwendungen. Die neue C++-Sprachbindung wurde für Quellcode-Kompatibilität auf Anwendungsebene konzipiert, Client-Anwendungen müssen jedoch neu kompiliert werden. Um volle Quellcode-Kompatibilität für ältere Anwendungen zu erreichen, die mit PDFlib bis Version 7 entwickelt wurden, stehen folgende Hilfsmittel zur Verfügung:

- Mit der folgenden Anweisung (vor dem Inkludieren von *pdflib.hpp*) können Sie die *wstring*-basierte Schnittstelle deaktivieren:

```
#define PDFCPP_PDFLIB_WSTRING 0
```

- Mit der folgenden Anweisung (vor dem Inkludieren von *pdflib.hpp*) können Sie den Namensraum *PDFlib* deaktivieren:

```
#define PDFCPP_USE_PDFLIB_NAMESPACE 0
```

Fehlerbehandlung in C++. PDFlib-API-Funktionen lösen im Fehlerfall eine C++-Exception aus. Diese Exceptions müssen im Client-Code mit den üblichen *try/catch*-Klauseln von C++ abgefangen werden. Für ausführlichere Fehlerinformationen stellt die Klasse PDFlib die öffentliche (*public*) Klasse *PDFlib::Exception* mit Methoden zur Verfügung, die

die genaue Fehlermeldung, die Exception-Nummer sowie den Namen der API-Funktion liefern, die die Exception ausgelöst hat.

Native C++-Exceptions, die durch PDFlib-Routinen ausgelöst wurden, verhalten sich wie erwartet. Das folgende Codefragment fängt Exceptions ab, die von PDFlib ausgelöst werden:

```
try {
        ...PDFlib-Anweisungen...
catch (PDFlib::Exception &ex) {
        wcerr << L"PDFlib-Exception im Beispiel Hello: " << endl
              << L"[" << ex.get_errnum() << L"] " << ex.get_apiname()
              << L": " << ex.get_errmsg() << endl;
}
```

Speicherverwaltung in C++. Vom Client übergebene Routinen zur Speicherverwaltung funktionieren in der C++-Sprachbindung genauso wie in der C-Sprachbindung.

Dem PDFlib-Konstruktor können optional Speicherverwaltungsroutinen und ein Zeiger auf Benutzerdaten übergeben werden. In *pdflib.hpp* werden standardmäßig NULL-Argumente übergeben, die eine Aktivierung der Speicherverwaltungsroutinen von PDFlib bewirken. Sämtliche Speicherverwaltungsfunktionen müssen C-Funktionen sein. Es dürfen keine C++-Methoden zum Einsatz kommen.

Einsatz von PDFlib als DLL, die zur Laufzeit geladen wird. Ähnlich wie bei der C-Sprachbindung kann PDFlib mit der C++-Sprachbindung dynamisch zur Laufzeit an Ihre Anwendung gebunden werden (siehe Abschnitt »Einsatz von PDFlib als DLL, die zur Laufzeit geladen wird«, Seite 29). Das dynamische Laden beim Kompilieren des Anwendungsmoduls, das *pdflib.hpp* enthält, können Sie folgendermaßen aktivieren;

```
#define PDFCPP_DL    1
```

Kompilieren Sie zusätzlich das Hilfsmodul *pdflibdl.c* und binden Sie die entsprechende Objektdatei mit Ihrer Anwendung. Da die Details des dynamischen Ladens im PDFlib-Objekt versteckt sind, ist das C++-API davon nicht betroffen: alle Methodenaufrufe sehen gleich aus, unabhängig davon, ob dynamisches Laden aktiviert ist.

Hinweis *Das Laden der DLL zur Laufzeit wird nicht auf allen Plattformen unterstützt.*

2.3 COM-Sprachbindung

COM (Component Object Model)[1] ist ein sprachunabhängiger Standard zur Kommunikation von Softwarekomponenten. Die COM-Edition von PDFlib ist als DLL auf Basis des PDFlib-Kerns ausgelegt. Die Wrapper-DLL ruft die PDFlib-Kernfunktionen auf und ist verantwortlich für die Kommunikation mit den zugrunde liegenden COM-Mechanismen, die Registrierung und die Type-Library sowie die Behandlung von COM-Exceptions. Die technischen Daten des COM-Wrappers von PDFlib lauten wie folgt (zerbrechen Sie sich nicht den Kopf, wenn Ihnen nicht alle Aspekte geläufig sind – diesbezügliche Kenntnisse sind für den Einsatz von PDFlib nicht erforderlich):

- PDFlib fungiert als Win32 *in-process* COM-Serverkomponente (auch *Automation Server* genannt) ohne Benutzeroberfläche.
- PDFlib ist eine *both-threaded* Komponente, das heißt, sie wird sowohl als *apartment-threaded* als auch als *free-threaded* Komponente behandelt. Außerdem nutzt PDFlib einen *free-threaded marshaller*. Clients können das PDFlib-Objekt also direkt verwenden (statt eine *Proxy/Stub*-Kombination nutzen), was eine erhebliche Performance-Steigerung zur Folge hat.
- Die PDFlib-Binärdatei *pdflib_com.dll* ist eine selbstregistrierende DLL mit einer Type-Library.
- PDFlib ist zustandslos, das heißt statt Properties werden Methoden-Parameter verwendet.
- Die duale Schnittstelle von PDFlib unterstützt sowohl frühe als auch späte Bindung.
- PDFlib unterstützt ausführliche Fehlerinformationen.

Installation der COM-Edition von PDFlib. PDFlib kann in allen Umgebungen eingesetzt werden, die COM-Komponenten unterstützen. Wir zeigen unsere Beispiele in den folgenden Umgebungen:

- Visual Basic
- Active Server Pages (ASP) mit JScript
- Windows Script Host (WSH) mit VBScript

Active Server Pages und Windows Script Host unterstützen JScript und VBScript. PDFlib funktioniert auch in Visual Basic for Applications (VBA) und vielen anderen Entwicklungsumgebungen für COM.

Die Installation von PDFlib ist einfach zu bewerkstelligen. Beachten Sie dabei Folgendes:

- Wenn Sie PDFlib auf einer NTFS-Partition installieren, brauchen alle PDFlib-Benutzer die Lesen-Berechtigung auf das Installationsverzeichnis und die Ausführen-Berechtigung für *...\PDFlib 9.0. 1\bin\pdflib_com.dll*.
- Der installierende Benutzer benötigt die Schreiben-Berechtigung für die Windows-Registry. Administrator-Berechtigungen oder Berechtigungen der Gruppe »Hauptbenutzer« reichen in der Regel aus.

Die MSI-Installationsroutinen von PDFlib. PDFlib ist als MSI-Paket verfügbar (Microsoft Windows Installer), das Installations-, Reparatur- und Deinstallationsfunktionen bietet. Um PDFlib mit dem MSI-Paket zu installieren, doppelklicken Sie einfach auf die *.msi*-Datei oder klicken sie mit der rechten Maustaste an und wählen *Installieren*.

1. Weitere Informationen zu COM finden Sie unter www.microsoft.com/com.

Die Installationsroutinen, die mit der COM-Komponente von PDFlib ausgeliefert werden, regeln automatisch alle Aspekte, die für den PDFlib-Einsatz mit COM relevant sind. Der Vollständigkeit halber beschreiben wir trotzdem im Folgenden, welche Laufzeitumgebung für den PDFlib-Einsatz erforderlich ist (die von der Installationsroutine automatisch so eingestellt wird):

- Die PDFlib COM-DLL *pdflib_com.dll* wird ins Installationsverzeichnis kopiert.
- Die PDFlib COM-DLL wird in der Windows-Registry eingetragen. Um eine Registrierung zu erreichen, verwendet die Installationsroutine die selbstregistrierende PDFlib-DLL.
- Wird eine lizenzierte Version von PDFlib installiert, dann wird die Seriennummer im System eingetragen.

Silent-Installation. Wird PDFlib als Bestandteil eines anderen Software-Pakets ausgeliefert oder auf sehr vielen Rechnern eingesetzt, die mit einem Tool wie SMS verwaltet werden, ist es mühsam, PDFlib auf jedem Rechner einzeln manuell zu installieren. Für solche Situationen gibt es die Möglichkeit, PDFlib automatisch ohne jede Benutzerinteraktion zu installieren.

Die MSI-Installationsroutine unterstützt die Silent-Installation. Mit dem folgenden Befehl in der Befehlszeile können Sie PDFlib zum Beispiel ohne Benutzereingaben installieren:

```
PDFlib-x.y.z-MSWin32-COM.msi
```

Eine vollständige Liste aller Befehlszeilenparameter finden Sie in der Dokumentation zum Microsoft Windows Installer.

Einsatz der PDFlib-COM-Komponente auf dem Server eines ISP. Die Installation von Software auf dem Server eines Internet Service Provider (ISP) ist in der Regel wesentlich schwieriger als auf einem lokalen Rechner, da ISPs oft sehr widerwillig reagieren, wenn Kunden neue Software installieren möchten. PDFlib ist sehr ISP-freundlich, da sie weder Dateien im Windows-Verzeichnis noch zwingend Registrierungseinträge benötigt:

- Es ist nur eine einzige DLL nötig, die sich in einem beliebigen Verzeichnis befinden kann. Diese DLL muss mit *regsvr32* registriert werden.
- Standardmäßig werden nur ein paar wenige private Einträge in der Registrierung vorgenommen, die sich unter *HKEY_LOCAL_MACHINE\SOFTWARE\PDFlib* befinden. Diese Einträge können auch manuell vorgenommen werden.
- PDFlib kann sogar ohne jegliche private Einträge in der Registrierung verwendet werden. Der Benutzer muss die fehlenden Einträge dann durch geeignete Aufrufe der Funktion *set_option( )* kompensieren und damit die Optionen *SearchPath*, *resourcefile* und *license* setzen. Zur PDFlib-Installation benötigt die COM-Implementierung selbst aber einige Registrierungseinträge.

PDFlib-COM-Komponente als Bestandteil des eigenen Produkts. Wenn Sie als Entwickler eine Runtime-Lizenz für PDFlib erworben haben und die PDFlib-COM-Komponente als Bestandteil Ihres eigenen Produkts ausliefern wollen, müssen Sie die PDFlib-Installation entweder komplett ausliefern und die PDFlib-Installationsroutine als Teil des Produkt-Setups ablaufen lassen oder aber folgende Schritte durchführen:

- Integrieren Sie die PDFlib-Dateien in eine eigene Installation. Die von PDFlib benötigten Dateien lassen sich durch einen Blick in das Installationsverzeichnis ermitteln, da nur dort Dateien installiert werden.

- Vergessen Sie keinen der erforderlichen PDFlib-Registrierungsschlüssel. Dazu können Sie die Einträge in der mitgelieferten Registrierungsdatei-Vorlage *pdflib.reg* ergänzen und diese bei der Installation Ihres eigenen Produkts verwenden.
- Zur Selbstregistrierung muss *pdflib_com.dll* (zum Beispiel über *regsvr32)* aufgerufen werden.
- Übergeben Sie Ihren Lizenzschlüssel zur Laufzeit mit der PDFlib-Funktion *set_option( )*, siehe auch Kapitel 0, »Anwendung des PDFlib-Lizenzschlüssels«, Seite 11:

```
oPDF.set_option("license", "...Ihr Lizenzschlüssel...")
```

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.3 aufgeführten Datentypen übergeben werden.

Tabelle 2.3 Datentypen der COM-Sprachbindung

API-Datentyp	*Datentypen der COM-Sprachbindung*
Strings	`BSTR` *Delphi mit COM:* `String` *(für 8-Bit-Encodings) oder* `WideString` *(für Unicode)*
Binärdaten	`variant` *vom Typ* `VT_ARRAY	VT_UI1` *(Varianten-Array vorzeichenloser Bytes)*

Fehlerbehandlung in COM. Die Fehlerbehandlung für die PDFlib-COM-Komponente erfolgt entsprechend der COM-Konventionen: Tritt eine PDFlib-Exception auf, so wird eine COM-Exception ausgelöst, die den PDFlib-Fehlercode und eine Klartextbeschreibung des Fehlers enthält. Außerdem wird der vom PDFlib-Objekt belegte Speicher freigegeben.

Die COM-Exception kann im PDFlib-Client abgefangen und auf die Art bearbeitet werden, die die Client-Umgebung für COM-Fehler vorsieht. Eine ausführliche Beschreibung der entsprechenden Mechanismen in PDFlib finden Sie in Abschnitt 3.1.1, »Behandlung von Ausnahmen (Exceptions)«, Seite 61.

Einsatz von PDFlib mit Active Server Pages. Beim Einsatz externer Dateien (zum Beispiel Rasterbilddateien) muss die Funktion *MapPath* von ASP verwendet werden, um die Namen von Pfaden auf der lokalen Festplatte auf Pfade abzubilden, die in ASP-Skripten nutzbar sind. Wenn Sie mit *MapPath* nicht vertraut sind, sollten Sie sich die mit PDFlib ausgelieferten ASP-Beispiele ansehen oder die ASP-Dokumentation zu Rate ziehen. Außerdem sollten Sie in ASP-Skripten keine absoluten Pfadnamen verwenden, da diese ohne MapPath unter Umständen nicht funktionieren.

Hinweis UNC-Pfadnamen funktionieren beim Einsatz von PDFlib in IIS nicht.

Das Verzeichnis mit Ihren ASP-Skripten muss die Ausführen-Berechtigung besitzen und außerdem die Schreiben-Berechtigung, falls die PDF-Ausgabe nicht im Arbeitsspeicher *(in-core)* erzeugt wird (die mitgelieferten ASP-Beispiele generieren PDF alle im Arbeitsspeicher).

Sie können die Ausführung von COM-Objekten wie *PDFlib_com* auf Active Server Pages beschleunigen, indem Sie das Objekt außerhalb des eigentlichen Skriptcodes auf der ASP-Seite instanziieren, womit der Geltungsbereich des Objekts von der Seite auf die Session ausgeweitet wird. Genauer gesagt, verwenden Sie zur Erstellung des Objekts *PDFlib_com* dann statt *CreateObject* (wie es im Beispiel im nächsten Abschnitt gezeigt wird):

```
<%@ LANGUAGE = "JavaScript" %>
<%
        var oPDF;
        oPDF = Server.CreateObject("PDFlib_com.PDF");
        oPDF.begin_document("", "");
        ...
```

Zur Erzeugung des Objekts *PDFlib_com* verwenden Sie das Tag *OBJECT* mit den Attributen *RUNAT, ID* und *ProgID*:

```
<OBJECT RUNAT=Server ID=oPDF ProgID="PDFlib_com.PDF"> </OBJECT>

<%@ LANGUAGE = "JavaScript" %>
<%
        oPDF.begin_document("", "");
        ...
```

Sie können die Ausführung weiter beschleunigen, indem Sie das Verfahren auf die Datei *global.asa* anwenden, wobei Sie das Attribut *Scope=Application* benutzen, womit Sie dem Objekt den Geltungsbereich *Application* geben.

Einsatz von PDFlib mit Visual Basic. Bei externen COM-Komponenten unterstützt Visual Basic sowohl frühe Bindung (zur Kompilierzeit) als auch späte Bindung (zur Laufzeit). Bei PDFlib sind zwar beide Bindungsarten möglich, die frühe Bindung ist jedoch eindeutig vorzuziehen. Dazu sind folgende Schritte erforderlich:

- Erstellen Sie über *Project, References...* eine Referenz von Ihrem VB-Projekt auf PDFlib und selektieren Sie das Control *PDFlib_com*.
- Deklarieren Sie Objektvariablen vom Typ *PDFlib_com.PDF* statt vom generischen Typ *Object*:

```
Dim oPDF As PDFlib_com.PDF
Set oPDF = CreateObject("PDFlib_com.PDF")   ' oder: Set oPDF = New PDFlib_com.PDF
```

Die Erstellung einer Referenz und die Verwendung der frühen Bindung bringen mehrere Vorteile mit sich:

- VB kann den Code auf Schreibfehler überprüfen.
- IntelliSense (automatische Befehlsergänzung) und kontextsensitive Hilfe sind verfügbar.
- Der VB-Objektbrowser zeigt alle PDFlib-Methoden mit ihren Parametern und einer Kurzbeschreibung an.
- VB-Programme sind viel schneller mit früher als mit später Bindung.

Bei der PDFlib-Programmierung mit VB ist eine kleine Besonderheit zu beachten: Aufgrund eines von Microsoft bestätigten Fehlers in Visual Basic 6 können einige PDFlib-Funktionen nicht direkt verwendet werden, da Visual Basic die Namen einiger PDFlib-Methoden fälschlicherweise mit integrierten VB-Methoden überschreibt. Die folgende Zeile lässt sich in VB 6 beispielsweise nicht erfolgreich kompilieren:

```
oPDF.circle 10, 10, 30
```

Zur Umgehung dieses Problems enthält das PDFlib-API folgende äquivalente Methoden:

```
pcircle (äquivalent zu circle)
pscale (äquivalent zu scale)
```

Alternativ dazu können Sie folgende Variante verwenden, die der technische Support von Microsoft vorschlug:

```
oPDF.[circle] 10, 10, 30
```

Der Trick dabei sind die eckigen Klammern um den Methodennamen. Von dem Problem sind nur die beiden folgenden PDFlib-Funktionen betroffen:

```
circle
scale
```

In der COM-Komponente von PDFlib ist der Datentyp *integer* eine vorzeichenbehaftete 32-Bit-Zahl. In Visual Basic entspricht dies dem Datentyp *long*. Verlangt die PDFlib-Referenz also ein Argument vom Typ *int*, müssen Visual-Basic-Programmierer den Typ *long* verwenden (obwohl VB auch bei *int*-Argumenten korrekt kompiliert).

Ein Visual-Basic-Programm kann Fehler erkennen und darauf reagieren. Exceptions werden in Visual Basic mit der Klausel *On Error GoTo* abgefangen:

```
Sub main()
        Dim oPDF As PDFlib_com.PDF
        On Error GoTo ErrExit

        ...PDFlib-Anweisungen...

        End
ErrExit:
        MsgBox Hex(Err.Number) & ":  " & Err.Description
End Sub
```

Einsatz der COM-Edition von PDFlib mit .NET. Alternativ zu PDFlib.NET (siehe Abschnitt 2.6, ».NET-Sprachbindung«, Seite 43) kann die COM-Edition von PDFlib mit .NET verwendet werden. Dazu müssen Sie aus der PDFlib-COM-Edition mit dem Hilfsprogramm *tlbimp.exe* eine .NET-Assembly erstellen:

```
tlbimp pdflib_com.dll /namespace:pdflib_com /out:Interop.pdflib_com.dll
```

Diese Assembly verwenden Sie dann in Ihrer .NET-Anwendung. Wenn Sie innerhalb von Visual Studio .NET eine Referenz auf *pdflib_com.dll* hinzufügen, wird automatisch eine Assembly erzeugt.

Das folgende Codefragment zeigt den Einsatz der COM-Edition von PDFlib mit VB.NET:

```
Imports PDFlib_com
    ...
Dim p As PDFlib_com.IPDF
    ...
p = New PDF()
    ...
buf = p.get_buffer()
```

Das folgende Codefragment zeigt den Einsatz der COM-Edition von PDFlib mit C#:

```
using PDFlib_com;
    ...
static PDFlib_com.IPDF p;
    ...
```

```
p = New PDF();
    ...
buf = (byte[])p.get_buffer();
```

Der übrige Code ist der gleiche wie bei der .NET-Edition von PDFlib. Beachten Sie aber, dass Sie in C# das Ergebnis von *get_buffer( )* konvertieren müssen, da der vom COM-Objekt zurückgegebene VARIANT-Datentyp nicht automatisch konvertiert wird (bei *create_pvf( )* ist ebenfalls eine Konvertierung erforderlich).

2.4 Cobol-Sprachbindung

In Cobol werden die PDFlib-API-Funktionen nicht über die Standardnamen für C, sondern über spezielle Kurznamen aufgerufen. Diese Kürzel werden hier nicht näher beschrieben, sondern in einer eigenen Umsetzungstabelle *(xref.txt)* aufgeführt. So ist statt *PDF_load_font( )* beispielsweise die abkürzende Schreibweise *PDLODFNT* zu verwenden.

In Cobol programmierte PDFlib-Clients werden statisch mit dem PDFLBCOB-Objekt gelinkt. Dieses lädt dynamisch das PDLBDLCB Load Module (DLL), welches beim ersten Aufruf von PDNEW (entspricht *PDF_new( ))* wiederum das PDFlib Load Module (DLL) dynamisch lädt. Das Instanz-Handle der neu allozierten internen PDFlib-Struktur wird im Parameter *P* gespeichert, der in allen folgenden Aufrufen übergeben werden muss.

Das PDLBDLCB Load Module liefert die Schnittstellen zwischen den Cobol-Funktionsnamen mit jeweils 8 Zeichen Länge und den PDFlib-Kernroutinen. Außerdem bildet es die asynchrone Ausnahmebehandlung von PDFlib auf das von Cobol erwartete, monolithische Verfahren gemäß »prüfe den Rückgabewert jeder Funktion« ab.

Hinweis PDLBDLCB und PDFLIB müssen dem COBOL-Programm via STEPLIB verfügbar gemacht werden.

Datentypen. Die Datentypen, die in der PDFlib-Referenz benutzt werden, müssen wie in den folgenden Beispielen in Cobol-Datentypen umgesetzt werden:

```
05  PDFLIB-A4-WIDTH         USAGE COMP-1 VALUE 5.95E+2. // float
05  WS-INT                  PIC S9(9) BINARY.           // int
05  WS-FLOAT                COMP-1.                     // float
05  WS-STRING               PIC X(128).                 // const char *
05  P                       PIC S9(9) BINARY.           // long *
05  RETURN-RC               PIC S9(9) BINARY.           // int *
```

Alle an das PDFlib-API übergebene Cobol-Strings sollten mit einem zusätzlichen Byte zur Speicherung des abschließenden Nullbytes (LOW-VALUES (NULL)) definiert werden.

Rückgabewerte. Der Rückgabewert einer PDFlib-API-Funktion wird in einem zusätzlichen Parameter namens *ret* bereitgestellt, der per Referenz übergeben und mit dem Ergebnis des jeweiligen Funktionsaufrufs gefüllt wird. Der Rückgabewert 0 besagt, dass die Funktion erfolgreich ausgeführt wurde; alle anderen Werte weisen auf einen Fehler hin, der zu einem Abbruch der PDF-Generierung führt. Funktionen ohne Rückgabewert (C-Funktionen mit einem Ergebnis vom Typ *void)* verwenden diesen Parameter nicht.

Fehlerbehandlung. In der Cobol-Sprachbindung gibt es keine PDFlib-Ausnahmebehandlung, sondern den zusätzlichen Parameter *rc* (zur Speicherung des *R*eturn *C*ode), der von allen API-Funktionen als Fehlerindikator unterstützt wird. Der Parameter *rc* wird per Referenz übergeben und zur Übermittlung von Problemen verwendet. Jeder Wert ungleich 0 weist auf ein Fehlschlagen des Funktionsaufrufs hin.

2.5 Java-Sprachbindung

Java unterstützt ein portierbares Verfahren zum Anbinden von nativem Programmcode an Java-Programme, nämlich das Java Native Interface (JNI). Das JNI bietet Programmierkonventionen, um native C- oder C++-Routinen aus Java-Code heraus aufzurufen und umgekehrt. Um der Java-VM zugänglich zu sein, müssen alle C-Routinen in geeignetem Wrapper-Code verpackt werden. Die daraus resultierende Bibliothek ist als dynamisches Objekt zu generieren, damit sie von der Java-VM geladen werden kann.

Um von Java aus verwendbar zu sein, wird mit PDFlib JNI-Wrapper-Code mitgeliefert. Anhand des geschilderten Verfahrens kann PDFlib an Java angebunden werden, indem die dynamische Bibliothek von der Java-VM geladen wird. Das eigentliche Laden der Bibliothek erfolgt mittels einer statischen Member-Funktion der Java-Klasse *pdflib*. Dadurch muss sich der Java-Client nicht mit den Einzelheiten zum Laden einer dynamischen Bibliothek auseinandersetzen.

Aufgrund der Stabilität und Robustheit von PDFlib wird die Stabilität und Sicherheit der Java-Anwendung beim Anbinden der nativen PDFlib-Bibliothek an die Java-VM in keinerlei Weise beeinträchtigt. Als Vorteil kann die höhere Geschwindigkeit einer nativen Implementierung genutzt werden.

Installation der Java-Edition von PDFlib. Damit die PDFlib-Anwendung funktioniert, benötigt die Java-VM Zugriff auf den PDFlib-Java-Wrapper und das PDFlib-Java-Paket. PDFlib ist in einem Java-Paket mit dem folgenden Namen enthalten:

```
com.pdflib.pdflib
```

Dieses Paket befindet sich in der Datei *pdflib.jar* und enthält die Klassen *pdflib* und *PDFlibException*. Um dieses Paket Ihrer Anwendung verfügbar zu machen, müssen Sie *pdflib.jar* an die Umgebungsvariable *CLASSPATH* anfügen, die Option *-classpath pdflib.jar* in die Aufrufe von Java-Compiler und Java-Laufzeitumgebung aufnehmen oder die entsprechenden Schritte in Ihrer IDE durchführen. Im JDK können Sie die Java-VM so konfigurieren, dass sie ein vorgegebenes Verzeichnis nach nativen Bibliotheken durchsucht. Dazu weisen Sie der Property *java.library.path* den Namen des gewünschten Verzeichnisses zu, zum Beispiel:

```
java -Djava.library.path=. pdfclock
```

Der Wert dieser Eigenschaft lässt sich wie folgt überprüfen:

```
System.out.println(System.getProperty("java.library.path"));
```

Außerdem sind die folgenden plattformabhängigen Schritte durchzuführen:

- Unter Unix muss die Bibliothek *libpdf_java.so* (unter OS X: *libpdf_java.jnilib)* in eines der Standardverzeichnisse für dynamisch ladbare Bibliotheken oder in ein entsprechend konfiguriertes Verzeichnis kopiert werden.
- Unter Windows muss die Bibliothek *pdf_java.dll* ins Windows-Systemverzeichnis oder in ein Verzeichnis kopiert werden, das in der Umgebungsvariablen *PATH* aufgeführt ist.

Einsatz von PDFlib in J2EE-Applikationsservern und Servlet-Containern. PDFlib eignet sich hervorragend für serverseitige Java-Anwendungen. Das PDFlib-Paket enthält

Beispiel-Code und Konfigurationen für die Verwendung von PDFlib in J2EE-Umgebungen. Beachten Sie dabei die folgenden Konfigurationsaspekte:

- Das Verzeichnis, in dem der Server die nativen Bibliotheken erwartet, ist je nach Anbieter unterschiedlich. Üblich sind Systemverzeichnisse, Verzeichnisse der zugrunde liegenden Java-VM oder lokale Server-Verzeichnisse. Einzelheiten hierzu finden Sie in der Dokumentation, die vom Hersteller der Server bereitgestellt wird.
- Applikationsserver und Servlet-Container verwenden oft einen speziellen Klassenlader, der möglicherweise Einschränkungen unterliegt oder einen bestimmten Klassenpfad verwendet. Bei manchen Servern muss ein besonderer Engine-Klassenpfad festgelegt werden, damit das PDFlib-Paket gefunden werden kann.

Ausführlichere Hinweise zum Einsatz von PDFlib mit verschiedenen Servlet-Engines und Applikationsservern finden Sie in der Dokumentation im Verzeichnis J2EE des PDFlib-Pakets.

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.4 aufgeführten Datentypen übergeben werden.

Tabelle 2.4 Datentypen der Java-Sprachbindung

API-Datentyp	*Datentypen der Java-Sprachbindung*
Strings	`string`
Binärdaten	`byte[]`

Fehlerbehandlung in Java. Die Java-Sprachbindung installiert einen speziellen Error-Handler, der PDFlib-Fehler in native Java-Exceptions übersetzt. Beim Auftreten einer Exception löst PDFlib eine native Java-Exception der folgenden Klasse aus:

```
PDFlibException
```

Die Java-Exceptions können mit der üblichen Kombination aus *try* und *catch* behandelt werden:

```
try {

...PDFlib-Anweisungen...

} catch (PDFlibException e) {
            System.err.print("PDFlib-Exception im Beispiel Hello:\n");
            System.err.print("[" + e.get_errnum() + "] " + e.get_apiname() +
                             ": " + e.get_errmsg() + "\n");

} catch (Exception e) {
        System.err.println(e.getMessage());

} finally {
        if (p != null) {
            p.delete();                     /* PDFlib-Objekt löschen */
        }
}
```

Da PDFlib passende *throws*-Klauseln deklariert, muss der Client-Code alle möglichen PDFlib-Exceptions abfangen oder diese selbst deklarieren.

Konvertierung von Unicode und anderen Encodings. Um PDFlib-Anwendern die Arbeit zu erleichtern, zeigen wir im Folgenden einige nützliche Methoden der Stringkonvertierung. Weitere Informationen hierzu finden Sie in der Java-Dokumentation. Der folgende Konstruktor erzeugt einen Unicode-String aus einem Byte-Array, wobei das Standard-Encoding der Plattform verwendet wird:

```
String(byte[] bytes)
```

Der folgende Konstruktor erzeugt einen Unicode-String aus einem Byte-Array, wobei das im Parameter *enc* übergebene Encoding (z.B. *SJIS, UTF8, UTF-16)* verwendet wird:

```
String(byte[] bytes, String enc)
```

Die folgende Methode der Klasse String konvertiert einen Unicode-String anhand des im Parameter *enc* definierten Encodings in einen String:

```
byte[] getBytes(String enc)
```

Javadoc-Dokumentation für PDFlib. Das PDFlib-Paket enthält Javadoc-Dokumentation für PDFlib. Javadoc enthält nur abgekürzte Beschreibungen aller PDFlib-API-Methoden; Weiterführende Informationen entnehmen Sie bitte der PDFlib-Referenz.

Um Javadoc für PDFlib in Eclipse zu konfigurieren, gehen Sie folgendermaßen vor:

- Klicken Sie im Paket-Explorer mit der rechten Maustaste auf das Java-Projekt und wählen Sie *Javadoc Location*.
- Klicken Sie auf *Browse*... und wählen Sie den Pfad, wo sich Javadoc befindet (Bestandteil des PDFlib-Pakets).

Dann können Sie die Javadoc für PDFlib durchsuchen, z.B. mit der *Java-Browsing*-Perspektive oder über das Hilfe-Menü.

Einsatz von PDFlib mit Groovy. Die Java-Sprachbindung von PDFlib kann auch mit Groovy verwendet werden. Die API-Aufrufe sind identisch mit den Java-Aufrufen; nur die Objekt-Instanziierung ist etwas anders. Ein einfaches Beispiel für die Verwendung von PDFlib mit Groovy ist im PDFlib-Paket enthalten.

2.6 .NET-Sprachbindung

Hinweis *Detaillierte Informationen zu den verschiedenen Ausprägungen und Möglichkeiten für die Verwendung von PDFlib mit dem . NET Framework finden Sie im Dokument PDFlib-in-.NET-How-To.pdf, das im Produkt-Paket enthalten und auch über die PDFlib-Website verfügbar ist.*

Die .NET-Edition von PDFlib unterstützt alle wesentlichen .NET-Konzepte. Technisch gesehen handelt es sich bei der .NET-Edition von PDFlib um eine C++-Klasse (mit einem managed Wrapper um die unmanaged PDFlib-Kernbibliothek), die unter Kontrolle des .NET- Frameworks abläuft. Diese Klasse wird als statische Assembly mit einem starken Namen *(strong name)* ausgeliefert. Die PDFlib-Assembly *(pdflib_dotnet.dll)* enthält die Bibliothek selbst sowie zusätzliche Meta-Informationen.

Installation der .NET-Edition von PDFlib. Installieren Sie PDFlib mit der bereitgestellten MSI-Installationsroutine von Windows. Die MSI-Installationsroutine von PDFlib.NET installiert die PDFlib-Assembly einschließlich der zugehörigen Hilfsdateien, Dokumentation und Beispiele interaktiv auf dem Rechner. Außerdem wird PDFlib registriert, so dass Sie auf der Registerkarte .NET im Dialogfeld *Add Reference* von Visual Studio .NET sofort darauf zugreifen können.

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.5 aufgeführten Datentypen übergeben werden.

Tabelle 2.5 Datentypen der .NET-Sprachbindung

API-Datentyp	*Datentypen der .NET-Sprachbindung*
Strings	`string`
Binärdaten	`byte[ ]`

Fehlerbehandlung in .NET. PDFlib.NET unterstützt .NET-Exceptions und löst eine Exception mit detaillierter Fehlermeldung aus, sobald ein Laufzeitproblem auftritt. Der Client ist für das Abfangen der Exception und eine angemessene Reaktion zuständig. Andernfalls fängt das .NET-Framework die Exception ab, was gewöhnlich zum Abbruch der Anwendung führt.

Um Informationen über die Exception zu übermitteln, definiert PDFlib eine eigene Exception-Klasse namens *PDFlib_dotnet.PDFlibException* mit den Members *get_errnum, get_errmsg* und *get_apiname*. PDFlib implementiert das Interface *IDisposable*, so dass Clients die Methode *Dispose( )* zur Bereinigung aufrufen können.

Einsatz von PDFlib mit C#. Um PDFlib.NET in Ihrem C#-Projekt nutzen zu können, müssen Sie in Visual C# .NET eine Referenz auf die PDFlib.NET-Assembly anlegen. Dazu klicken Sie auf *Project, Add Reference..., Browse...* und wählen *PDFlib_dotnet.dll* aus dem Installationsverzeichnis.

Mit dem Befehlszeilencompiler können Sie PDFlib.NET wie folgt referenzieren:

```
csc.exe /r:..\..\bin\PDFlib_dotnet.dll hello.cs
```

Im Client-Code können von PDFlib ausgelöste .NET-Exceptions mit der üblichen Kombination aus *try* und *catch* behandelt werden:

```
try {
        ...PDFlib-Anweisungen...
catch (PDFlibException e)
{
        // Exception abgefangen, ausgelöst von pdflib
        Console.WriteLine("PDFlib-Exception im Beispiel Hello:\n");
        Console.WriteLine("[{0}] {1}: {2}\n",
            e.get_errnum(), e.get_apiname(), e.get_errmsg());
} finally {
        if (p != null) {
            p.Dispose();
        }
}
```

Einsatz von PDFlib mit VB.NET. Um PDFlib.NET in Ihrem VB.NET-Projekt nutzen zu können, müssen Sie in Visual Basic .NET eine Referenz auf die PDFlib.NET-Assembly erzeugen. Dazu klicken Sie auf *Project, Add Reference..., Browse...* und wählen dann *PDFlib_dotnet.dll* aus dem Installationsverzeichnis.

Mit dem Befehlszeilen-Compiler können Sie PDFlib.NET wie folgt referenzieren:

```
vbc.exe /r:..\..\bin\pdflib_dotnet.dll hello.vb
```

Visual Basic .NET unterstützt zwei Arten der Ausnahmebehandlung:

- Strukturierte Ausnahmebehandlung (die auch in anderen modernen Sprachen wie C# verwendet wird)
- Traditionelle unstrukturierte Ausnahmebehandlung (die einzig mögliche Ausnahmebehandlung in Visual Basic 6.0)

Der Client-Code kann von PDFlib ausgelöste .NET-Exceptions auf beide der oben erwähnten Arten behandeln, die Syntax ist jedoch unterschiedlich. Wir empfehlen, Exceptions mit strukturierter Ausnahmebehandlung abzufangen. Dazu wird eine *try/catch*-Klausel verwendet:

```
Try
        ...PDFlib-Anweisungen...
Catch e As PDFlibException
        Console.WriteLine("PDFlib-Exception in Beispiel Hello:")
        Console.WriteLine("[{0}] {1}: {2}",
            e.get_errnum(), e.get_apiname(), e.get_errmsg())
Finally
        If Not p Is Nothing Then
            p.Dispose()
        End If
End Try
```

Um Exceptions mit traditioneller unstrukturierter Fehlerbehandlung abzufangen, verwenden Sie eine *On Error GoTo*-Klausel:

```
Imports Microsoft.VisualBasic

Public Shared Sub Main()
        On Error GoTo ErrExit
        ...PDFlib-Anweisungen...
        Exit Sub

ErrExit:
```

```
        Console.WriteLine("PDFlib-Exception abgefangen: {0}", Err.Description)
End Sub
```

Einsatz von PDFlib mit C++ und CLI. In C++ geschriebene .NET-Anwendungen (basierend auf der *Common Language Infrastructure* CLI) können ohne die C++-Sprachbindung von PDFlib direkt auf die PDFlib.NET-DLL zugreifen. Dazu muss PDFlib im Quellcode folgendermaßen referenziert werden:

```
using namespace PDFlib_dotnet;
```

Konvertierung von Unicode und anderen Encodings. Um dem PDFlib-Anwender die Arbeit zu erleichtern, zeigen wir eine nützliche Methode zur Konvertierung eines C#-Strings. Weitere Informationen finden Sie in der .NET-Dokumentation. Der folgende Konstruktor erzeugt einen Unicode-String aus einem Byte-Array (mit definiertem Offset und definierter Länge), wobei das im Parameter *Encoding* übergebene Encoding verwendet wird:

```
public String(sbyte*, int, int, Encoding)
```

2.7 Objective-C-Sprachbindung

Obwohl die C- und C++-Sprachbindungen mit Objective-C[1] verwendet werden können, bieten wir auch eine genuine Sprachbindung für Objective-C an. Das PDFlib-Framework ist in den folgenden Ausprägungen erhältlich:

- *PDFlib* für OS X
- *PDFlib_ios* für iOS

Beide Frameworks enthalten Sprachbindungen für C, C++ und Objective-C.

Installation der PDFlib-Edition für Objective-C unter OS X. Um PDFlib in Ihrer Anwendung einsetzen zu können, kopieren Sie *PDFlib.framework* oder *PDFlib_ios.framework* in das Verzeichnis */Library/Frameworks*. Sie können das PDFlib-Framework auch an einem anderen Ort installieren, benötigen dazu aber das *install_name_tool* von Apple, das hier nicht beschrieben wird. Die Header-Datei *PDFlib_objc.h* mit den Methoden-Deklarationen von PDFlib müssen Sie im Quellcode Ihrer Anwendung importieren:

```
#import "PDFlib/PDFlib_objc.h"
```

oder

```
#import "PDFlib_ios/PDFlib_objc.h"
```

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.6 aufgeführten Datentypen übergeben werden.

Tabelle 2.6 Datentypen der Objective-C-Sprachbindung

API-Datentyp	*Datentypen der Objective-C-Sprachbindung*
Strings	`Nsstring` (`nil` *ist gleich einem leeren String)*
Binärdaten	`NSData`

Namenskonventionen für Parameter. Für PDFlib-Methodenaufrufe müssen Sie die Parameter gemäß der folgenden Konventionen übergeben:

- Der Wert des ersten Parameters wird direkt nach dem Methodennamen, durch einen Doppelpunkt getrennt, angegeben.
- Für jeden weiteren Parameter muss der Parametername mit seinem Wert (wiederum jeweils getrennt durch einen Doppelpunkt) angegeben werden. Die Parameternamen finden Sie in der PDFlib-Referenz oder in der Datei *PDFlib_objc.h*.

Die folgende Zeile aus der PDFlib-Referenz:

```
void begin_page_ext(double width, double height, String optlist)
```

entspricht der folgenden Objective-C-Methode:

```
- (void) begin_page_ext: (double) width height: (double) height optlist: (NSString *) optlist;
```

1. Siehe developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html

Ihre Anwendung muss daher ungefähr folgenden Aufruf absetzen:

```
[pdflib begin_page_ext:595.0 height:842.0 optlist:@""];
```

XCode Code Sense kann zur Code-Vervollständigung im PDFlib-Framework verwendet werden.

Fehlerbehandlung in Objective-C. Die Objective-C-Sprachbindung übersetzt PDFlib-Fehler in native Objective-C-Exceptions. Bei einem Laufzeitproblem löst PDFlib eine native Objective-C-Exception der Klasse *PDFlibException* aus. Diese Exceptions können mit der üblichen Kombination aus *try* und *catch* behandelt werden:

```
@try {
        ...PDFlib-Anweisungen...
}
@catch (PDFlibException *ex) {
        NSString * errorMessage =
            [NSString stringWithFormat:@"PDFlib error %d in '%@': %@",
            [ex get_errnum], [ex get_apiname], [ex get_errmsg]];
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: errorMessage];
        [alert runModal];
        [alert release];
}
@catch (NSException *ex) {
        NSAlert *alert = [[NSAlert alloc] init];
        [alert setMessageText: [ex reason]];
        [alert runModal];
        [alert release];
}
@finally {
        [pdflib release];
}
```

Außer der Methode *get_errmsg* können Sie auch noch das Feld *reason* des Exception-Objekts verwenden, um Fehlermeldungen zu erhalten.

2.8 Perl-Sprachbindung

Der PDFlib-Wrapper für Perl[1] besteht aus einer C-Wrapperdatei und zwei Perl-Paketmodulen, eins zur Bereitstellung eines Perl-Äquivalents für jede PDFlib-API-Funktion und eine anderes für das PDFlib-Objekt. Das C-Modul wird zum Aufbau einer dynamischen Bibliothek verwendet, die vom Perl-Interpreter unter Zuhilfenahme der Paketdatei zur Laufzeit geladen wird. Perl-Skripten referenzieren das Bibliotheksmodul mit einer *use*-Anweisung.

Installation der PDFlib-Perl-Edition. Das Erweiterungsverfahren von Perl lädt dynamische Bibliotheken zur Laufzeit mittels des DynaLoader-Moduls. Perl selbst muss mit einer Option zur Unterstützung dynamischer Bibliotheken kompiliert worden sein (das ist bei den meisten Perl-Konfigurationen der Fall).

Damit die PDFlib-Sprachbindung funktioniert, benötigt der Perl-Interpreter Zugriff auf den PDFlib-Perl-Wrapper und die Module *pdflib_pl.pm* und *PDFlib/PDFlib.pm*. Zusätzlich zu den unten beschriebenen plattformspezifischen Methoden können Sie mit der Perl-Befehlszeilenoption *-I* zum Modulsuchpfad *@INC* ein Verzeichnis hinzufügen, zum Beispiel:

```
perl -I/path/to/pdflib hello.pl
```

Unix. Perl sucht *pdflib_pl.so* (unter OS X: *pdflib_pl.bundle), pdflib_pl.pm* und *PDFlib/PDFlib.pm* im aktuellen Verzeichnis oder in dem Verzeichnis, das mit folgendem Befehl ausgegeben wird:

```
perl -e 'use Config; print $Config{sitearchexp};'
```

Perl durchsucht außerdem das Unterverzeichnis *auto/pdflib_pl*. Der obige Befehl liefert eine Ausgabe, die in etwa wie folgt aussieht:

```
/usr/lib/perl5/site_perl/5.8/i686-linux
```

Windows. PDFlib unterstützt den ActiveState-Port von Perl 5 für Windows namens ActivePerl.[2] Die DLL *pdflib_pl.dll* und die Module *pdflib_pl.pm* und *PDFlib/PDFlib.pm w*erden im aktuellen Verzeichnis gesucht oder im Verzeichnis, das mit folgendem Perl-Befehl ausgegeben wird:

```
perl -e "use Config; print $Config{sitearchexp};"
```

Der obige Befehl liefert eine Ausgabe, die in etwa wie folgt aussieht:

```
C:\Programme\Perl5.8\site\lib
```

1. Siehe www.perl.com
2. Siehe www.activestate.com

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.7 aufgeführten Datentypen übergeben werden.

Tabelle 2.7 Datentypen der Perl-Sprachbindung

API-Datentyp	*Datentypen der Perl-Sprachbindung*
Strings	`string`
Binärdaten	`string`

Fehlerbehandlung in Perl. Die Perl-Sprachbindung übersetzt PDFlib-Fehler in native Perl-Exceptions. Die Perl-Exceptions können durch geeignete Sprachelemente, die kritische Abschnitte klammern, verarbeitet werden, zum Beispiel:

```
eval {
        ...PDFlib-Anweisungen...
};
if ($@) {
        die("$0: PDFlib-Exception in:\n$@");
}
```

Mehrere Arten der Stringbehandlung. Je nach den Anforderungen Ihrer Anwendung können Sie mit UTF-8, UTF-16 oder anderen Encodings arbeiten. Die folgenden Codefragmente zeigen diese drei Varianten. Alle Beispiele erzeugen dieselbe japanische Ausgabe, nehmen die Strings aber in verschiedenen Formaten entgegen.

Das erste Beispiel verwendet Unicode UTF-8 und das Modul *Unicode::String*, das in den meisten neuen Perl-Distributionen enthalten und auf CPAN verfügbar ist. Da Perl intern mit UTF-8 arbeitet, ist keine explizite UTF-8-Konvertierung erforderlich:

```
use Unicode::String qw(utf8 utf16 uhex);
...
$p->set_option("stringformat=utf8");
$font = $p->load_font("Arial Unicode MS", "unicode", "");
$p->setfont($font, 24.0);
$p->fit_textline(uhex("U+65E5 U+672C U+8A9E"), $x, $y, "");
```

Das zweite Beispiel verwendet Unicode UTF-16 und Little-Endian-Bytereihenfolge:

```
$p->set_option("textformat=utf16le");
$font = $p->load_font("Arial Unicode MS", "unicode", "");
$p->setfont($font, 24.0);
$p->fit_textline("\xE5\x65\x2C\x67\x9E\x8A", $x, $y, "");
```

Das dritte Beispiel verwendet Shift-JIS. Außer auf Windows-Systemen benötigt es zur Stringkonvertierung Zugriff auf die CMap *90ms-RKSJ-H:*

```
$p->set_option("searchpath={{../../../resource/cmap}}");
$font = $p->load_font("Arial Unicode MS", "cp932", "");
$p->setfont($font, 24.0);
$p->fit_textline("\x93\xFA\x96\x7B\x8C\xEA", $x, $y, "");
```

Konvertierung von Unicode und anderen Encodings. Um PDFlib-Anwendern die Arbeit zu erleichtern, zeigen wir im Folgenden nützliche Methoden zur Stringkonvertierung. Weitere Informationen finden Sie in der Perl-Dokumentation. Der folgende Konstruktor erzeugt einen UTF-16 Unicode-String aus einem Byte-Array:

```
$logos="\x{039b}\x{03bf}\x{03b3}\x{03bf}\x{03c3}\x{0020}" ;
```

Der folgende Konstruktor erzeugt einen Unicode-String aus einem Unicode-Zeichennamen:

```
$delta = "\N{GREEK CAPITAL LETTER DELTA}";
```

Das Modul *Encode* unterstützt zahlreiche Encodings und besitzt eine Schnittstelle, um Konvertierungen zwischen diesen Encodings durchzuführen:

```
use Encode 'decode';
$data = decode("iso-8859-3", $data);    # nach UTF-8 konvertieren
```

2.9 PHP-Sprachbindung

Installation der PDFlib-Edition für PHP. Ausführliche Informationen über die verschiedenen Möglichkeiten des Einsatzes von PDFlib mit PHP[1] finden Sie in der Datei *PDFlib-in-PHP-HowTo.pdf*, die mit ausgeliefert wird und auf der PDFlib-Website verfügbar ist.

Sie müssen PHP per Konfiguration über die externe PDFlib-Bibliothek informieren. Dazu gibt es zwei Möglichkeiten:

- Fügen Sie in *php.ini* eine der folgenden Zeilen ein:

```
extension=php_pdflib.so        ; für Unix und OS X
extension=php_pdflib.dll       ; für Windows
```

  PHP sucht die Bibliothek in dem Verzeichnis, das unter Unix in der Variablen *extension_dir* in der Datei *php.ini* verzeichnet ist. Unter Windows werden außerdem die Standardsystemverzeichnisse durchsucht. Mit dem folgenden einzeiligen PHP-Skript können Sie ermitteln, welche Version der PDFlib-Sprachbindung für PHP Sie installiert haben:

```
<?phpinfo()?>
```

  Angezeigt wird eine lange Info-Seite über Ihre aktuelle PHP-Konfiguration. Suchen Sie auf der Seite nach dem Abschnitt *PDFlib*.

- Laden Sie PDFlib zur Laufzeit, wobei Sie eine der folgenden Zeilen an den Anfang Ihres Skripts stellen müssen:

```
dl("php_pdflib.so");           # für Unix
dl("php_pdflib.dll");          # für Windows
```

Modifizierte Fehlerrückgabe für PDFlib-Funktionen in PHP. Da PHP per Konvention den Wert 0 (FALSE) zurückgibt, wenn in einer Funktion ein Fehler auftritt, wurden alle PDFlib-Funktionen entsprechend angepasst und liefern im Fehlerfall 0 statt -1. In der PDFlib-Referenz wird in den Funktionsbeschreibungen auf diesen Unterschied hingewiesen. Achten Sie jedoch darauf, wenn Sie die Beispiel-Codefragmente in Kapitel 3, »Erzeugen von PDF-Dokumenten«, Seite 61 durchsehen, da dort entsprechend der üblichen PDFlib-Konvention im Fehlerfall -1 zurückgegeben wird.

Behandlung von Dateinamen in PHP. Nicht qualifizierte Dateinamen (also solche ohne jede Pfadangabe) sowie relative Dateinamen für PDF-, Rasterbild-, Font- und andere Dateien auf dem Laufwerk werden in der Unix- und der Windows-Version von PHP unterschiedlich behandelt:

- Auf Unix-Systemen sucht PHP Dateien ohne Pfadangabe in dem Verzeichnis, in dem sich das Skript befindet.
- Unter Windows sucht PHP Dateien ohne Pfadangabe nur in dem Verzeichnis, in dem sich die PHP-DLL befindet.

Damit Dateinamen unabhängig von der Plattform immer gleich behandelt werden, sollten Sie unbedingt die *SearchPath*-Funktion von PDFlib verwenden (siehe Abschnitt 3.1.3, »Ressourcenkonfiguration und Dateisuche«, Seite 65).

1. Siehe www.php.net

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.8 aufgeführten Datentypen übergeben werden.

Tabelle 2.8 Datentypen der PHP-Sprachbindung

API-Datentyp	*Datentypen der PHP-Sprachbindung*
Strings	`string`
Binärdaten	`string`

Ausnahmebehandlung in PHP. Da PHP 5 strukturierte Ausnahmebehandlung unterstützt, werden PDFlib-Exceptions als PHP-Exceptions weitergeleitet. PDFlib löst im Fehlerfall eine Ausnahme vom Typ *PDFlib-Exception* aus, die von der PHP-Standardklasse *Exception* abgeleitet ist. PDFlib-Exceptions können also mit der üblichen Kombination aus *try* und *catch* abgefangen werden:

```
try {

.. PDFlib-Anweisungen...

} catch (PDFlibException $e) {
    print "PDFlib-Exception aufgetreten:\n";
    print "[" . $e->get_errnum() . "] " . $e->get_apiname() . ": "
                $e->get_errmsg() . "\n";
}
catch (Exception $e) {
    print $e;
}
```

Konvertierung von Unicode und anderen Encodings. Zur Stringkonvertierung können Sie das Modul *iconv* verwenden. Weitere Informationen finden Sie in der PHP-Dokumentation.

PDFlib-Entwicklung mit Eclipse und Zend Studio. Die PHP Development Tools (PDT)[1] unterstützen die PHP-Entwicklung mit Eclipse und Zend Studio. Für PDT kann kontextsensitive Hilfe mit den unten beschriebenen Schritten konfiguriert werden.

Fügen Sie PDFlib zu den Eclipse Voreinstellungen hinzu, um es bei allen PHP-Projekten bekannt zu machen:

- Wählen Sie *Window, Preferences, PHP, PHP Libraries, New...* Ein Wizard wird gestartet.
- Fügen Sie unter *User library name* das Wort *PDFlib ein*, klicken Sie auf *Add External folder...* und wählen Sie das Verzeichnis *bind\php\Eclipse PDT*.

In einem bestehenden oder neuen PHP-Projekt können Sie folgendermaßen einen Verweis auf die PDFlib-Bibliothek legen:

- Klicken Sie im PHP-Explorer mit der rechten Maustaste auf das PHP-Projekt und wählen Sie *Include Path, Configure Include Path...*
- Gehen Sie zur Registermarke *Libraries*, klicken Sie auf *Add Library...* und wählen Sie *User Library*, *PDFlib*.

Dann können Sie in der PHP-Explorer-Ansicht die Liste der PDFlib-Methoden unter dem Knoten *PHP Include Path/PDFlib/PDFlib* durchsuchen. Beim Schreiben von neuem PHP-

1. See www.eclipse.org/pdt

Code bietet Eclipse mit Code-Vervollständigung und kontext-sensitiver Hilfe Unterstützung für alle PDFlib-Methoden.

2.10 Python-Sprachbindung

Installation der Python-Edition von PDFlib. Der Erweiterungsmechanismus von Python[1] lädt dynamische Bibliotheken zur Laufzeit. Damit die PDFlib-Sprachbindung funktioniert, benötigt der Python-Interpreter Zugriff auf die PDFlib-Bibliothek für Python, nach der in den Verzeichnissen gesucht wird, die in der Umgebungsvariable PYTHONPATH aufgeführt sind. Der Name des Python-Wrappers ist plattformabhängig:

- Unix und OS X: *pdflib_py.so*
- Windows: *pdflib_py.pyd*

Neben der PDFlib-Bibliothek müssen die folgenden Dateien in dem Verzeichnis vorhanden sein, in dem die Bibliothek liegt:

- *PDFlib/PDFlib.py*
- *PDFlib/__init__.py*

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.9 aufgeführten Datentypen übergeben werden.

Tabelle 2.9 Datentypen der Python-Sprachbindung

API-Datentyp	*Datentypen der Python-Sprachbindung*
Strings	`string`
Binärdaten	`string`

Fehlerbehandlung in Python. PDFlib installiert einen speziellen Error-Handler, der PDFlib-Fehler in native Python-Exceptions übersetzt. Die Python-Exceptions können mit der üblichen Kombination aus *try* und *catch* behandelt werden:

```
try:
        ...PDFlib-Anweisungen...
except PDFlibException:
        print("PDFlib-Exception in:\n[%d] %s: %s" %
        ((p.get_errnum()), p.get_apiname(),  p.get_errmsg()))

finally:
        p.delete()
```

1. Siehe www.python.org

2.11 REALbasic-Sprachbindung

Installation der REALbasic-Edition von PDFlib. Das PDFlib-Plugin für REALbasic *(PDFlib.rbx) muss* in den Ordner *Plugins* kopiert werden, der sich im selben Ordner wie die REALbasic-Anwendung befindet. Das PDFlib-Plugin für REALbasic wird in einem einzigen Paket ausgeliefert und enthält folgende Varianten:

- OS X Carbon (PowerPC und Intel)
- Windows
- Linux

Das bedeutet, dass Sie die OS X- oder Windows-Version verwenden können, um Anwendungen für alle unterstützten Plattformen zu erzeugen. Beim Erstellen einer eigenständigen Applikation sucht sich REALbasic aus dem PDFlib-Plugin geeignete Bestandteile heraus, bettet aber nur die plattformspezifischen Abschnitte in die generierte Anwendung ein.

Zusätzliche REALbasic-Klassen. Das PDFlib-Plugin erweitert die Objekthierarchie von REALbasic um zwei neue Klassen:

- Die Klasse *PDFlib* enthält alle PDFlib-API-Methoden.
- Mit der Klasse *PDFlibException*, die von *RuntimeException* abgeleitet ist, lassen sich von PDFlib ausgelöste Exceptions behandeln (siehe unten).

Mit PDFlib können sowohl GUI-Anwendungen als auch Anwendungen für die Kommandozeile erstellt werden. Da PDFlib kein Steuerelement ist, wird auch kein neues Symbol in der Steuerelement-Palette von REALbasic installiert. Sobald das PDFlib-Plugin verfügbar ist, ist die Klasse PDFlib mit ihren Methoden jedoch in REALbasic bekannt. So funktionieren zum Beispiel die Befehlsergänzung oder Parameterüberprüfung für PDFlib-API-Methoden in vollem Umfang.

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.10 aufgeführten Datentypen übergeben werden.

Tabelle 2.10 Datentypen der REALbasic-Sprachbindung

API-Datentyp	*Datentypen der REALbasic-Sprachbindung*
Strings	`string`
Binärdaten	`MemoryBlock`

Fehlerbehandlung in REALbasic. Im Fehlerfall löst PDFlib eine REALbasic-Exception der Klasse *PDFlibException* aus. PDFlib-Exceptions lassen sich mit Standardverfahren von REALbasic behandeln: Entweder Sie verwenden einen *try/catch*-Block (empfehlenswert) oder einen Exception-Block.

2.12 RPG-Sprachbindung

PDFlib bietet ein */copy*-Modul, das alle Prototypen sowie einige nützliche Konstanten definiert, die zur Kompilierung von ILE-RPG-Programmen mit eingebetteten PDFlib-Funktionen benötigt werden.

Unicode-String-Behandlung. Da alle in PDFlib verfügbaren Funktionen Unicode-Strings variabler Länge als Parameter verwenden, müssen Sie einen Ein-Byte-String mit der integrierten Funktion *%UCS2* in einen Unicode-String konvertieren. Alle von PDFlib-Funktionen zurückgegebenen Strings sind Unicode-Strings variabler Länge. Mit der integrierten Funktion *%CHAR* konvertieren Sie Unicode-Strings in Ein-Byte-Strings.

Hinweis *Die Funktionen %CHAR und %UCS2 konvertieren Strings anhand der CCSID des aktuellen Jobs von und nach Unicode. Die mit PDFlib mitgelieferten Beispiele basieren auf CCSID 37 (US EBCDIC). Wenn Sie die Beispiele mit anderen Codepages ausführen, werden unter Umständen nicht alle Sonderzeichen in Optionslisten (z.B. { [ ] }) korrekt umgesetzt.*

Da alle Strings als Strings variabler Länge übergeben werden, dürfen Sie keinen Längenparameter in Funktionen angeben, die explizite Übergabe der String-Länge erwarten (die Länge eines Strings variabler Länge wird in den ersten beiden Bytes des Strings gespeichert).

Kompilieren und Binden von RPG-Programmen für PDFlib. Zum Einsatz von PDFlib-Funktionen mit RPG sind die kompilierten Serviceprogramme PDFlib und PDFLIB_RPG erforderlich. Um die PDFlib-Definitionen zur Kompilierzeit in Ihr ILE-RPG-Programm einzufügen, müssen Sie diese in den D-Anweisungen per */copy*-Anweisung angeben:

```
d/copy QRPGLESRC,PDFLIB
```

Wenn die PDFlib-Quelldateibibliothek sich nicht am Anfang Ihrer Bibliotheksliste befindet, müssen Sie zudem die Bibliothek angeben:

```
d/copy PDFsrclib/QRPGLESRC,PDFLIB
```

Bevor Sie mit der Kompilierung des ILE-RPG-Programms beginnen, müssen Sie ein Bindungsverzeichnis anlegen, das die mit PDFlib ausgelieferten Serviceprogramme PDFLIB und PDFLIB_RPG enthält. Das folgende Beispiel zeigt, wie Sie in der Bibliothek PDFLIB das Bindungsverzeichnis PDFLIB erstellen:

```
CRTBNDDIR BNDDIR(PDFLIB/PDFLIB) TEXT('PDFlib Binding Directory')
```

Nach dem Anlegen des Bindungsverzeichnisses müssen Sie die Serviceprogramme PDFLIB und PDFLIB_RPG zu Ihrem Bindungsverzeichnis hinzufügen. Das folgende Beispiel zeigt, wie Sie das Serviceprogramm PDFLIB in der Bibliothek PDFLIB zum bereits erzeugten Bindungsverzeichnis hinzufügen:

```
ADDBNDDIRE BNDDIR(PDFLIB/PDFLIB) OBJ((PDFLIB/PDFLIB *SRVPGM))
ADDBNDDIRE BNDDIR(PDFLIB/PDFLIB) OBJ((PDFLIB/PDFLIB_RPG *SRVPGM))
```

Sie können Ihr Programm nun mit dem Befehl *CRTBNDRPG* (oder der Option 14 in PDM) kompilieren:

```
CRTBNDRPG PGM(PDFLIB/HELLO) SRCFILE(PDFLIB/QRPGLESRC) SRCMBR(*PGM) DFTACTGRP(*NO)
BNDDIR(PDFLIB/PDFLIB)
```

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.9 aufgeführten Datentypen übergeben werden.

Tabelle 2.11 Datentypen der RPG-Sprachbindung

API-Datentyp	*Datentypen der RPG-Sprachbindung*
Strings	*Unicode-String (verwenden Sie* `%ucs2`)
Binärdaten	`data`

Fehlerbehandlung in RPG. In ILE-RPG geschriebene PDFlib-Clients können zur Fehlerbehandlung *monitor/on-error/endmon* von ILE-RPG verwenden. Außerdem können Exceptions auch mit dem globalen Unterprogramm **PSSR* von ILE-RPG überwacht werden. Wenn eine Exception auftritt, zeigt das Job-Protokoll die Fehlernummer, die fehlerhafte Funktion, und den Grund für die Exception an. PDFlib sendet eine Escape-Meldung an das aufrufende Programm.

```
c     eval      p=PDF_new
*
c     monitor
*
c     eval      doc=PDF_begin_document(p:%ucs2('/tmp/my.pdf'):docoptlist)
:
:
*     Fehlerbehandlung
c     on-error
*     Versuchte Fehlerbehebung
*     Vergessen Sie nicht, das PDFlib-Objekt freizugeben
c     callp     PDF_delete(p)
c     endmon
```

2.13 Ruby-Sprachbindung

Installation der Ruby-Edition von PDFlib. Der Erweiterungsmechanismus von Ruby[1] lädt eine dynamische Bibliothek zur Laufzeit. Damit die PDFlib-Sprachbindung funktioniert, benötigt der Ruby-Interpreter Zugriff auf die PDFlib-Erweiterungsbibliothek für Ruby. Diese Bibliothek (unter Windows und Unix: *PDFlib.so*; unter OS X: *PDFlib.bundle)* wird normalerweise im Unterverzeichnis *site_ruby* des lokalen Ruby-Installationsverzeichnisses installiert, das heißt in einem Verzeichnis mit etwa folgendem Namen:

```
/usr/local/lib/ruby/site_ruby/<version>/
```

Ruby durchsucht aber auch andere Verzeichnisse nach Erweiterungen. Mit folgendem Ruby-Aufruf erhalten Sie eine Liste dieser Verzeichnisse:

```
ruby -e "puts $:"
```

Diese Liste enthält in der Regel auch das aktuelle Verzeichnis, so dass Sie die PDFlib-Erweiterungsbibliothek und die Skripten zum Testen einfach ins gleiche Verzeichnis stellen können.

Datentypen. Parameter müssen der PDFlib-Programmschnittstelle (API) gemäß der in Tabelle 2.12 aufgeführten Datentypen übergeben werden.

Tabelle 2.12 Datentypen der Ruby-Sprachbindung

API-Datentyp	*Datentypen der Ruby-Sprachbindung*
Strings	`string`
Binärdaten	`string`

Fehlerbehandlung in Ruby. Die Ruby-Sprachbindung installiert einen Error-Handler, der PDFlib-Exceptions in native Ruby-Exceptions übersetzt. Die Ruby-Exceptions können mit der üblichen *rescue*-Technik behandelt werden:

```
begin
        ...PDFlib-Anweisungen...
rescue  PDFlibException => pe
  print "PDFlib-Exception im Beispiel Hello:\n"
  print "[" + pe.get_errnum.to_s + "] " + pe.get_apiname + ": " + pe.get_errmsg + "\n"
end
```

Ruby on Rails. Ruby on Rails[2] ist ein Open-Source-Framework, das die Webentwicklung mit Ruby erleichtert. Die PDFlib-Erweiterung für Ruby ist auch mit Ruby on Rails einsetzbar; entsprechende Beispiele sind im Paket enthalten. Um die PDFlib-Beispiele für Ruby on Rails auszuführen, gehen Sie wie folgt vor:

- Installieren Sie Ruby und Ruby on Rails.
- Richten Sie einen neuen Controller von der Kommandozeile aus ein:

```
$ rails new pdflibdemo
```

1. Siehe www.ruby-lang.org/en
2. Siehe www.rubyonrails.org

```
$ cd pdflibdemo
$ cp <PDFlib dir>/bind/ruby/<version>/PDFlib.so vendor/  # use .so/.dll/.bundle
$ rails generate controller home demo
$ rm public/index.html
```

- Editieren Sie *config/routes.rb:*

```
...
# Vergessen Sie nicht, public/index.html zu löschen
root :to => "home#demo"
```

- Editieren Sie *app/controllers/home_controller.rb* wie unten beschrieben und fügen Sie PDFlib-Code zur Erstellung des PDF-Inhalts ein. Denken Sie daran, das die PDF-Ausgabe im Speicher erzeugt werden muss, das heißt *begin_document( )* muss ein leerer Dateiname mitgegeben werden. Als Ausgangspunkt können Sie den Code aus dem Beispiel *hello-rails.rb* verwenden:

```
class HomeController < ApplicationController
def demo
require "PDFlib"
begin
     p = PDFlib.new
     ...
     ...PDFlib-Anwendungscode, siehe hello-rails.rb...
     ...
     send_data p.get_buffer(), :filename => "hello.pdf",
     :type => "application/pdf", :disposition => "inline"
     rescue PDFlibException => pe
     # error handling
end
end
end
```

- Um die Installation zu testen, starten Sie den WEBrick-Server mit folgendem Kommando:

```
$ rails server
```

  Geben Sie im Browser *http://0.0.0.0:3000* ein. Das generierte PDF-Dokument wird im Browser angezeigt.

Lokale PDFlib-Installation. Wenn Sie PDFlib nur mit Ruby on Rails einsetzen möchten und nicht global zum allgemeinen Einsatz mit Ruby installieren können, können Sie PDFlib auch lokal im Verzeichnis *vendors* in der Rails-Hierarchie installieren. Dies ist insbesondere dann nützlich, wenn Sie nicht über die Berechtigung verfügen, allgemeine Ruby-Erweiterungen zu installieren, aber in Rails mit PDFlib arbeiten möchten.

3 Erzeugen von PDF-Dokumenten

3.1 Allgemeine Aspekte der PDFlib-Programmierung

Cookbook *Codebeispiele zu allgemeinen Aspekten der PDFlib-Programmierung finden Sie in der Kategorie* `general` *des PDFlib Cookbook.*

3.1.1 Behandlung von Ausnahmen (Exceptions)

Eine bestimmte Art von Fehlern wird in vielen Sprachen zurecht als Ausnahme *(Exception)* bezeichnet – es handelt sich um bloße Ausnahmesituationen, die während eines Programmlaufs nicht allzu häufig erwartet werden. Die generelle Strategie besteht darin, konventionelle Verfahren zur Fehlerbenachrichtigung (also besondere Funktionsrückgabewerte wie -1) für solche Funktionsaufrufe zu verwenden, die häufig fehlschlagen. Besondere Verfahren zur Ausnahmebehandlung setzt man dagegen in selten zu erwartenden Fällen ein, für die man den Code nicht mit Bedingungen zupflastern möchte. In dieser Art verfährt auch PDFlib. So geht man bei manchen Operationen davon aus, dass sie relativ häufig schiefgehen, zum Beispiel:

- der Versuch, ohne entsprechende Berechtigung eine Ausgabedatei zu öffnen
- der Versuch, eine Eingabe-PDF-Datei mit einem falschen Pfadnamen zu öffnen
- der Versuch, eine beschädigte Bilddatei zu öffnen

PDFlib zeigt solche Fehler durch die Rückgabe eines speziellen Wertes an, der in der PDFlib-Referenz angegeben wird (normalerweise –1, aber 0 in der PHP-Sprachbindung). Dieser Fehlercode muss vom Anwendungsentwickler bei allen Funktionen überprüft werden, die im Fehlerfall -1 zurückgeben. Andere Ereignisse sind möglicherweise schädlich, treten aber eher selten auf, zum Beispiel:

- es ist kein Speicher mehr verfügbar;
- Verletzungen des Geltungsbereichs (zum Beispiel das Schließen eines Dokuments vor dem Öffnen);
- die Übergabe falscher Parameter an PDFlib-Funktionen (zum Beispiel der Versuch, einen Kreis mit negativem Radius zu zeichnen) oder die Übergabe falscher Optionen.

Stößt PDFlib auf eine solche Ausnahmesituation, so wird kein spezieller Fehlerwert an die aufrufenden Funktion zurückgegeben, sondern eine Exception ausgelöst. Es ist wichtig sich klarzumachen, dass die Generierung des PDF-Dokuments nicht abgeschlossen werden kann, wenn eine Exception auftritt. Die einzigen Methoden, die nach einer Exception noch aufgerufen werden dürfen, sind *PDF_delete( ), PDF_get_apiname( ), PDF_get_errnum( )* und *PDF_get_errmsg( )*. Alle anderen PDFlib-Methoden müssen vermieden werden. In der Exception sind folgende Informationen enthalten:

- eine eindeutige Fehlernummer;
- der Name der PDFlib-API-Funktion, die die Exception ausgelöst hat;
- ein beschreibender Text mit detaillierten Angaben zum Problem.

Abfragen der Ursache für einen gescheiterten Funktionsaufruf. Wie oben erwähnt, muss die Generierung der PDF-Ausgabe beim Auftreten einer Exception auf jeden Fall abgebrochen werden. Signalisiert eine Funktion durch Rückgabe eines Fehlerwerts dagegen ein nicht fatales Problem, kann die Anwendung das Dokument fortsetzen. Dazu

können der Programmfluss oder die Eingabedaten geändert werden. Kann beispielsweise ein bestimmter Font nicht geladen werden, brechen die meisten Clients die Dokumentgenerierung ab; es mag aber auch Clients geben, die mit einem anderen Font fortfahren möchten. In diesem Fall ist es unter Umständen wünschenswert, eine Fehlermeldung zu erhalten, die das Problem genauer beschreibt. Dazu können direkt nach einem gescheiterten Funktionsaufruf, das heißt, nach einem Funktionsaufruf, der den Fehlerwert -1 (in PHP: 0) liefert, die Funktionen *PDF_get_errnum( )*, *PDF_get_errmsg( )* und *PDF_get_apiname( )* aufgerufen werden.

Strategien zur Fehlerbehandlung. Stößt PDFlib auf einen Fehler, so kann die Reaktion gemäß einer von mehreren Strategien erfolgen, die sich mit der Option *errorpolicy* konfigurieren lässt. Alle Funktionen, die Fehlercodes zurückgeben können, unterstützen auch die Option *errorpolicy*. Folgende Fehlerstrategien werden unterstützt:

- *errorpolicy=legacy:* Diese abgekündigte Einstellung gewährleistet ein Verhalten, das zu früheren PDFlib-Versionen kompatibel ist, wo Exceptions und Fehlerrückgabewerte von Optionen wie *fontwarning*, *imagewarning* etc. gesteuert wurden. Diese Einstellung ist nur in Anwendungen sinnvoll, die Quellcodekompatibilität zu PDFlib 6 benötigen. Sie sollte in neu erstellten Anwendungen nicht verwendet werden. Die Einstellung *legacy* ist die Standardfehlerstrategie.
- *errorpolicy=return:* Tritt ein Fehler auf, gibt die entsprechende Funktion unabhängig von allen *warning*-Optionen den Fehlerwert -1 (in PHP: 0) zurück. Um das Problem zu erkennen, muss der Anwendungsentwickler den Rückgabewert überprüfen und je nach Anwendung angemessen reagieren. Wir empfehlen diese Einstellung, da sie eine einheitliche Fehlerbehandlung ermöglicht.
- *errorpolicy=exception:* Im Fehlerfall wird eine Exception ausgelöst. Das Ausgabedokument ist nach einer Exception aber unvollständig und nicht mehr zu gebrauchen. Diese Einstellung ist bei nachlässiger Programmierung ohne jede Fehlerabfrage denkbar. Das Ausgabedokument wird jedoch geopfert, auch wenn das Problem von der Anwendung vielleicht behoben werden könnte.

Die folgenden Codefragmente zeigen verschiedene Strategien in Bezug auf die Ausnahmebehandlung. Die Beispiele versuchen, einen Font zu laden, der eventuell nicht verfügbar ist.

Bei *errorpolicy=return* muss der Rückgabewert auf einen Fehlerwert überprüft werden. Liegt ein Fehler vor, kann die Fehlerursache abgefragt und damit angemessen auf die Situation reagiert werden:

```
font = p.load_font("MyFontName", "unicode", "errorpolicy=return");
if (font == -1)
{
        /* Fonthandle ist ungültig; Ursache ermitteln */
        errmsg = p.get_errmsg());
        /* Anderen Font versuchen oder abbrechen */
        ...
}
/* Fonthandle ist gültig; fortfahren */
```

Bei *errorpolicy=exception* muss das Dokument beim Auftreten eines Fehlers aufgegeben werden:

```
font = p.load_font("MyFontName", "unicode", "errorpolicy=exception");
/* Wenn keine Exception auslöst wurde, ist das Fonthandle gültig;
```

```
 * Bei einer Exception kann die PDF-Ausgabe nicht fortgesetzt werden
 */
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/error_handling`.

Warnungen. Manche Probleme werden von PDFlib zwar intern erkannt, eine Programmunterbrechung durch Auslösen einer Exception wäre aber nicht gerechtfertigt. Anstatt eine Exceptions auszulösen, wird eine Fehlerbeschreibung protokolliert. Das Logging lässt sich wie folgt aktivieren:

```
p.set_option("logging", "filename=private.log");
```

In Bezug auf Warnungen empfehlen wir folgendes Vorgehen:

- Aktivieren Sie das Logging in der Entwicklungsphase und sehen Sie sich alle Warnmeldungen in der Log-Datei genau an. Vielleicht liegt ein Problem in Ihrem Code oder Ihren Daten vor. Deshalb sollten Sie unbedingt versuchen, die Ursache einer jeden Warnung zu verstehen oder sie zu beseitigen.
- Deaktivieren Sie das Logging im produktiven Einsatz, und aktivieren Sie es nur noch bei Problemen.

3.1.2 Das PDFlib Virtual File System (PVF)

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/starter_pvf`.

Neben Dateien im Dateisystem gibt es eine Technik namens *PDFlib Virtual File System* (PVF), die es Clients ermöglicht, Daten direkt im Speicher zu übergeben. Dies ist weitaus schneller und kann zum Beispiel für Daten verwendet werden, die aus einer Datenbank stammen und gar nicht einzeln als Datei existieren. Es sind ebenso Situationen denkbar, wo der Client die erforderlichen Daten als Ergebnis anderer Verarbeitungsschritte bereits im Speicher zur Verfügung gestellt bekommt.

PVF basiert auf dem Konzept virtueller Read-Only-Dateien, deren Namen wie normale Dateinamen in jeder API-Funktion verwendet werden können. Darüber hinaus sind sie in UPR-Konfigurationsdateien einsetzbar. Die Namen für virtuelle Dateien können vom Client auf beliebige Art erzeugt werden. Natürlich müssen sie so gewählt werden, dass es keine Namenskonflikte mit regulären Dateien gibt. Deshalb sind für virtuelle Dateinamen folgende hierarchische Namenskonventionen empfehlenswert (*filename* bezieht sich auf einen vom Client festgelegten Namen, der in der entsprechenden Kategorie eindeutig ist). Außerdem sollten übliche Namenssuffixe beibehalten werden:

- Rasterbilddateien: */pvf/image/filename*
- Font- und Metrikdateien (der tatsächliche Fontname sollte die Basis des Dateinamens bilden): */pvf/font/filename*
- ICC-Profile: */pvf/iccprofile/filename*
- PDF-Dokumente: */pvf/pdf/filename*

Zum Auffinden einer Datei überprüft PDFlib zuerst, ob sich der Name auf eine virtuelle Datei bezieht, und versucht dann, die Datei auf der Festplatte zu öffnen.

Lebensdauer virtueller Dateien. Manche Funktionen verwenden die in einer virtuellen Datei übergebenen Daten sofort, während andere einzelne Teile zu verschiedenen Zeitpunkten lesen. Aus diesem Grund muss man die Lebensdauer virtueller Dateien be-

achten. PDFlib versieht jede virtuelle Datei mit einer internen Sperre, die erst wieder entfernt wird, wenn der Dateiinhalt nicht mehr benötigt wird. Sofern der Client die Daten (mit der Option *copy* in *PDF_create_pvf( )*) nicht sofort von PDFlib kopieren lässt, darf er sie erst ändern, löschen oder freigeben, nachdem die Sperre von PDFlib aufgehoben wurde. PDFlib löscht mit *PDF_delete( )* automatisch auch alle virtuellen Dateien. Der eigentliche Inhalt (die in der virtuellen Datei enthaltenen Daten) muss jedoch immer vom Client freigegeben werden.

Verschiedene Strategien. PVF unterstützt verschiedene Ansätze im Hinblick auf die für virtuelle Dateien erforderliche Speicherverwaltung. Diese sind darauf ausgerichtet, dass PDFlib nach dem API-Aufruf, der einen virtuellen Dateinamen verarbeitet hat, unter Umständen Zugriff auf den Dateiinhalt benötigt, dass dies aber niemals nach *PDF_end_document( )* erforderlich ist. *PDF_delete_pvf( )* gibt den eigentlichen Dateiinhalt nicht frei (außer die Option *copy* wurde übergeben), sondern nur die für die PVF-Dateinamenverwaltung verwendeten Datenstrukturen. Damit ergeben sich folgende Strategien:

- Minimieren des Speicherbedarfs: Es ist empfehlenswert, *PDF_delete_pvf( )* unmittelbar nach dem API-Aufruf, der den virtuellen Dateinamen verarbeitet hat, auszuführen und ein weiteres Mal nach *PDF_end_document( )*. Der zweite Aufruf ist erforderlich, da PDFlib unter Umständen noch Zugriff auf die Daten benötigt, so dass die Sperre auf die virtuelle Datei eventuell beim ersten Aufruf nicht aufgehoben wird. Sicherlich werden die Daten in einigen Fällen bereits beim ersten Aufruf freigegeben, aber der zweite Aufruf richtet keinen Schaden an. Der Client darf den Dateiinhalt nur nach erfolgreicher Ausführung von *PDF_delete_pvf( )* freigeben.
- Optimieren der Geschwindigkeit durch mehrfache Verwendung virtueller Dateien: Manche Clients möchten bestimmte Daten (zum Beispiel Fonts) vielleicht mehrmals verwenden, etwa in verschiedenen Ausgabedokumenten, und sich eine Wiederholung von *create/delete*-Zyklen für ein und denselben Dateiinhalt sparen. In diesem Fall ist es empfehlenswert, *PDF_delete_pvf( )* erst aufzurufen, wenn keine weiteren PDF-Ausgabedokumente auf Basis der virtuellen Datei mehr generiert werden.
- Faule Programmierung: Ist der Speicherbedarf kein Thema, braucht der Client *PDF_delete_pvf( )* überhaupt nicht aufzurufen. In diesem Fall löscht PDFlib alle angelegten virtuellen Dateien beim Aufruf von *PDF_delete( )*.

In allen Fällen darf der Client die entsprechenden Daten nur nach einem erfolgreichen Aufruf von *PDF_delete_pvf( )* oder nach *PDF_delete( )* freigeben.

Erzeugen von PDF-Ausgabe in einer virtuellen Datei. Neben der Bereitstellung von Benutzerdaten an PDFlib kann PVF auch die von PDFlib erzeugten PDF-Dokumentdaten verwalten. Dies kann durch Übergabe der Option *createpvf* an *PDF_begin_document( )* erreicht werden. Der PVF-Dateiname kann später an andere API-Funktionen von PDFlib übergeben werden. Dies ist zum Beispiel sinnvoll bei der Erzeugung von PDF-Dokumenten für die Aufnahme in ein PDF-Portfolio. Es ist nicht möglich, die durch PDFlib erzeugten PVF-Daten direkt abzurufen. Verwenden Sie die aktive oder passive *in-core*-Schnittstelle zur PDF-Generierung, um die PDF-Daten direkt im Arbeitsspeicher zu übergeben (siehe Abschnitt 3.1.4, »Erzeugen von PDF-Dokumenten im Arbeitsspeicher«, Seite 70).

3.1.3 Ressourcenkonfiguration und Dateisuche

Bei den meisten komplexeren Anwendungen benötigt PDFlib Zugriff auf Ressourcen wie Fontdateien, ICC-Farbprofile usw. Um die Ressourcenverarbeitung von PDFlib plattformunabhängig und benutzerdefinierbar zu machen, kann eine Konfigurationsdatei angegeben werden, in der die verfügbaren Ressourcen gemeinsam mit ihren Dateinamen aufgeführt sind. Außerdem kann die Konfiguration dynamisch zur Laufzeit durch Anfügen von Ressourcen mit *PDF_set_option( )* erfolgen. Für die Konfigurationsdatei verwendet PDFlib ein einfaches Textformat namens *Unix PostScript Resource* (UPR). Wir haben das ursprüngliche UPR-Format für unsere Zwecke erweitert. Das von PDFlib verwendete UPR-Dateiformat wird unten beschrieben.

Mit der Option *enumeratefonts* kann PDFlib angewiesen werden, alle unter dem Suchpfad verfügbaren Fonts zu sammeln (siehe »Dateisuche und Ressourcenkategorie SearchPath«, Seite 66). Mit der Option *saveresources* kann die aktuelle Liste der PDFlib-Ressourcen in einer Datei gespeichert werden:

```
/* Font-Verzeichnis dem Suchpfad hinzufügen */
p.set_option("searchpath={{C:/fonts}}");

/* Alle Fonts im Suchpfad sammeln und eine UPR-Datei erzeugen */
p.set_option("enumeratefonts saveresources={filename={C:/fonts/pdflib.upr}}");
```

Ressourcenkategorien. Tabelle 3.1 führt die von PDFlib unterstützten Ressourcenkategorien auf. Die meisten Kategorien ordnen einen Ressourcennamen (der im PDFlib-API verwendet wird) einer virtuellen Datei oder einer Datei im Dateisystem zu. Andere Ressourcenkategorien als in Tabelle 3.1 werden ignoriert. Bei den Namen der Kategorien wird zwischen Groß-/Kleinschreibung unterschieden. Die Werte werden als Name-Strings behandelt; sie können in ASCII oder UTF-8 (mit BOM am Anfang einer Zeile), oder in EBCDIC-UTF-8 unter zSeries kodiert sein. Unicode-Werte können bei der Ressource *HostFont* bei lokalisierten Fontnamen nützlich sein.

Tabelle 3.1 In PDFlib unterstützte Ressourcenkategorien

Kategorie	Format	Erklärung
SearchPath	`pathname`	*Relativer oder absoluter Pfadname von Verzeichnissen mit Datenfiles*
CMap	`cmapname=filename`	*CMap-Datei für CJK-Encoding*
FontAFM	`fontname=filename`	*PostScript-Fontmetrikdatei im AFM-Format*
FontPFM	`fontname=filename`	*PostScript-Fontmetrikdatei im PFM-Format*
FontOutline	`fontname=filename`	*PostScript-, TrueType-, OpenType-,WOFF- oder CEF-Fontdatei*
Encoding	`encodingname=filename`	*Textdatei mit 8-Bit-Zeichensatz oder Codepage-Tabelle*
HostFont	`fontname=hostfontname`	*Name eines im System installierten Fonts (normalerweise sind beide Fontnamen identisch)*
FontnameAlias	`aliasname=fontname`	*Erzeugen eines Alias für einen Font, der PDFlib bereits bekannt ist*
ICCProfile	`profilename=filename`	*Name eines ICC-Farbprofils*

Das UPR-Dateiformat. UPR-Dateien liegen im Textformat vor und sind sehr einfach aufgebaut, so dass sie problemlos manuell im Texteditor oder auch automatisch erstellt werden können. Beginnen wir mit der Syntax:

- Eine Zeile besteht aus maximal 1023 Zeichen.
- Ein Gegenschrägstrich '\' am Zeilenende hebt das Zeilenende auf. Dies kann zur Zeilenverlängerung verwendet werden.
- Ein Prozentzeichen '%' leitet einen Kommentar bis zum Ende der Zeile ein. Prozentzeichen, die Teil der Zeilendaten sind (d.h. die keinen Kommentar einleiten), müssen mit einem vorangestellten Backslash geschützt werden.
- Backslash-Zeichen vor einem Backslash, die das Zeilenende schützen sowie Backslash-Zeichen, die ein Prozentzeichen schützen, müssen dupliziert werden, wenn sie Teil der Zeilendaten sind.
- Ein einzelner Punkt ' . ' dient als Abschnittsende.
- Es wird zwischen Groß- und Kleinschreibung unterschieden.
- Leer- und Tabulatorzeichen werden ignoriert, außer in Ressourcen- und Dateinamen.
- Ressourcennamen und -werte dürfen kein Gleichheitszeichen '=' enthalten.
- Wenn eine Ressource mehrfach definiert wird, überschreibt die letzte Definition die früheren Definitionen.

UPR-Dateien bestehen aus folgenden Komponenten:

- Eine Kopfzeile zur Identifizierung der Datei, die folgendermaßen aussieht:

```
PS-Resources-1.0
```

- Ein optionaler Abschnitt, der alle Ressourcenkategorien auflistet, die in der Datei beschrieben werden. Jede Zeile beschreibt eine Kategorie. Die Liste wird mit einem Punkt abgeschlossen. Die verfügbaren Ressourcenkategorien werden unten beschrieben. Wenn dieser optionale Abschnitt fehlt, muss trotzdem ein einzelner Punkt vorhanden sein.
- Ein Abschnitt für jede der Ressourcenkategorien, die am Dateianfang aufgeführt wurden. Jeder Abschnitt beginnt mit einer Zeile für die Ressourcenkategorie, gefolgt von einer beliebigen Anzahl von Zeilen, die die verfügbaren Ressourcen beschreiben. Diese Liste wird durch eine Zeile mit einem einzelnen Punkt abgeschlossen. Jede Ressourcenzeile besteht aus dem Namen der Ressource (bei Gleichheitszeichen sind Anführungszeichen erforderlich). Erfordert die Ressource einen Dateinamen, muss dieser nach einem Gleichheitszeichen angefügt werden. PDFlib berücksichtigt die Option *SearchPath* (siehe unten) bei der Suche nach Dateien, deren Namen als Ressourcen eingetragen sind.

Dateisuche und Ressourcenkategorie SearchPath. PDFlib liest verschiedenste Daten, zum Beispiel Rasterbilder, Font- und Metrikdaten, PDF-Dokumente oder ICC-Farbprofile aus Dateien des Dateisystems. Neben relativen und absoluten Pfadnamen können Sie Dateinamen auch ohne jede Pfadangabe verwenden. Dazu definieren Sie mit der Ressourcenkategorie *SearchPath* eine Liste von Pfadnamen für die Verzeichnisse, die die benötigten Dateien enthalten. Beim Öffnen einer Datei versucht PDFlib zuerst, diese mit genau dem übergebenen Dateinamen zu öffnen. Schlägt dieser Versuch fehl, sucht PDFlib nacheinander in den Verzeichnissen, die in der Ressourcenkategorie *SearchPath* aufgeführt sind. Aus den *SearchPath*-Einträgen wird eine Liste aufgebaut, die in umgekehrter Reihenfolge durchsucht wird (später hinzugefügte Pfade werden also zuerst

durchsucht). Mit diesem Verfahren sind PDFlib-Anwendungen nicht mehr von plattformspezifischen Dateisystemkonventionen abhängig. *SearchPath*-Einträge können Sie wie folgt setzen:

```
p.set_option("SearchPath={{/pfad/zu/verzeichnis1}}");
p.set_option("SearchPath={{/pfad/zu/verzeichnis2}}");
```

Der Suchpfad kann mehrfach gesetzt werden, und mehrere Verzeichnisnamen können in einem einzigen Aufruf übergeben werden. Es wird empfohlen, selbst für einen einzigen Eintrag doppelt geschweifte Klammern zu verwenden, um Probleme bei Verzeichnisnamen mit Leerzeichen zu vermeiden. Eine leere String-Liste (das heißt {{}}) löscht alle vorhandenen Suchpfad-Einträge einschließlich der Standardeinträge.

Um diesen Suchmechanismus zu unterbinden, geben Sie in den jeweiligen PDFlib-Funktionsaufrufen vollständige Pfadnamen an. Beachten Sie die folgenden plattformabhängigen Eigenschaften der Ressourcenkategorie *SearchPath:*

- Unter Windows initialisiert PDFlib die Ressourcenkategorie *SearchPath* mit Einträgen aus der Registry. Die folgenden Registry-Einträge können eine Liste von Pfadnamen enthalten, die durch Strichpunkte ';' getrennt sind. Sie werden in der unten angegebenen Reihenfolge durchsucht:

  ```
  HKLM\SOFTWARE\PDFlib\PDFlib9\9.0.1\SearchPath
  HKLM\SOFTWARE\PDFlib\PDFlib9\SearchPath
  HKLM\SOFTWARE\PDFlib\SearchPath
  ```

- Die Installationsroutine für COM initialisiert den Registry-Eintrag *SearchPath* mit folgendem Verzeichnis (oder einem ähnlichen Verzeichnis, falls Sie PDFlib in einem anderen Verzeichnis installiert haben):

  ```
  C:\Programme\PDFlib\PDFlib 9.0.1\resource
  ```

- Unter i5/iSeries wird die Ressourcenkategorie *SearchPath* mit folgenden Werten initialisiert:

  ```
  /PDFlib/PDFlib/9.0/resource/icc
  /PDFlib/PDFlib/9.0/resource/fonts
  /PDFlib/PDFlib/9.0/resource/cmap
  /PDFlib/PDFlib/9.0
  /PDFlib/PDFlib
  /PDFlib
  ```

  Der letzte Eintrag ist besonders nützlich zum Speichern einer Lizenzdatei für mehrere Produkte.

Voreingestellte Dateisuchpfade. Unter Unix, Linux, OS X und i5/iSeries werden per Voreinstellung einige Verzeichnisse standardmäßig sogar ohne Angabe von Pfad und Verzeichnisnamen nach Dateien durchsucht. Vor der Suche und dem Lesen der UPR-Datei (welche zusätzliche Suchpfade enthalten können), werden die folgenden Verzeichnisse durchsucht:

```
<rootpath>/PDFlib/PDFlib/9.0/resource/cmap
<rootpath>/PDFlib/PDFlib/9.0/resource/codelist
<rootpath>/PDFlib/PDFlib/9.0/resource/glyphlst
<rootpath>/PDFlib/PDFlib/9.0/resource/fonts
<rootpath>/PDFlib/PDFlib/9.0/resource/icc
<rootpath>/PDFlib/PDFlib/9.0
```

```
<rootpath>/PDFlib/PDFlib
<rootpath>/PDFlib
```

Unter Unix, Linux und OS X wird *<rootpath>* zuerst durch */usr/local* und dann durch das HOME-Verzeichnis ersetzt. Unter i5/iSeries ist *<rootpath>* leer.

Voreingestellte Dateinamen für Lizenz- und Ressourcendateien. Standardmäßig werden die folgenden Dateinamen in den Standard-Verzeichnissen für Suchpfade gesucht:

```
licensekeys.txt             (Lizenzdatei; auf MVS: license)
pdflib.upr                  (Ressourcedatei; auf MVS: upr)
```

Mit diesem Feature kann man Lizenzdateien verwenden, ohne Umgebungsvariablen oder Laufzeitoptionen setzen zu müssen.

UPR-Beispieldatei. Das folgende Listing zeigt ein Beispiel für eine UPR-Konfigurationsdatei:

```
PS-Resources-1.0
.
SearchPath
/usr/local/lib/fonts
C:/psfonts/pfm
C:/psfonts
/users/kurt/my_images
.
FontAFM
Code-128=Code_128.afm
.
FontPFM
Corporate-Bold=corpb___.pfm
Mistral=c:/psfonts/pfm/mist____.pfm
.
FontOutline
Code-128=Code_128.pfa
ArialMT=Arial.ttf
.
HostFont
Wingdings=Wingdings
.
ICCProfile
highspeedprinter=cmykhighspeed.icc
.
```

Suchen der UPR-Ressourcendatei. Sollen nur in PDFlib integrierte Ressourcen (zum Beispiel PDF-Standardfonts, ICC-Profil sRGB) oder Systemressourcen (Host-Fonts) verwendet werden, dann ist keine UPR-Konfigurationsdatei erforderlich, da PDFlib selbst über alle notwendigen Ressourcen verfügt.

Sollen andere Ressourcen verwendet werden, so können Sie diese mit *PDF_set_option()* (siehe unten) oder in einer UPR-Ressourcendatei festlegen. PDFlib liest diese Datei automatisch, sobald die erste Ressource abgefragt wird. Im Einzelnen wird wie folgt vorgegangen:

- Unter Unix, Linux, OS X und i5/iSeries werden einige Verzeichnisse standardmäßig sogar ohne Angabe von Pfad und Verzeichnisnamen nach Lizenz- und Ressourcenda-

teien durchsucht. Vor der Suche und dem Lesen der UPR-Datei werden die folgenden Verzeichnisse in der angegebenen Reihenfolge durchsucht:

```
<rootpath>/PDFlib/PDFlib/9.0/resource/icc
<rootpath>/PDFlib/PDFlib/9.0/resource/fonts
<rootpath>/PDFlib/PDFlib/9.0/resource/cmap
<rootpath>/PDFlib/PDFlib/9.0
<rootpath>/PDFlib/PDFlib
<rootpath>/PDFlib
```

Unter Unix, Linux und OS X wird *<rootpath>* zuerst durch */usr/local* und dann durch das HOME-Verzeichnis ersetzt. Auf i5/iSeries ist *<roothpath>* leer. Mit diesem Feature kann man Lizenz-, UPR-Dateien oder Ressourcen verwenden, ohne Umgebungsvariablen oder Laufzeitoptionen setzen zu müssen.

- Ist die Umgebungsvariable *PDFLIBRESOURCE* definiert, verwendet PDFlib deren Wert als Name für die zu lesende UPR-Datei. Kann diese Datei nicht gelesen werden, wird eine Exception ausgelöst.
- Ist die Umgebungsvariable *PDFLIBRESOURCE* nicht definiert, versucht PDFlib eine Datei mit folgendem Namen zu öffnen:

```
upr (auf MVS; ein Dataset wird erwartet)
pdflib/<version>/fonts/pdflib.upr (auf IBM i5/iSeries)
pdflib.upr (unter Windows, Unix und allen anderen Systemen)
```

Kann diese Datei nicht gelesen werden, wird eine Exception ausgelöst.

- Unter Windows versucht PDFlib zudem, die folgenden Registry-Einträge in der angegeben Reihenfolge zu lesen:

```
HKLM\Software\PDFlib\PDFlib\9.0.1\resourcefile
HKLM\Software\PDFlib\PDFlib\resourcefile
HKLM\Software\PDFlib\resourcefile
```

Die Werte dieser Einträge werden als Name der zu lesenden Ressourcendatei verwendet. Kann diese Datei nicht gelesen werden, wird eine Exception ausgelöst. Seien Sie vorsichtig beim manuellen Zugriff auf die Registry unter 64-Bit Windows-Systemen: wie üblich arbeitet die 64-Bit PDFlib-Binärversion mit der 64-Bit-Ansicht der Windows Registry, während die 32-Bit PDFlib-Binärversion mit der 32-Bit-Ansicht der Registry arbeitet. Wenn Sie Registry-Schlüssel für ein 32-Bit-Produkt manuell eintragen müssen, achten Sie darauf, die 32-Bit-Version des Werkzeugs *regedit* zu verwenden. Sie können es folgendermaßen über das Startmenü aufrufen:

```
%systemroot%\syswow64\regedit
```

- Durch explizites Setzen der Option *resourcefile* kann der Client PDFlib veranlassen, eine Ressourcendatei zur Laufzeit einzulesen:

```
p.set_option("resourcefile={/path/to/pdflib.upr}");
```

Dieser Aufruf kann beliebig oft wiederholt werden; die Ressourceneinträge werden akkumuliert.

Konfiguration von Ressourcen zur Laufzeit. Statt eine UPR-Datei zur Konfiguration zu verwenden, können Sie einzelne Ressourcen mit *PDF_set_option( )* direkt im Quellcode konfigurieren. Diese Funktion erhält einen Kategorienamen sowie den zugehörenden

Ressourceneintrag so, wie diese auch im entsprechenden Abschnitt in der UPR-Datei erscheinen, zum Beispiel:

```
p.set_option("FontAFM={Foobar-Bold=foobb___.afm}");
p.set_option("FontOutline={Foobar-Bold=foobb___.pfa}");
```

Hinweis *Für weitere Informationen zur Fontkonfiguration siehe Abschnitt 5.4.4, »Suche nach Fonts«, Seite 136.*

Abfrage von Ressourcenwerten. Neben der Konfiguration von Ressourceneinträgen können Sie Werte mit *PDF_get_option( )* abfragen. Dabei geben Sie den Kategorienamen als Schlüssel und die Ressourcennummer (Beginn bei 1) als Option an. Den *n*-ten Eintrag in der Liste *SearchPath* fragen Sie beispielsweise mit folgendem Aufruf ab:

```
idx = p.get_option("SearchPath", "resourcenumber=" + n);
sp = p.get_string(idx, "");
```

Ist *n* größer als die Anzahl aller verfügbaren Einträge in der gewünschten Kategorie, wird ein leerer String zurückgegeben. Der zurückgegebene String ist bis zum nächsten Aufruf einer API-Funktion gültig.

3.1.4 Erzeugen von PDF-Dokumenten im Arbeitsspeicher

Neben der Generierung von PDF-Dokumentdateien kann PDFlib auch dazu veranlasst werden, PDF-Dokumente direkt im Arbeitsspeicher zu erzeugen *(in-core)*. Dieses Verfahren ist schneller, da kein Schreiben auf die Festplatte erforderlich ist. Das PDF-Dokument kann dann zum Beispiel direkt via HTTP übertragen werden. Es dürfte insbesondere Webmaster erfreuen zu hören, dass ihr Server nicht unbedingt mit temporären PDF-Dateien überhäuft werden muss.

Mit dieser Art der PDF-Generierung können Sie die Daten entweder stückweise abholen (zum Beispiel jedes Mal, wenn eine Seite komplett ist) oder das vollständige PDF-Dokument am Ende in einem Stück (nach *PDF_end_document( )*) verwenden. Die stückweise Generierung und Weiterverarbeitung der PDF-Daten ist in mehrerer Hinsicht vorteilhaft. Erstens sinken die Speicheranforderungen, da die Daten nicht vollständig im Speicher vorgehalten werden müssen. Außerdem lässt sich die Leistung erheblich steigern, wenn der erste Teil bereits über eine langsame Verbindung gesendet werden kann, während der nächste Teil gerade generiert wird. Die Gesamtgröße der Daten ist aber erst nach Abschluss des letzten Stücks bekannt.

Mit der Option *createpvf* können Sie PDF-Daten im Arbeitsspeicher erzeugen und anschließend an PDFlib übergeben, ohne sie im Dateisystem zu speichern (siehe »Erzeugen von PDF-Ausgabe in einer virtuellen Datei«, Seite 64).

Aktive Schnittstelle zur PDF-Generierung im Arbeitsspeicher. Um PDF-Daten im Arbeitsspeicher zu generieren, übergeben Sie an *PDF_begin_document( )* einfach einen leeren Dateinamen und holen die Daten über *PDF_get_buffer( )* ab:

```
p.begin_document("", "");
... Dokument anlegen ...
p.end_document("");

buf = p.get_buffer();
```

```
... PDF-Daten aus dem Puffer verwenden ...
p.delete();
```

Hinweis *Die PDF-Daten im Puffer müssen als Binärdaten behandelt werden.*

Dies wird als »aktiver« Modus bezeichnet, da der Client aktiv entscheidet, wann der Pufferinhalt abgeholt werden soll. Der aktive Modus ist in allen unterstützten Sprachbindungen verfügbar.

Hinweis *C- und C++-Clients dürfen den zurückgegebenen Puffer nicht freigeben.*

Passive Schnittstelle zur PDF-Generierung im Arbeitsspeicher. Im »passiven« Modus, der nur in den Sprachbindungen für C und C++ verfügbar ist, installiert der Benutzer (via *PDF_open_document_callback( )*) eine Callback-Funktion, die von PDFlib zu nicht determinierten Zeitpunkten aufgerufen wird, sobald PDF-Daten zur Abholung bereitstehen. Um maximale Flexibilität zu gewährleisten, kann vom Client konfiguriert werden, in welcher Art das Leeren des Puffers *(Flushing)* erfolgt, das heißt, mit welchem Timing und welcher Puffergröße die PDF-Daten von der Bibliothek zum Client übertragen werden. Abhängig von der jeweiligen Umgebung mag es von Vorteil sein, das PDF-Dokument auf einmal, in mehreren Stücken oder in vielen kleinen Abschnitten abzuholen, damit PDFlib den internen Dokumentpuffer nicht vergrößern muss. Die gewünschte Flushing-Strategie kann mit der Option *flush* für *PDF_open_document_callback( )* eingestellt werden.

3.1.5 Maximalgröße von PDF-Dokumenten und andere Grenzwerte

Größe von PDF-Dokumenten. Auf den ersten Blick mag es nicht notwendig erscheinen, PDF-Dokumente im Bereich mehrerer Gigabytes anzulegen, doch gibt es Business-Anwendungen, die solche Dokumente erzeugen oder verarbeiten müssen, z.B. für eine große Anzahl von Rechnungen oder Kontoauszügen. Während bei PDFlib die Größe der erzeugten Dokumente nicht nach oben begrenzt ist, werden von der PDF-Referenz und einigen PDF-Standards mehrere Einschränkungen vorgegeben.

- Begrenzung der Dateigröße auf 10 GB: PDF-Dokumente wurden lange Zeit intern von Querverweis-Tabellen auf 10 Dezimalstellen und damit 10^{10}-1 Bytes begrenzt, was in etwa 9,3 GB entspricht. Allerdings kann diese Grenze mit komprimierten Objekt-Streams überschritten werden. Wenn Sie PDF-Ausgaben über 10 GB erstellen möchten, müssen Sie PDF 1.5 oder höher verwenden. Während komprimierte Objekt-Streams die Dateigröße ohnehin verringern, unterliegen die komprimierten Querverweis-Streams, die Teil der Implementierung von *objectstreams* sind, nicht länger der Grenze von 10 Dezimalstellen und ermöglichen damit die Erstellung von PDF-Dokumenten über 10 GB hinaus.
- Anzahl der Objekte: Während die Anzahl der Objekte in einem Dokument durch PDF generell nicht begrenzt wird, limitieren die Standards PDF/A, PDF/X-4 und PDF/X-5 die Zahl der indirekten Objekte in einem Dokument auf 8.388.607. Wenn ein Dokument mehr Objekte benötigt, löst PDFlib im PDF/A-, PDF/X-4- und PDF/X-5-Modus eine Exception aus. In anderen Modi können Dokumente mit einer höheren Objektzahl immer erzeugt werden. Diese Überprüfung kann mit der Dokumentoption *limitcheck=false* deaktiviert werden.
  Die Anzahl der PDF-Objekte hängt von der Komplexität der Seiteninhalte, der Anzahl der interaktive Elemente usw. ab. Da man bei typischen großvolumigen Doku-

menten mit einfachem Inhalt mit durchschnittlich etwa 4-10 Objekten pro Seite rechnet, können Dokumente mit etwa ein bis zwei Millionen Seiten ohne Überschreitung der vom Standard vorgegebenen Objektgrenze erzeugt werden.

PDF-Grenzwerte. PDFlib gibt für bestimmte Einheiten Grenzwerte vor, damit PDF-Ausgabe erzeugt werden kann, die kompatibel ist zu den Beschränkungen, die durch die PDF Reference, Acrobat oder einen PDF-Standard vorgegebenen werden. Diese Grenzwerte sind unten beschrieben.
Die folgenden Grenzwerte werden durch die entsprechende Modifizierung der Werte erzielt:

- Kleinste absolute Fließkommazahl in PDF: 0,000015. Zahlen mit einem kleineren absoluten Wert durch 0 ersetzt.
- (PDF 1.4, aber keine neueren PDF-Versionen) Größter absoluter Wert, der als Gleitpunktzahl im PDF ausgedrückt werden kann: 32767,0. Zahlen mit einem größeren Absolutwert werden durch die nächste Ganzzahl (Integer) ersetzt.

Das PDF-Format gibt gewisse Beschränkungen vor. Die Überschreitung eines der folgenden Grenzwerte löst eine Exception aus:

- Größter zulässiger Zahlenwert in PDF: 2.147.483.647
- Maximale Länge von Hypertext-Strings: 65535
- Maximale Länge von Textstrings auf einer Seite: 32.763 Bytes (also 16.381 Zeichen für CID-Fonts) bei *kerning=false* und *wordspacing=0*; ansonsten 4095 Zeichen
- Folgende Optionen sind auf maximal 8191 Listeneinträge beschränkt:
  views, namelist, polylinelist, fieldnamelist, itemnamelist, itemtextlist, children, group
- Maximale Anzahl der indirekten Objekte in PDF/A-1/2/3 und PDF/X-4/5: 8.388.607

3.1.6 Einsatz von PDFlib auf EBCDIC-Systemen

Die Operatoren und Strukturelemente des Dateiformats PDF basieren auf ASCII, das mit EBCDIC-Systemen wie i5/iSeries und zSeries mit den Betriebssystemen z/OS, USS oder MVS nicht gut funktioniert. (Anders dagegen zLinux, das auf ASCII basiert.) Deshalb steht eine spezielle Mainframe-Variante von PDFlib zur Verfügung, mit der sich auf ASCII basierende PDF-Operatoren und auf EBCDIC (oder anderen Zeichensätzen) basierende Textausgabe kombinieren lassen. Die EBCDIC-Variante von PDFlib ist für verschiedene Betriebssysteme und Rechnerarchitekturen verfügbar.

Um PDFlib-Funktionen auf EBCDIC-Systemen zu nutzen, müssen folgende Elemente im EBCDIC-Format übergeben werden (genauer gesagt, in Codepage 037 auf i5/iSeries und Codepage 1047 auf zSeries):

- PFA-Fontdateien, UPR-Konfigurationsdateien, AFM-Metrikdateien
- Encoding- und Codepage-Dateien
- String-Parameter für PDFlib-Funktionen
- Namen von Eingabe- und Ausgabedateien
- Umgebungsvariablen (sofern von der Laufzeitumgebung unterstützt)
- PDFlib-Fehlermeldungen werden auch im EBCDIC-Format erzeugt (außer in Java).

Um Eingabedateien im ASCII-Format (PFA, UPR, AFM, Encodings) zu verwenden, setzen Sie die Option *asciifile* auf *true* (Vorgabewert ist *false* auf zSeries und *true* auf i5/iSeries). PDFlib erwartet diese Dateien dann im ASCII-Format. String-Parameter müssen aber nach wie vor in EBCDIC kodiert sein.

Im Gegensatz dazu müssen folgende Elemente immer binär behandelt werden (das heißt, dass keinerlei Konvertierung durchgeführt werden darf):

- PDF-Eingabe- und -Ausgabe-Dateien
- PFB-Fontdateien und PFM-Metrikdateien
- TrueType- und OpenType-Fontdateien
- Rasterbilddateien und ICC-Profile

3.2 Seitenbeschreibungen

3.2.1 Koordinatensysteme

PDFlib arbeitet mit dem Standardkoordinatensystem von PDF, das seinen Ursprung in der linken unteren Ecke der Seite hat und auf der Einheit Punkt basiert:

```
1 pt = 1/72 Zoll = 25,4/72 mm = 0,3528 mm
```

Die erste Koordinate läuft nach rechts, die zweite nach oben. Das Standardkoordinatensystem kann von PDFlib-Clientprogrammen durch Rotieren, Skalieren, Verschieben oder Scheren modifiziert werden, so dass sich neue benutzerspezifische Koordinaten ergeben. Für diese Transformationen werden die Funktionen *PDF_rotate( )*, *PDF_scale( )*, *PDF_translate( )* und *PDF_skew( )* verwendet. Nach der Transformation müssen alle Koordinaten in Grafik- und Textfunktionen an das neue Koordinatensystem angepasst übergeben werden. Zu Beginn einer neuen Seite wird wieder auf das Standardkoordinatensystem umgestellt.

Verwendung metrischer Koordinaten. Durch eine Skalierung des Koordinatensystems können Sie problemlos auch metrische Koordinaten einsetzen. Der Skalierungsfaktor ergibt sich aus der Definition der Einheit Punkt oben:

```
p.scale(28.3465, 28.3465);
```

Nach diesem Aufruf interpretiert PDFlib alle Koordinaten (außer für Hypertext-Funktionen, siehe unten) in Zentimeter, da *72/2.54 = 28.3465*.

Ähnlich kann mit der Option *userunit* in *PDF_begin/end_page_ext( )* (PDF 1.6) ein Skalierungsfaktor für die ganze Seite übergeben werden. Beachten Sie dabei, dass sich *user units* nur auf die Anzeige der Seite in Acrobat auswirken, und nicht auf die Skalierung der Koordinaten in PDFlib.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/metric_topdown_coordinates`.

Koordinaten für interaktive Elemente. Bei interaktiven Funktionen, zum Beispiel bei den Rechtecken von Notizen, Verknüpfungen oder Dateianlagen, geht PDF immer von Koordinaten im Standardkoordinatensystem aus und nicht von einem (möglicherweise transformierten) Benutzerkoordinatensystem. Da dies recht mühselig werden kann, bietet PDFlib die automatische Umsetzung von Benutzerkoordinaten in das von PDF erwartete Format. Die automatische Konvertierung wird aktiviert, indem Sie die Option *usercoordinates* auf *true* setzen:

```
p.set_option("usercoordinates=true");
```

Da PDF bei Links und Formularfeldern nur Rechtecke erlaubt, deren Kanten parallel zum Seitenrand verlaufen, müssen die übergebenen Rechtecke modifiziert werden, wenn das Koordinatensystem durch Skalierung, Rotation, Verschiebung oder Scherung transformiert wurde. In diesem Fall berechnet PDFlib das kleinste umschließende Rechteck mit Kanten parallel zum Seitenrand, transformiert es in Standardkoordinaten und verwendet das Ergebnis statt der übergebenen Koordinaten.

Damit haben Sie die Möglichkeit, Seitenbeschreibungen sowie interaktive Elemente in einem einzigen Koordinatensystem zu definieren, sofern Sie die Option *usercoordinates* auf *true* gesetzt haben.

Koordinatenanzeige. Um PDFlib-Anwendern beim Arbeiten mit dem Koordinatensystem von PDF behilflich zu sein, enthält die PDFlib-Distribution die PDF-Datei *grid.pdf*, die die Koordinaten für gängige Seitenformate zeigt. Als nützliches Hilfsmittel zur PDFlib-Entwicklung können Sie sich die Seite mit dem für Sie interessanten Format auf eine durchsichtige Folie ausdrucken.

Seitenkoordinaten können Sie in Acrobat folgendermaßen anzeigen lassen:

- So können Sie Cursorkoordinaten anzeigen lassen:
  Acrobat X/XI: *Anzeige, Ein-/Ausblenden, Cursorkoordinaten*
  Acrobat 9*: Anzeige, Cursorkoordinaten*
- Die Koordinaten werden in der Einheit angezeigt, die aktuell in Acrobat ausgewählt ist. Um die angezeigte Maßeinheit in Acrobat 9/X/XI zu ändern, gehen sie folgendermaßen vor: Wählen Sie den Menübefehl *Bearbeiten, Voreinstellungen, [Allgemein...], Einheiten und Hilfslinien* und wählen Sie unter *Einheit, Seiten- und Linealeinheiten* nach Bedarf Punkt, Millimeter, Zoll, Pica oder Zentimeter.

Beachten Sie dabei, dass sich die angezeigten Koordinaten auf einen Ursprung in der linken oberen Ecke der Seite beziehen und nicht, wie bei PDF üblich, auf die linke untere Ecke. Für weitere Informationen zur Auswahl von Koordinatensystemen, die sich an die Koordinatenanzeige von Acrobat anpassen, siehe »Top-Down-Koordinaten«, Seite 76.

Drehen von Objekten. Es ist wichtig sich klarzumachen, dass Objekte nicht mehr verändert werden können, nachdem sie auf der Seite gezeichnet wurden. Es gibt zwar PDFlib-Funktionen zum Drehen, Verschieben, Skalieren und Scheren des Koordinatensystems, diese wirken sich aber nicht auf bereits auf der Seite vorhandene Objekte aus, sondern nur auf später gezeichnete.

Mit der Option *rotate* für die Funktionen *PDF_fit_textline( ), PDF_fit_textflow( ), PDF_fit_image( )* und *PDF_fit_pdi_page( )* ist es problemlos möglich, Text, Rasterbilder oder importierte PDF-Seiten zu drehen. Mit der Option *orientate* können Sie diese Objekte innerhalb der Fitbox um 90° oder ein Vielfaches davon drehen. Das folgende Beispiel generiert um 45° gedrehten Text:

```
p.fit_textline("gedrehter Text", 50.0, 700.0, "rotate=45");
```

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `text_output/rotated_text`.

Um Vektorgrafiken zu drehen, transformieren Sie das Koordinatensystem mit den Funktionen *PDF_translate( )* und *PDF_rotate( )*. Das folgende Beispiel erstellt ein gedrehtes Rechteck, dessen linke untere Ecke sich am Punkt *(200, 100)* befindet. Es verschiebt den Koordinatenursprung zur gewünschten Ecke des Rechtecks, dreht das Koordinatensystem und platziert das Rechteck am Punkt *(0, 0)*. Das Sichern und Wiederherstellen des Grafikkontexts mit *save/restore* ermöglicht es, nach der Ausgabe des gedrehten Rechtecks einfach weitere Objekte im ursprünglichen Koordinatensystem auszugeben:

```
p.save();
        p.translate(200, 100);              /* Ursprung zur Rechteck-Ecke verschieben */
        p.rotate(45.0);                     /* Koordinaten drehen */
        p.rect(0.0, 0.0, 75.0, 25.0);       /* gedrehtes Rechteck zeichnen */
```

```
        p.stroke();
p.restore();
```

Top-Down-Koordinaten. Im Gegensatz zum Bottom-Up-Koordinatensystem von PDF arbeiten manche grafischen Umgebungen mit Top-Down-Koordinaten. Wenn Sie diese Art vorziehen, können Sie sich das entsprechende Koordinatensystem problemlos mit den Transformationsfunktionen von PDFlib einrichten. Da die Transformationen aber auch die Textausgabe beeinflussen (Text steht ungewollt auf dem Kopf), sind weitere Aufrufe erforderlich, damit der Text nicht gespiegelt erscheint.

Um den Einsatz von Top-Down-Koordinaten zu erleichtern, unterstützt PDFlib einen besonderen Modus, in dem alle relevanten Koordinaten anders interpretiert werden: Die *topdown*-Einstellung ermöglicht dem PDFlib-Benutzer seine gewohnte Arbeitsweise auch im Top-Down-Koordinatensystem beizubehalten. Statt des PDF-Standardkoordinatensystems mit dem Ursprung *(0, 0)* in der linken unteren Ecke der Seite, in dem die *y*-Koordinaten von unten nach oben wachsen, wird ein Koordinatensystem verwendet, dessen Ursprung in der linken oberen Ecke liegt, wobei die *y*-Koordinaten nach unten hin größer werden. Das Top-Down-Koordinatensystem kann mit der Option *topdown* von *PDF_begin_page_ext( )* für die Seite aktiviert werden:

```
p.begin_page_ext(595.0, 842.0, "topdown");
```

Der Vollständigkeit halber folgt eine detaillierte Aufstellung aller Elemente, bei denen sich durch die Einrichtung eines Top-Down-Koordinatensystems Änderungen ergeben.

»Absolute« Koordinaten werden im Benutzerkoordinatensystem wie üblich und unverändert interpretiert:

- Alle Funktionsparameter, die in Funktionsbeschreibungen als »Koordinaten« bezeichnet werden. Zum Beispiel: *x, y* in *PDF_moveto( ); x, y* in *PDF_circle( ), x, y* (aber nicht *width* und *height!)* in *PDF_rect( ); llx, lly, urx, ury* in *PDF_create_annotation( )*).

»Relative« Koordinatenwerte werden durch interne Änderung an das Topdown-System angepasst:

- Text (mit positiver Schriftgröße) wird zum oberen Seitenrand hin ausgerichtet.
- Wenn im Handbuch von der »linken unteren« Ecke eines Rechtecks oder einer Box etc. die Rede ist, wird dies so interpretiert, wie Sie es auch auf der Seite sehen.
- Wird ein Drehwinkel festgelegt, bleibt das Rotationszentrum im Ursprung *(0, 0)* des Benutzerkoordinatensystems. Eine Drehung im Uhrzeigersinn wird immer noch wie eine solche wahrgenommen.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/metric_topdown_coordinates`.

3.2.2 Seitengröße

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `pagination/page_sizes`.

Gebräuchliche Seitenformate. Für die Optionen *width* und *height* in *PDF_begin/end_page_ext( )* können absolute Werte oder symbolische Seitenformatnamen verwendet werden. Letztere haben den Aufbau <*format*>*.width* und <*format*>*.height*, wobei <*format*> ein Standardseitenformat bezeichnet (zum Beispiel in Kleinbuchstaben: *a4.width)*.

Seitengröße. Im Gegensatz zu PDF und PDFlib, die keinerlei Einschränkungen hinsichtlich der verwendbaren Seitengröße aufweisen, unterliegt Acrobat einigen Beschränkungen. Andere PDF-Interpreter können aber durchaus mit größeren oder kleineren Formaten umgehen. Tabelle 3.2 zeigt die Einschränkungen bezüglich der Seitengröße in Acrobat. Ab PDF 1.6 kann mit der Option *userunit* in *PDF_begin/end_page_ext( )* ein Skalierungsfaktor eingestellt werden, der für die ganze Seite gilt.

Tabelle 3.2 Minimale und maximale Seitengröße von Acrobat

PDF-Viewer	Minimale Seitengröße	Maximale Seitengröße
Acrobat 5 und höher	*1/24" = 3 pt = 0.106 cm*	*200" = 14400 pt = 508 cm*
Acrobat 7 und höher mit der Option `userunit`	*3 user units*	*14400 user units* *Der größte Wert für* `userunit` *ist 75 000 für eine maximale Seitengröße von 14 400 * 75000 = 1 080 000 000 Punkte = 381 km*

Verschiedene Angaben für die Seitengröße. Während bei vielen PDFlib-Anwendungen nur die Höhe und Breite der Seite festgelegt wird, können in speziellen Anwendungen (insbesondere für die Druckvorstufe) zusätzliche Größenangaben wünschenswert sein. PDFlib unterstützt alle in PDF möglichen Größenangaben. PDFlib-Clients können folgende Größenangaben verwenden, die in unterschiedlichen Situationen sinnvoll sind:

- MediaBox: beschreibt die Größe des Ausgabemediums und entspricht unserer herkömmlichen Vorstellung von der Seitengröße.
- CropBox: gibt an, mit welcher Größe Acrobat die Seite anzeigt und druckt.
- TrimBox: beschreibt die Größe der (eventuell beschnittenen) Endseite.
- ArtBox: beschreibt den Teilbereich einer PDF-Seite, der für die Montage auf einer anderen Seite gedacht ist.
- BleedBox: gibt an, auf welche Größe die Seite in einer Produktionsumgebung beschnitten werden soll, und berücksichtigt eventuell benötigte Druckformaterweiterungen, die wegen Ungenauigkeiten im Produktionsprozess nötig sind.

PDFlib verwendet keinen dieser Werte intern, sondern schreibt sie lediglich in die Ausgabedatei. Standardmäßig wird eine MediaBox entsprechend der für die Seite festgelegten Höhe und Breite, aber keiner der anderen Einträge erzeugt. Das folgende Codefragment beginnt eine neue Seite und setzt die vier Werte der CropBox:

```
/* neue Seite mit selbst definierter CropBox beginnen */
p.begin_page_ext(595, 842, "cropbox={10 10 500 800}");
```

3.2.3 Direkte Pfade und Pfadobjekte

Ein Pfad ist eine Form, die aus einer beliebigen Anzahl von geraden Linien, Rechtecken, Kreisen, Bézierkurven oder Ellipsen besteht. Er kann aus mehreren getrennten Abschnitten, sogenannten Teilpfaden, bestehen. Auf einen Pfad können verschiedene Operationen angewandt werden:

- Beim *Stroking* (Zeichnen) wird der Pfad selbst gezeichnet, wobei die vom Client übergebenen Zeichenoptionen (etwa Farbe und Strichstärke) berücksichtigt werden.
- Beim *Filling* (Füllen) wird der gesamte vom Pfad eingeschlossene Bereich gezeichnet, wobei die vom Client übergebenen Füllungsparameter berücksichtigt werden.

- Beim *Clipping* (Beschneiden) wird der Abbildungsbereich für nachfolgende Zeichenoperationen verkleinert, indem der aktuelle Clipping-Bereich (standardmäßig unbegrenzt) durch die Schnittmenge aus dem aktuellen Clipping-Bereich und dem Pfad ersetzt wird.
- Wenn man den Pfad einfach beendet, ergibt das einen Pfad, der zwar in der PDF-Datei vorhanden, aber unsichtbar ist. Dies ist aber in der Regel nicht nützlich.

Direkte Pfade. Mit Pfadfunktionen wie *PDF_moveto( ), PDF_lineto( ), PDF_rect( )* können Sie einen direkten Pfad konstruieren, der sofort auf der aktuellen Seite oder einem anderen Content-Stream ausgegeben wird (z.B. ein Template oder eine Type 3-Glyph-Beschreibung). Der Pfad muss sofort nach der Konstruktion entweder mit *PDF_stroke( ), PDF_fill( ), PDF_clip( )* oder einer ähnlichen Funktion verarbeitet werden. Diese Funktionen verarbeiten und löschen den Pfad. Die einzige Möglichkeit, einen Pfad wiederzuverwenden, ist über *PDF_save( )* und *PDF_restore( )*.

Es führt zu einem Fehler, wenn Sie einen direkten Pfad konstruieren, ohne eine der obigen Operationen auf ihn anzuwenden. Durch das System der Geltungsbereiche (Scopes) stellt PDFlib sicher, dass sich Clients an diese Einschränkung halten. Wenn Sie die Darstellungseigenschaften eines Pfades ändern möchten (zum Beispiel Farbe oder Strichstärke), müssen Sie dies vor jeglichen Zeichenoperationen tun. Alle diesbezüglichen Regeln lassen sich zusammenfassen zu: »Innerhalb einer Pfaddefinition darf die Darstellung nicht geändert werden.«

Die bloße Konstruktion eines Pfads ist auf der Seite nicht wahrnehmbar; Sie müssen den Pfad explizit zeichnen oder füllen, um sichtbare Ergebnisse zu erzielen:

```
p.set_graphics_option("strokecolor=red");
p.moveto(100, 100);
p.lineto(200, 100);
p.stroke();
```

Die meisten Grafikfunktionen arbeiten mit dem Konzept eines aktuellen Punkts, den man sich wie die momentane Stiftposition beim Zeichnen vorstellen kann.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `graphics/starter_graphics`.

Pfadobjekte. Pfadobjekte sind eine komfortable und leistungsstarke Alternative zu direkten Pfaden. Pfadobjekte kapseln alle Zeichenschritte für die Konstruktion eines Pfads. Pfadobjekte können Sie mit *PDF_add_path_point( )* erzeugen oder aus einer Rasterbilddatei, die einen Beschneidungspfad enthält, extrahieren (siehe unten). *PDF_add_path_point( )* erleichtert die Pfadkonstruktion durch mehrere komfortable Optionen. Sobald ein Pfadobjekt erzeugt wurde, kann es für verschiedene Zwecke genutzt werden:

- Das Pfadobjekt kann zur Seitenbeschreibung mit *PDF_draw_path( )* benutzt werden, also gefüllt, zum Zeichnen von Pfaden oder als Beschneidungspfad verwendet werden.
- Pfadobjekte können als *Wrapping*-Form für Textflow verwendet werden: Text kann so formatiert werden, dass er innerhalb oder außerhalb einer beliebigen Form fließt (siehe Abschnitt 8.2.10, »Umfließen von Pfaden und Bildern«, Seite 240).
- Auch Text kann auf einem Pfad platziert werden, d.h. die Zeichen folgen dann den Linien und Kurven des Pfads (siehe Abschnitt 8.1.7, »Text auf einem Pfad«, Seite 221).
- Pfadobjekte können in Tabellenzellen platziert werden.

Anders als direkte Pfade können Pfadobjekte wiederverwendet werden, bis sie explizit mit *PDF_delete_path( )* gelöscht werden. Information über einen Pfad erhalten Sie mit *PDF_info_path( )*. Das folgende Codefragment erzeugt eine einfache Pfadform mit einem Kreis, durchzieht ihn an zwei Stellen auf der Seite mit Linien und löscht ihn schließlich:

```
path = p.add_path_point(  -1,   0, 100, "move", "");
path = p.add_path_point(path, 200, 100, "control", "");
path = p.add_path_point(path,   0, 100, "circular", "");

p.draw_path(path,   0,   0, "stroke");
p.draw_path(path, 400, 500, "stroke");
p.delete_path(path);
```

Statt Pfadobjekte mit einzelnen Zeichenschritten zu erzeugen, können Sie den Beschneidungspfad aus einer importierten Rastergrafik extrahieren:

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* Pfadobjekt aus dem Beschneidungspfad des Rasterbildes erzeugen */
path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
        throw new exception("Fegler: Beschneidungspfad nicht gefunden!");

p.draw_path(path, 0, 0, "stroke");
```

3.2.4 Templates (Form XObjects)

Templates (Form XObjects) in PDF. PDFlib unterstützt ein unter dem Fachbegriff »*Form XObject*« bekanntes PDF-Feature. Da diese Bezeichnung jedoch leicht mit interaktiven Formularen verwechselt werden kann, verwenden wir stattdessen die Bezeichnung *Templates*. Ein PDFlib-Template kann man sich als Puffer außerhalb der Seite vorstellen, in den Text, Vektorgrafiken und Rasterbilder umgelenkt werden (statt sich direkt auf der Seite zu befinden). Wenn das Template angelegt worden ist, kann es wie ein Rasterbild verwendet und beliebig oft auf beliebigen Seiten platziert werden. Wie Rasterbilder können Templates geometrisch transformiert, zum Beispiel skaliert oder geschert werden. Wird ein Template auf mehreren Seiten (oder auf einer Seite mehrmals) verwendet, werden die PDF-Operatoren, die für die Konstruktion des Templates zuständig sind, nur einmal in die PDF-Datei aufgenommen, was die Größe der Ausgabedatei entsprechend verringert. Templates empfehlen sich für Elemente, die wiederholt auf mehreren Seiten auftreten, zum Beispiel ein feststehender Hintergrund, ein Firmenlogo oder grafische Elemente, die aus CAD-Software oder Software für Landkarten stammen. Templates werden zudem für mehrfach platzierte Rasterbilder mit Beschneidungspfad empfohlen. Templates können Sie folgendermaßen erzeugen:

- direkt mit *PDF_begin_template_ext( );*
- indirekt aus Vektorgrafiken mit *PDF_load_graphics( )* und der Option *templateoptions*;
- indirekt aus Rasterbildern mit *PDF_load_image( )* und der Option *createtemplate*; ohne diese Option erzeugt *PDF_load_image( )* ein ähnliches PDF-Konstrukt namens Image XObject.

Hinweis *Mit PDF_open_pdi_page( ) importierte PDF-Seiten erzeugen auch PDF-Form-XObjects; diese werden aber mit PDI-Funktionen und nicht mit Template-Funktionen verarbeitet.*

Einsatz von Templates mit PDFlib. Ein Template kann mit der Funktion *PDF_fit_image( )* auf einer Seite oder auf einem anderen Template genauso wie ein Rasterbild platziert werden (siehe Abschnitt 7.4, »Platzieren von Bildern, Grafiken und importierten PDF-Seiten«, Seite 207). Das allgemeine Idiom für die Erzeugung und Verwendung von Templates sieht wie folgt aus:

```
/* Template definieren */
template = p.begin_template_ext(template_width, template_height, "");
...mit Text-, Vektorgrafik- und Rasterbildfunktionen zeichnen...
p.end_template_ext(0, 0);
...
p.begin_page(page_width, page_height);
/* Template verwenden */
p.fit_image(template, 0.0, 0.0, "");
...weitere Zeichenoperationen...
p.end_page();
...
p.close_image(template);
```

Auf einem Template können alle Text-, Grafik- und Farbfunktionen benutzt werden. Während der Konstruktion des Templates dürfen jedoch folgende Funktionen nicht verwendet werden:

- *PDF_begin_item( )* und die Option *tag* von verschiedenen Funktionen: In einem Template können keine Strukturelemente erzeugt werden.
- Alle interaktiven Funktionen: Diese müssen auf der Seite definiert werden, auf der sie im Dokument erscheinen sollen und können damit nicht als Bestandteil eines Templates generiert werden.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/repeated_contents`.

3.2.5 Seiten aus externen PDF-Dokumenten referenzieren

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `pdfx/starter_pdfx5g`.

PDF-Dokumente können Seiten aus externen Dokumenten referenzieren: die referenzierte Seite (skaliert oder gedreht) ist kein Bestandteil des Dokuments. Sie wird wie jeder andere Seiteninhalt dargestellt und gedruckt. Dies kann für die Referenzierung von wiederverwendeten Grafikinhalten (z.B. Logos oder Titelseiten) verwendet werden, ohne die entsprechenden PDF-Daten einbinden zu müssen. PDFlib unterstützt starke Verweise, d.h. Verweise, bei denen die referenzierte Seite durch interne Metadaten identifiziert wird. Wenn die referenzierte Seite nicht verfügbar ist oder nicht den erwarteten Metadaten entspricht, wird statt der referenzierten Seite ein Platzhalter angezeigt. Der Fachbegriff für dieses Feature ist *Reference XObject*.

Einsatz referenzierter Seiten in Acrobat. Referenzierte Seiten sind ein wesentlicher Bestandteil von PDF/X-5g und PDF/X-5pg. Das generierte (neue) Dokument nennt man Container-Dokument; das externe PDF-Dokument mit der referenzierten Seite nennt man Zieldatei.

Um referenzierte Seiten darzustellen, achten Sie darauf, Acrobat wie folgt zu konfigurieren:

- *Bearbeiten, Voreinstellungen, Allgemein..., Seitenanzeige, Referenzziele für XObjects anzeigen:* Wählen Sie *Immer* (die Einstellung *Nur PDF-X/5-kompatible* funktioniert aufgrund eines Fehlers in Acrobat X/XI nicht).
- *Bearbeiten, Voreinstellungen, Allgemein..., Seitenanzeige, Speicherort für referenzierte Dateien:* Geben Sie den Verzeichnisnamen an, in dem die Zieldatei liegt.
- *Bearbeiten, Voreinstellungen, Allgemein..., Sicherheit (Erweitert), Vertrauenswürdige Sites, Verzeichnispfad hinzufügen:* Wählen Sie den Verzeichnisnamen, in dem das Container-Dokument liegt. Dies müssen Sie unabhängig von der Einstellung *Erweiterte Sicherheit aktivieren* ausführen.

Wenn die folgenden Bedingungen erfüllt sind, wird statt des Platzhalters die Zieldatei dargestellt, deren Dateiname und Seitenzahl im Container-PDF festgelegt wurden:
- Das Container-Dokument ist laut Konfiguration in Acrobat vertrauenswürdig.
- Die Zieldatei liegt in dem festgelegten Verzeichnis.
- Die Zieldatei verlangt kein Kennwort und kann fehlerfrei geöffnet werden.
- Die Seitenzahl der referenzierten Seite ist im Container-Dokument vorhanden.
- Nur für PDF/X-5: Die ID und bestimmte XMP-Metadateneinträge in der Zieldatei stimmen mit den entsprechenden Einträgen im Container-Dokument überein.

Wenn eine oder mehrere dieser Bedingungen nicht zu erfüllt sind, wird ohne Anzeige einer Acrobat-Fehlermeldung statt der Zieldatei der Platzhalter angezeigt.

Platzhalter für die Zieldatei. PDFlib kann eins der folgenden Objekte als Platzhalter (Proxy) für die referenzierte Seite verwenden:
- Eine andere importierte PDF-Seite (z.B. eine vereinfachte Version der Zieldatei). Eine als Platzhalter verwendete PDF-Seite für ein externes Ziel muss die gleiche Geometrie wie die Zielseite aufweisen.
- SVG-Grafiken mit der Option *templateoptions:* in diesem Fall wird die Originalgröße der Grafik an Größe und Seitenverhältnis der Zielseite angepasst.
- Ein Template, z.B. eine einfache geometrische Form wie etwa ein durchgestrichenes Rechteck. Templates werden an Größe und Seitenverhältnis der Zielseite angepasst.

Mit dem folgenden Codefragment wird ein Template für einen Platzhalter mit einem Verweis auf eine externe Seite konstruiert:

```
proxy = p.begin_template_ext(0, 0,
        "reference={filename=target.pdf pagenumber=1 strongref=true}");
if (proxy == -1)
{
        /* Fehler */
}
...Template-Inhalt konstruieren...
p.end_template_ext(0, 0);
```

Der Platzhalter kann ganz normal auf der Seite platziert werden. Er enthält den Verweis auf die externe Seite.

3.3 Verschlüsseltes PDF

3.3.1 Sicherheitsfunktionen von PDF

PDF-Dokumente können mit Kennwortschutz versehen werden, welcher folgende Sicherheitsfunktionen bietet:

- Das Benutzerkennwort (auch Kennwort zum Öffnen des Dokuments) ist zum Öffnen der Datei erforderlich. Nur mit einem Benutzerkennwort versehene Dateien sind vor dem Knacken geschützt.
- Das Master-Kennwort (auch Berechtigungskennwort) ist zum Ändern von Sicherheitseinstellungen wie Berechtigungen, Benutzer- oder Master-Kennwort erforderlich. Dateien mit Benutzer- und Master-Kennwort können mit einem von beiden Kennwörtern geöffnet werden.
- Berechtigungen beschränken die mit dem PDF-Dokument erlaubten Aktionen, wie zum Beispiel das Drucken oder das Extrahieren von Text.
- Dateianlagen können separat verschlüsselt werden, ohne dass das Dokument selbst verschlüsselt ist.

Verwendet ein PDF-Dokument eine dieser Schutzfunktionen, wird es verschlüsselt. Zum Anzeigen oder Ändern der Sicherheitseinstellungen eines Dokuments klicken Sie in Acrobat auf *Datei, Eigenschaften, Sicherheit, Details anzeigen* bzw. *Einstellungen ändern*.

Verschlüsselungsalgorithmus und Schlüssellänge. Bei der PDF-Verschlüsselung werden folgende Verschlüsselungsalgorithmen verwendet:

- RC4, eine symmetrische Stromverschlüsselung (derselbe Algorithmus kann zum Verschlüsseln wie zum Entschlüsseln verwendet werden). RC4 ist ein proprietärer Algorithmus.
- AES (Advanced Encryption Standard) wurde im Standard FIPS-197 spezifiziert. AES ist eine moderne Block-Verschlüsselung, die in einer Vielzahl von Anwendungen verwendet wird.

Da die eigentlichen Schlüssel unhandliche binäre Sequenzen sind, werden sie von benutzerfreundlichen Kennwörtern abgeleitet, die aus normalem Text bestehen. Im Zuge der PDF- und Acrobat-Entwicklung wurden die PDF-Verschlüsselungsmethoden erweitert, um stärkere Algorithmen, längere Schlüssel und komplexere Kennwörter verwenden zu können. Tabelle 3.3 gibt eine Übersicht über die Merkmale von Verschlüsselung, Schlüsseln und Kennwörtern für alle PDF-Versionen.

Tabelle 3.3 Verschlüsselungsalgorithmen, Schlüssellängen und Kennwörter für verschiedene PDF-Versionen

PDF- und Acrobat-Version, pCOS Algorithmus-Nummer	***Verschlüsselungsalgorithmus und Schlüssellänge***	***Max. Länge des Kennworts und Kennwort-Encoding***
PDF 1.1 - 1.3 (Acrobat 2-4), Algorithmus 1	*RC4 40-Bit (schwacher Algorithmus, sollte nicht verwendet werden)*	*32 Zeichen (Latin-1)*
PDF 1.4 (Acrobat 5), Algorithmus 2	*RC4 128-Bit*	*32 Zeichen (Latin-1)*
PDF 1.5 (Acrobat 6), Algorithmus 3	*wie bei PDF 1.4, aber andere Anwendung der Verschlüsselungsmethode*	*32 Zeichen (Latin-1)*
PDF 1.6 (Acrobat 7) und PDF 1.7 = ISO 32000-1 (Acrobat 8), Algorithmus 4	*AES-128*	*32 Zeichen (Latin-1)*

Tabelle 3.3 Verschlüsselungsalgorithmen, Schlüssellängen und Kennwörter für verschiedene PDF-Versionen

PDF- und Acrobat-Version, pCOS Algorithmus-Nummer	Verschlüsselungsalgorithmus und Schlüssellänge	Max. Länge des Kennworts und Kennwort-Encoding
PDF 1.7ext3 (Acrobat 9), Algorithmus 9	AES-256 mit schwacher Handhabung von Kennwörtern (sollte nicht verwendet werden)	127 UTF-8-Bytes (Unicode)
PDF 1.7ext8 (Acrobat X/XI) und PDF 2.0 = ISO 32000-2, Algorithmus 11	AES-256 mit verbesserter Handhabung von Kennwörtern	127 UTF-8-Bytes (Unicode)

Bei der PDF-Verschlüsselung wird nicht direkt das Benutzer- oder Master-Kennwort zur Verschlüsselung des Dokumentinhalts verwendet, sondern aus dem Kennwort und anderen Daten einschließlich der Berechtigungen ein Chiffrierschlüssel errechnet. Die Länge des tatsächlich für die Verschlüsselung des Dokuments verwendeten Chiffrierschlüssels ist unabhängig von der Länge des Kennworts (siehe Tabelle 3.3.)

Kennwörter. Die PDF-Verschlüsselung arbeitet intern je nach PDF-Version mit 40-, 128- oder 256-Bit-Schlüsseln. Der binäre Chiffrierschlüssel wird aus dem vom Benutzer übergebenen Kennwort abgeleitet. Für das Kennwort bestehen Beschränkungen in der Länge und beim Encoding:

- Bis PDF 1.7 (ISO 32000-1) durften Kennwörter maximal 32 Zeichen lang sein und nur Zeichen vom Encoding Latin-1 enthalten.
- Mit PDF 1.7ext3 wurden Unicode-Zeichen eingeführt und die maximale Länge in der UTF-8-Darstellung des Kennworts auf 127 Bytes erhöht. Da UTF-8 Zeichen mit einer variablen Länge von 1-4 Bytes kodiert sind, ist die zulässige Anzahl von Unicode-Zeichen im Kennwort kleiner als 127, wenn es Nicht-ASCII-Zeichen enthält. Da japanische Zeichen in der Regel 3 Bytes in der UTF-8-Darstellung benötigen, können bis zu 42 japanische Zeichen für ein Kennwort verwendet werden.

Um Unklarheiten zu vermeiden, werden Unicode-Kennwörter durch einen Prozess namens *SASLprep* normiert (spezifiziert in RFC 4013, der auf *Stringprep* in RFC 3454 basiert). Dieser Prozess beseitigt Nicht-Textzeichen und normalisiert bestimmte Zeichenklassen (Nicht-ASCII-Leerzeichen werden z.B. auf ASCII-Leerzeichen U+0020 abgebildet). Das Kennwort wird in der Unicode-Normalisierungsform NFKC normiert. Spezielle Verarbeitungsschritte für bidirektionalen Text sollen Mehrdeutigkeiten vermeiden, wenn etwa links- und rechtsläufige Zeichen in einem Kennwort vermischt werden.

Die Stärke der PDF-Verschlüsselung hängt nicht nur von der Länge des Chiffrierschlüssels ab, sondern auch von der Länge und Qualität des Kennworts. Eigennamen oder echte Wörter sollten bekanntermaßen nicht als Kennwörter verwendet werden, da sie leicht zu erraten sind oder sich mit einem sogenannten Wörterbuchangriff systematisch ausprobieren lassen. Untersuchungen haben ergeben, dass für sehr viele Kennwörter einfach der Name des Partners oder Haustiers, der eigene Geburtstag, der Spitzname des Kindes usw. verwendet werden, die sich leicht erraten lassen.

Zugriffsberechtigungen. In PDF können verschiedene, individuelle Berechtigungen für Dokumente vergeben werden, die teilweise voneinander abhängig sind:

- *Drucken:* Wenn Drucken unzulässig ist, deaktiviert Acrobat die zugehörige Funktion. In Acrobat wird beim Druck zwischen hoher und niedriger Auflösung unterschieden. Drucken mit niedriger Auflösung erzeugt ein Bitmap-Bild auf der Seite, das sich nur für den persönlichen Gebrauch, aber nicht für hochwertige Reproduktion eignet. Be-

achten Sie, dass Bitmap-Druck nicht nur zu geringer Ausgabequalität führt, sondern das Drucken auch erheblich verlangsamt.

- *Ändern des Dokuments:* Wenn dies deaktiviert ist, ist jede Änderung am Dokument unzulässig. Extrahieren von Inhalt und Drucken ist erlaubt.
- *Kopieren und Seitenentnahme:* Wenn dies deaktiviert ist, ist das Auswählen und Kopieren von Seiteninhalten in die Zwischenablage zur Weiterverwendung unzulässig. Auch die Funktionen für Barrierefreiheit sind deaktiviert. Wenn Sie solche Dokumente in Acrobat durchsuchen möchten, müssen Sie die Voreinstellung *Nur zertifizierte Plugins* in den allgemeinen Voreinstellungen von Acrobat auswählen.
- *Kommentieren und Formularfelder:* Wenn dies deaktiviert ist, ist das Hinzufügen, Ändern und Löschen von Kommentaren und Formularfeldern unzulässig. Das Ausfüllen von Formularfeldern ist erlaubt.
- *Formularfelder ausfüllen oder unterschreiben:* Wenn dies deaktiviert ist, können Benutzer Formularfelder ausfüllen und unterschreiben, aber keine Formularfelder erstellen.
- *Kopieren von Inhalt für Barrierefreiheit zulässig:* Dokumentinhalte dürfen von Software für Barrierefreiheit (z.B. Screenreader) verwendet werden. Diese Einstellung wird ab PDF 2.0 nicht mehr unterstützt; Kopieren von Inhalt für Barrierefreiheit basiert dann auf den Einstellungen für *Kopieren und Seitenentnahme*.
- *Dokumentzusammenstellung:* Wenn dies deaktiviert ist, ist das Einfügen, Löschen oder Drehen von Seiten sowie das Erstellen von Lesezeichen und Miniaturansichten unzulässig.

Bei einer Zugriffsbeschränkung, zum Beispiel mit der Einstellung *Drucken unzulässig*, deaktiviert Acrobat die zugehörige Funktion. Dies gilt jedoch nicht unbedingt für PDF-Viewer oder andere Software von Drittherstellern. Es ist Sache der jeweiligen Tool-Entwickler, ob sie die definierten Zugriffsberechtigungen berücksichtigen oder nicht. Es gibt einige PDF-Tools, die Berechtigungseinstellungen vollständig ignorieren; außerdem können Zugriffsbeschränkungen mit kommerziellen Cracker-Tools außer Kraft gesetzt werden. Dies ist jedoch vom Schlüsselknacken zu unterscheiden; es ist einfach nicht möglich, das Drucken einer PDF-Datei zuverlässig zu unterbinden, wenn sie am Bildschirm anzeigbar sein soll. Dies wird sogar in der ISO-Norm 32000-1 dokumentiert:

»Sobald das Dokument geöffnet und erfolgreich entschlüsselt wurde, hat ein PDF-Anzeigeprogramm technisch gesehen Zugriff auf den gesamten Inhalt des Dokuments. Die PDF-Verschlüsselung enthält keine Mechanismen, die eine Durchsetzung der im Encryption-Dictionary festgelegten Dokumentberechtigungen erzwingen.«

Verschlüsselung einzelner Dokumentkomponenten. Standardmäßig umfasst die PDF-Verschlüsselung immer alle Komponenten eines Dokuments. Allerdings kann es manchmal sinnvoll sein, gezielt nur einzelne Komponenten eines Dokuments zu verschlüsseln:

- Ab PDF 1.5 (Acrobat 6) wurde die Funktion »Metadaten als Klartext« eingeführt. Damit können verschlüsselte Dokumente unverschlüsselte XMP-Metadaten enthalten. Dies erlaubt Suchmaschinen, auch aus verschlüsselten Dokumenten Metadaten auszulesen.
- Ab PDF 1.6 (Acrobat 7) können Dateianhänge auch in ansonsten ungeschützten Dokumenten verschlüsselt werden. Damit kann ein ungeschütztes Dokument als Container für vertrauliche Anhänge dienen.

Sicherheitseinstellungen. Beachten Sie, dass nur mit einem Benutzerkennwort versehene Dateien vor dem Knacken geschützt sind. Beachten Sie folgende Empfehlungen, da andernfalls die entsprechende Verschlüsselung schwach ist und geknackt werden könnte:

- Vermeiden Sie Kennwörter aus nur 1-6 Zeichen, da sie anfällig sind für Angriffe, bei denen alle möglichen Kennwörter systematisch durchprobiert werden (Brute-Force-Angriff).
- Vermeiden Sie echte Wörter, da sie anfällig sind für Angriffe, bei denen alle möglichen echten Wörter systematisch durchprobiert werden (Wörterbuchangriff). Kennwörter sollten nicht-alphanumerische Zeichen enthalten. Verwenden Sie nicht einfach den Namen Ihres Partners oder Haustiers, den eigenen Geburtstag oder andere leicht zu erratende Dinge.
- Verwenden Sie den modernen AES-Algorithmus statt des älteren RC4-Algorithmus.
- Vermeiden Sie Verschlüsselung mit AES-256 gemäß PDF 1.7ext3 (Acrobat 9), da diese aufgrund einer Schwäche in der Kennwortbehandlung anfällig für Brute-Force-Angriffe auf Kennwörter ist. Deshalb verwenden auch Acrobat X und PDFlib nie die Verschlüsselung von Acrobat 9 zum Schutz neuer Dokumente (sondern nur zur Entschlüsselung vorhandener Dokumente).

Zusammengefasst: Verwenden Sie am besten AES-256 gemäß PDF 1.7ext8/PDF 2.0 oder AES-128 gemäß PDF 1.6/1.7, je nachdem, ob Acrobat X oder höher verfügbar ist. Kennwörter sollten mindestens 6 Zeichen lang sein und auch nicht-alphanumerische Zeichen enthalten.

PDF-Schutz im Web. Wenn PDF-Dokumente über das Web bereitgestellt werden, können Benutzer mit dem Browser immer eine lokale Kopie des Dokuments erstellen. Es gibt keine Möglichkeit für PDF-Dokumente, dies zu verhindern.

3.3.2 Schützen von Dokumenten mit PDFlib

PDFlib kann bei der Erzeugung von PDF-Dokumenten die normalen Sicherheitsfunktionen nutzen. Um die Seiten von geschützten PDF-Dokumenten mit PDFlib+PDI oder PDFlib Personalization Server (PPS) zu importieren, ist das Master-Kennwort oder die Option *shrug* erforderlich. Das Abfragen von Dokument-Eigenschaften über die pCOS-Schnittstelle wird durch den pCOS-Modus geregelt. Zum Beispiel können XMP-Metadaten des Dokuments, Dokument-Infofelder, Lesezeichen und Anmerkungsinhalte ohne das Master-Kennwort abgerufen werden, wenn das Dokument kein Benutzerkennwort verlangt (oder nur das Benutzerkennwort übergeben wurde). Weiterführende Informationen hierzu finden Sie in der pCOS-Pfadreferenz.

Hinweis *Mit PDFlib-Produkten können Sie PDF-Dokumente nicht für den Reader-Modus freigeben (z.B. Anmerkungen im Adobe Reader erlauben).*

Verschlüsselungsalgorithmus und Schlüssellänge. Verschlüsselungsalgorithmus und Schlüssellänge, die zum Dokumentschutz verwendet werden, sind abhängig von der PDF-Version des erzeugten Dokuments, was wiederum von der Option *compatibility* von *PDF_begin_document( )* abhängt. Der Verschlüsselungsalgorithmus wird folgendermaßen gewählt:

- Bei PDF 1.4 und PDF 1.5 wird die entsprechende Variante von RC4 mit 128-Bit-Schlüsseln verwendet.

- Bei PDF 1.6, PDF 1.7 und PDF 1.7ext3 wird AES mit 128-Bit-Schlüsseln verwendet. Beachten Sie, dass AES-256 gemäß PDF 1.7ext3 (Acrobat 9) aufgrund bestehender Schwächen nie verwendet wird.[1]
- Bei PDF 1.7ext8 und PDF 2.0 wird AES-256 gemäß Acrobat X verwendet.

PDFlib verschlüsselt immer mit B 128 Bit langen Schlüsseln, da 40-Bit-Schlüssel bekanntlich nicht sicher sind. Bei PDFlib+PDI und PPS sind mit 40 Bit verschlüsselte Dokumente jedoch zulässig.

Kennwörter mit PDFlib setzen. Kennwörter können mit den Optionen *userpassword* und *masterpassword* in *PDF_begin_document( )* gesetzt werden. PDFlib verarbeitet die vom Client übergebenen Kennwörter für das erzeugte Dokument wie folgt:

- Wenn ein Benutzerkennwort oder Berechtigungen, aber kein Master-Kennwort übergeben wurde, könnte ein normaler Benutzer die Sicherheitseinstellungen ändern und damit jeglichen Schutz umgehen. Aus diesem Grund behandelt PDFlib diese Situation als Fehler.
- Sind Benutzer- und Master-Kennwort identisch, wäre keine Unterscheidung zwischen Benutzer und Eigentümer der Datei mehr möglich, was einem effektiven Schutz zuwiderläuft. PDFlib behandelt diese Situation als Fehler.
- Bei AES-256 sind Unicode-Kennwörter erlaubt. Bei allen älteren Verschlüsselungsalgorithmen müssen Kennwörter aus dem Latin-1-Zeichensatz bestehen. Wenn diese Regel nicht eingehalten ist, wird eine Exception ausgelöst.
- Bei AES-256 werden Kennwörter auf 127 UTF-8-Bytes und bei älteren Verschlüsselungsalgorithmen auf 32 Zeichen gekürzt.

Berechtigungen mit PDFlib setzen. Zugriffsbeschränkungen können mit der Option *permissions* in *PDF_begin_document( )* gesetzt werden. Sie enthält Schlüsselwörter für Zugriffsbeschränkungen. Zusammen mit der Option *permissions* müssen Sie auch die Option *masterpassword* verwenden, da Acrobat-Benutzer die festgelegten Berechtigungen sonst einfach wieder entfernen könnten. Standardmäßig sind alle Aktionen zulässig. Bei Festlegung einer Zugriffsbeschränkung wird die zugehörige Funktion in Acrobat deaktiviert. Zugriffsbeschränkungen können ohne Benutzerkennwort angewandt werden. Sie können mehrere durch Leerzeichen getrennte Schlüsselwörter angeben:

```
p.begin_document(filename, "masterpassword=abcd1234 permissions={noprint nocopy}");
```

Tabelle 3.4 zeigt alle für Zugriffsberechtigungen unterstützten Schlüsselwörter.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `general/permission_settings`.

1. Wenn Sie mit PDFlib 8 AES-256 gemäß PDF 1.7ext3 (Acrobat 9) verwendet haben und die Kennwörter Zeichen außerhalb von PDFDocEncoding enthielten, müssen Sie entweder `compatibility=1.7ext8` *setzen, um starke AES-Verschlüsselung mit Unicode-Zeichen zu erlauben oder die Kennwörter auf Zeichen in PDFDocEncoding beschränken.*

Tabelle 3.4 Schlüsselwörter für Berechtigungen für die Option `permissions` *in PDF_begin_document( )*

Schlüsselwort	Erklärung
noprint	*Acrobat verhindert das Drucken der Datei.*
nomodify	*Acrobat verhindert, dass Benutzer Formularfelder hinzufügen oder anderen Änderungen vornehmen.*
nocopy	*Acrobat verhindert, dass Text oder Grafik kopiert oder extrahiert wird; der barrierefreie Zugang (Accessibility) wird deaktiviert.*
noannots	*Acrobat verhindert das Hinzufügen oder Ändern von Kommentaren oder Formularfeldern.*
noforms	*(Impliziert* `noannots`*) Acrobat verhindert das Ausfüllen von Formularfeldern, auch wenn* `noannots` *nicht angegeben wurde.*
noaccessible	*(Abgekündigt in PDF 2.0) Acrobat verhindert die Extraktion von Text oder Grafik zum barrierefreien Zugang*
noassemble	*(Impliziert* `nomodify`*) Acrobat verhindert das Einfügen, Löschen oder Drehen von Seiten und die Erstellung von Lesezeichen und Miniaturansichten (Thumbnails), auch wenn* `nomodify` *nicht angegeben wurde.*
nohiresprint	*(PDF 1.4) Acrobat verhindert den Ausdruck mit hoher Auflösung. Wurde* `noprint` *nicht angegeben, wird der Ausdruck mit der Option »Als Bild drucken« durchgeführt und die Seite in niedriger Auflösung ausgegeben.*
plainmetadata	*(PDF 1.5) Die XMP-Metadaten des Dokuments bleiben auch bei verschlüsselten Dokumenten unverschlüsselt.*

Verschlüsselte Dateianlagen. Ab PDF 1.6 können Dateianlagen verschlüsselt werden, auch wenn das Dokument selbst nicht verschlüsselt ist. Dies lässt sich mit der Option *attachmentpassword* von *PDF_begin_document( )* erzielen.

3.4 Fortgeschrittener Umgang mit Farbe

In der PDFlib-Referenz finden Sie eine ausführliche Liste aller unterstützten Farbräume mit Erläuterungen. Dies umfasst auch die Farbräume RGB und CMYK. In diesem Abschnitt werden komplexe Farbthemen erläutert.

Cookbook *Codebeispiele zur Verwendung von Farbe finden Sie in der Kategorie* `color` *des PDFlib Cookbook. Eine Übersicht über den Einsatz von Farbräumen erhalten Sie im Cookbook-Topic* `color/starter_color`.

3.4.1 Color Management mit ICC-Profilen

PDFlib unterstützt Color Management mit ICC-Profilen und Rendering-Intents. ICC-Profile spielen eine entscheidende Rolle beim Color-Management sowie bei den meisten PDF-Standards wie PDF/X und PDF/A.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `color/iccprofile_to_image`.

ICC-Profile. Das International Color Consortium (ICC)[1] definierte ein Dateiformat zur Festlegung von Farbeigenschaften von Eingabe- und Ausgabegeräten. Diese sogenannten ICC-Farbprofile gelten als Industriestandard und werden von allen wichtigen Herstellern von System und Anwendungen für Color Management unterstützt. PDFlib unterstützt Color Management mit ICC-Profilen für die in Tabelle 3.5 aufgeführten Verwendungszwecke. Die Anzahl der Komponenten in einer Farbspezifikation wird durch Color Management nicht geändert (zum Beispiel von RGB nach CMYK).

Hinweis *ICC-Farbprofile für gängige Druckausgabebedingungen sowie Links zu anderen frei verfügbaren ICC-Profilen können von www.pdflib.com heruntergeladen werden.*

Tabelle 3.5 Anwendungsmöglichkeiten von ICC-Profilen

Verwendungszweck	***Entsprechende API-Funktionen und -Optionen***
Angabe von ICC-basierten Farbräumen für Text und Vektorgrafik auf der Seite	*PDF_get/set_option( ) mit* `iccprofilegray/rgb/cmyk` *und PDF_setcolor( ) mit* `colorspace=iccbasedgray/rgb/cmyk` *Farboptionen mit dem Schlüsselwort* `iccbased`
Anwendung eines ICC-Profils auf ein importiertes Bild	*PDF_load_image( ): Option* `iccprofile`
Angabe von Default-Farbräumen zur Abbildung von Graustufen-, RGB- oder CMYK-Daten auf ICC-basierte Farbräume	*PDF_begin_page_ext( ): Optionen* `defaultgray/rgb/cmyk`
Festlegung einer Druckausgabebedingung für PDF/X oder PDF/A mittels eines referenzierten oder eingebetteten ICC-Profils	*PDF_load_iccprofile( ): Option* `usage=outputintent`
Festlegung eines Misch-Farbraums für Transparenzberechnungen	*PDF_begin/end_page_ext( ), PDF_open_pdi_page( ), PDF_begin_template_ext( ) und PDF_load_graphics( ) mit* `templateoptions`*: Option* `transparencygroup`*, Unteroption* `colorspace`

1. Siehe www.color.org

Tabelle 3.5 Anwendungsmöglichkeiten von ICC-Profilen

Verwendungszweck	*Entsprechende API-Funktionen und -Optionen*
Verarbeitung von ICC-Profilen, die in importierten Bilddateien eingebettet sind	*PDF_load_image( ): Option* `honoriccprofile`
Abfrage von ICC-Profilen, die in importierten Bilddateien eingebettet sind	*PDF_info_image( ) mit* `keyword=iccprofile`
Abfrage der Anzahl von Farbkomponenten eines ICC-Profils	*PDF_get_option( ) mit Schlüsselwort* `icccomponents`

Geeignete ICC-Profile. Farbprofile müssen bestimmte Bedingungen hinsichtlich der ICC-Versionsnummer, der Geräteklasse und des Farbraums des Profils erfüllen. Die ICC-Versionsnummer ist wie folgt beschränkt:

- PDF 1.4: ICC 2.x
- PDF 1.5 oder höher: ICC 2.x oder 4.x

Tabelle 3.6 stellt je nach Verwendungszweck zusätzliche Anforderungen an die Geräteklasse und den Farbraum für ICC-Profile dar.

Tabelle 3.6 Geeignete ICC-Profile für unterschiedliche Verwendungszwecke

Verwendungszweck	*Geräteklasse*	*Farbraum*
Ausgabebedingung für PDF/X	*prtr*	*Gray, RGB, CMYK*
Ausgabebedingung für PDF/A	*prtr, mntr*	*Gray, RGB, CMYK*
Transparenzgruppe	*prtr, mntr, scnr, spac*	*Gray, RGB, CMYK*
alle anderen Verwendungszwecke	*prtr, mntr, scnr, spac*	*Gray, RGB, CMYK, Lab*

Suche nach ICC-Profilen. PDFlib sucht ICC-Profile in folgenden Schritten, wobei der an *PDF_load_iccprofile( )* übergebene Parameter *profilename* benutzt wird:

- Ist *profilename=sRGB*, verwendet PDFlib das interne sRGB-Profil und beendet die Suche.
- Es wird überprüft, ob sich in der Ressourcenkategorie *ICCProfile* eine Ressource namens *profilename* befindet. Ist dies der Fall, wird ihr Wert in den folgenden Schritten als Dateiname verwendet. Ist diese Ressource nicht vorhanden, wird *profilename* selbst als Dateiname verwendet.
- Anhand des im vorigen Schritt ermittelten Dateinamens wird auf der Festplatte nach einer Datei gesucht, wobei nacheinander folgende Kombinationen durchprobiert werden:

```
<dateiname>
<dateiname>.icc
<dateiname>.icm
<colordir>/<filename>              (nur unter Windows und OS X)
<colordir>/<filename>.icc          (nur unter Windows und OS X)
<colordir>/<filename>.icm          (nur unter Windows und OS X)
```

Unter Windows bezeichnet *colordir* das Verzeichnis, in dem das Betriebssystem gerätespezifische ICC-Profile ablegt (z.B. *C:\Windows\System32\spool\drivers)*. Unter OS X werden für *colordir* folgende Pfade ausprobiert:

```
/System/Library/ColorSync/Profiles
/Library/ColorSync/Profiles
/Network/Library/ColorSync/Profiles
~/Library/ColorSync/Profiles
```

Der Farbraum sRGB und das sRGB-Profil. PDFlib unterstützt den zum Industriestandard gewordenen RGB-Farbraum sRGB. Er wird von verschiedensten Software- und Hardware-Herstellern unterstützt und findet breiten Einsatz bei benutzerfreundlichem Color Management für Endbenutzer-Geräte wie digitale Kameras oder Bürogeräten wie Farbdruckern und Bildschirmen. PDFlib unterstützt den Farbraum sRGB und hält die erforderlichen ICC-Profildaten intern vor. Das sRGB-Profil steht damit ohne Zusatzkonfiguration immer zur Verfügung und muss deshalb nicht explizit vom Client konfiguriert werden. Es kann mit *PDF_load_iccprofile( )* und *profilename=sRGB* angefordert werden. Als komfortables Kürzel kann alternativ das Schlüsselwort *srgb* überall dort übergeben werden, wo ein mit *PDF_load_iccprofile( )* erzeugtes ICC-Profil-Handle erwartet wird.

Das sRGB-Profil hat die Geräteklasse *mntr* (Ausgabegerät), das heißt, es kann als Ausgabe-Intent für PDF/A, nicht aber für PDF/X genutzt werden.

Rasterbilder mit eingebettetem ICC-Profil. Rasterbilder können eingebettete ICC-Profile enthalten, die die Art der Farbwerte des Bildes beschreiben. Ein eingebettetes ICC-Profil kann zum Beispiel die Farbeigenschaften des Scanners beschreiben, mit dem die Bilddaten erzeugt wurden. PDFlib kann eingebettete ICC-Profile in den Bilddateiformaten PNG, JPEG und TIFF verarbeiten. Ist die Option *honoriccprofile* auf den Standardwert *true* gesetzt, wird das in ein Bild eingebettete ICC-Profil aus dem Bild extrahiert und so in die PDF-Ausgabe einbettet, dass es von Acrobat auf das Bild angewandt wird. Dieser Vorgang wird manchmal als »Tagging eines Bildes mit einem ICC-Profil« bezeichnet. Die Pixelwerte des Bildes werden von PDFlib dabei nicht verändert.

Mit dem Schlüsselwort *iccprofile* von *PDF_info_image( )* können Sie sich ein ICC-Profil-Handle für das in ein Rasterbild eingebettete ICC-Profil besorgen. Dies ist dann nützlich, wenn das selbe Profil auf mehrere Bilder angewandt werden muss.

Mit der Option *icccomponents* lässt sich die Anzahl der Farbkomponenten in einem unbekannten ICC-Profil abfragen.

Anwendung externer ICC-Profile auf Rasterbilder. Neben den in Bilder eingebetteten ICC-Profilen kann ein externes Profil auf ein Rasterbild angewandt werden, indem ein Profil-Handle sowie die Option *iccprofile* an *PDF_load_image( )* übergeben wird.

ICC-basierte Farbräume für Seitenbeschreibungen. Die Farbwerte für Text und Vektorgrafik können direkt in dem durch ein Profil definierten ICC-basierten Farbraum festgelegt werden. Dazu wird zunächst der Farbraum spezifiziert, indem das ICC-Profil-Handle einer der Optionen *iccprofilegray, iccprofilergb* oder *iccprofilecmyk* als Wert zugewiesen wird. Dann können ICC-basierte Farbwerte gemeinsam mit einem der Farbraum-Schlüsselwörtern *iccbasedgray*, *iccbasedrgb* und *iccbasedcmyk* an eine Farboption oder an *PDF_setcolor( )* übergeben werden:

```
p.set_option("errorpolicy=return");
icchandle = p.load_iccprofile("myCMYK", "usage=iccbased");
if (icchandle == -1)
{
        return;
```

```
}
p.set_graphics_option("fillcolor={iccbased=" + icchandle + " 0 1 0 0}");
```

Abbildung von Gerätefarben auf ICC-basierte Default-Farbräume. PDF bietet eine Methode, um in einer Seitenbeschreibung vorliegende geräteabhängige Graustufen-, RGB- und CMYK-Farben auf geräteunabhängige Farben abzubilden. Damit lassen sich Farbwerte, die sonst geräteabhängig wären, mit einer kolorimetrisch exakten Farbspezifikation versehen. Eine solche Abbildung von Farbwerten wird durch Bereitstellung der Farbraumdefinitionen *DefaultGray*, *DefaultRGB* und *DefaultCMYK* erreicht. In PDFlib wird dazu eine der Optionen *defaultgray, defaultrgb* oder *defaultcmyk* von *PDF_begin_page_ext( )* gesetzt und dieser als Wert ein ICC-Profil-Handle zugewiesen. Das folgende Beispiel definiert den Farbraum sRGB als RGB-Default-Farbraum für Text, Rasterbilder und Vektorgrafik:

```
p.begin_page_ext(595, 842, "defaultrgb=srgb");
```

Wenn der Default-Farbraum von einem externen ICC-Profil abgeleitet werden soll, muss zuerst ein Profil-Handle erzeugt werden:

```
/* ICC-Profil-Handle erzeugen*/
icchandle = p.load_iccprofile("myRGB", "usage=iccbased");
p.begin_page_ext(595, 842, "defaultrgb=" + icchandle);
```

Druckausgabebedingungen für PDF/X und PDF/A. Das Profil eines Ausgabegeräts (Drucker) kann zur Festlegung einer Druckausgabebedingung für PDF/X oder PDF/A verwendet werden. Dies wird mit *usage=outputintent* im Aufruf von *PDF_load_iccprofile( )* erzielt. Für PDF/A kann als Ausgabebedingung ein beliebiges Druck- oder Monitorprofil angegeben werden. Weitere Informationen finden Sie in Abschnitt 11.4, »PDF/X zur Druckproduktion«, Seite 337, und Abschnitt 11.3, »PDF/A zur Archivierung«, Seite 325.

Farbmischungsräume für Transparenzgruppen. Die Darstellung von transparenten Objekten auf einer Seite kann verbessert werden, indem ein geeigneter Farbraum für die Farbberechnung in der Unteroption *colorspace* der Option *transparencygroup* übergeben wird. Bei PDF/X-4/5 und PDF/A-2/3 kann dazu ein ICC-Profil-Handle angegeben werden.

Geräteunabhängige CIE L*a*b*-Farbe. Geräteunabhängige Farbwerte können im Farbraum CIE 1976 L*a*b* angegeben werden, indem der Farbraumname *lab* angegeben wird. Farben werden im L*a*b* Farbraum durch einen Helligkeitswert zwischen 0 und 100 sowie zwei Farbwerte zwischen -127 und 128 festgelegt. Die für den Farbraum *lab* verwendete Lichtquelle ist vom Typ D50 (Tageslicht 5000 K, 2°-Beobachter).

Rendering Intents. Dass PDFlib-Clients geräteunabhängige Farbwerte festlegen können, heißt nicht unbedingt, dass jedes Ausgabegerät in der Lage ist, die erforderlichen Farben auch exakt zu reproduzieren. Es sind deshalb einige Kompromisse hinsichtlich eines Prozesses namens Gamut Mapping (Farbraumkompression) notwendig. Dabei geht es darum, das Farbspektrum soweit zu verringern, dass es von einem Ausgabegerät reproduziert werden kann. Zur Steuerung dieses Prozesses kann der Rendering Intent verwendet werden. Rendering Intents können für einzelne Rasterbilder durch Übergabe der Option *renderingintent* an *PDF_load_image( )* festgelegt werden. Für Text und Vektorgrafik kann die Option *renderingintent* an *PDF_create_gstate( )* übergeben werden.

3.4.2 Pantone-, HKS- und benutzerdefinierte Schmuckfarben

PDFlib unterstützt Schmuckfarben *(spot color,* in der PDF-Fachsprache als *Separation*-Farbraum bezeichnet, obwohl der Ausdruck Separation im Allgemeinen auch für Prozessfarben verwendet wird). Diese können zur Ausgabe von benutzerdefinierten Farben verwendet werden, die über die mit Prozessfarben mischbaren Farben hinausgehen. Schmuckfarben sind durch ihren Namen definiert und treten in PDF immer gemeinsam mit einer Alternativfarbe auf, die der Schmuckfarbe möglichst ähnlich ist. Die Alternativfarbe wird in Acrobat zur Bildschirmanzeige und zur Ausgabe auf Geräten verwendet, die keine Schmuckfarben unterstützen (zum Beispiel Bürodrucker). Auf der Druckmaschine wird die geforderte Schmuckfarbe zusätzlich zu den im Dokument benutzten Prozessfarben angewandt.

In PDFlib sind verschiedene Schmuckfarbbibliotheken integriert. Außerdem werden benutzerdefinierte Schmuckfarben unterstützt. Wird ein Schmuckfarbname mit *PDF_makespotcolor( )* angefordert, überprüft PDFlib zunächst, ob diese Schmuckfarbe in einer der integrierten Bibliotheken verzeichnet ist. Ist dies der Fall, verwendet PDFlib integrierte Werte für die Alternativfarbe. Anderenfalls wird von einer benutzerdefinierten Schmuckfarbe ausgegangen und der Client muss geeignete Werte für die Alternativfarbe liefern (über die aktuelle Farbe). Schmuckfarben können mit einem Farbauftrag, das heißt mit einem Prozentwert zwischen 0 und 1 versehen werden.

Standardmäßig können integrierte Schmuckfarben nicht mit benutzerdefinierten Alternativfarben umdefiniert werden. Dieses Verhalten lässt sich mit der Option *spotcolorlookup* ändern, etwa um die Kompatibilität zu älteren Anwendungen zu gewährleisten, die andere Farbdefinitionen verwenden, oder um Workflows zu unterstützen, die mit den von PDFlib erzeugten Lab-Alternativfarben für Pantone-Farben nicht zurechtkommen.

PDFlib generiert für integrierte Schmuckfarben automatisch geeignete Alternativfarben, sofern eine der PDF/X- oder PDF/A-Kompatibilitätsstufen gewählt wurde (siehe Abschnitt 11.4, »PDF/X zur Druckproduktion«, Seite 337). Bei benutzerdefinierten Schmuckfarben liegt es in der Verantwortung des Benutzers, zur gewählten PDF/X- oder PDF/A-Kompatibilitätsstufe passende Alternativfarben bereitzustellen.

Hinweis *Die Daten für integrierte Pantone®- und HKS®-Schmuckfarben und die zugehörigen Markenzeichen wurden von den jeweiligen Inhabern für den Einsatz in der PDFlib-Software lizenziert.*

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `color/spot_color`.

Pantone®-Farben. Pantone-Farben sind weit verbreitet und weltweit im Einsatz. PDFlib bietet volle Unterstützung für das Pantone Matching System® (insgesamt etwa 26 000 Farbtöne). Alle Farbnamen aus den in Tabelle 3.7 angeführten digitalen Farbbibliotheken stehen zur Verfügung. Inhaber von PDFlib-Lizenzen erhalten von unserem Support auf Anfrage eine Textdatei mit einer Liste aller Pantone-Schmuckfarbnamen.

Bei Schmuckfarbnamen wird zwischen Groß- und Kleinschreibung unterschieden; schreiben Sie die Namen deshalb wie in den Beispielen in Großbuchstaben. Es werden auch die alten Namenspräfixe CV, CVV, CVU, CVC und CVP akzeptiert und in die entsprechenden neuen Farbnamen umgewandelt, es sei denn die Option *preserveoldpantonenames* ist auf *true* gesetzt. Wie die Beispiele zeigen, beginnt

der Farbname immer mit dem Präfix *PANTONE*. Generell müssen Pantone-Farbnamen nach folgendem Schema aufgebaut sein:

```
PANTONE <Id> <Papiersorte>
```

Dabei bezeichnet *<Id>* die Farbe (zum Beispiel 185) und *<Papiersorte>* die englische Abkürzung für die verwendete Papiersorte (zum Beispiel *C* für *coated*, das heißt beschichtet). Die Namensbestandteile werden durch jeweils ein einzelnes Leerzeichen getrennt. Wenn Sie eine Schmuckfarbe abrufen, deren Name zwar mit dem Präfix *PANTONE* beginnt, aber keine existierende Pantone-Farbe bezeichnet, wird eine Warnung protokolliert. Das folgende Codefragment zeigt den Einsatz einer Pantone-Farbe mit einem Farbauftrag von 70 Prozent:

```
spot = p.makespotcolor("PANTONE 281 U");
p.setcolor("fill", "spot", spot, 0.7, 0, 0);
```

Hinweis *Die hier angezeigten PANTONE®-Farben stimmen nicht unbedingt mit den PANTONE-Standards überein. Die genaue Farbe können Sie in den Pantone-Farbtafeln nachschlagen. Pantone® und andere Warenzeichen von Pantone, Inc. sind Eigentum von Pantone, Inc. © Pantone, Inc., 2003.*

Tabelle 3.7 In PDFlib integrierte PANTONE-Schmuckfarbbibliotheken

Farbbibliothek	Beispielfarbname	Anmerkungen
PANTONE solid coated	PANTONE 185 C	
PANTONE+ Solid Coated-336 New	PANTONE 2071 C	336 Farben; verfügbar seit 2012
PANTONE solid uncoated	PANTONE 185 U	
PANTONE+ Solid Uncoated-336 New	PANTONE 2071 U	336 Farben; verfügbar seit 2012
PANTONE solid matte	PANTONE 185 M	
PANTONE process coated	PANTONE DS 35-1 C	
PANTONE process uncoated	PANTONE DS 35-1 U	
PANTONE process coated EURO	PANTONE DE 35-1 C	seit 2006 verfügbar
PANTONE process uncoated EURO	PANTONE DE 35-1 U	seit Mai 2006 verfügbar
PANTONE pastel coated	PANTONE 9461 C	seit 2006 mit zusätzlichen Farben
PANTONE pastel uncoated	PANTONE 9461 U	seit 2006 mit zusätzlichen Farben
PANTONE metallic coated	PANTONE 871 C	seit 2006 mit zusätzlichen Farben
PANTONE color bridge CMYK PC	PANTONE 185 PC	ersetzt PANTONE solid to process coated
PANTONE color bridge CMYK EURO	PANTONE 185 EC	ersetzt PANTONE solid to process coated EURO
PANTONE color bridge uncoated	PANTONE 185 UP	seit Juli 2006 verfügbar
PANTONE hexachrome coated	PANTONE H 305-1 C	nicht empfohlen; wird nicht fortgeführt
PANTONE hexachrome uncoated	PANTONE H 305-1 U	nicht empfohlen; wird nicht fortgeführt
PANTONE solid in hexachrome coated	PANTONE 185 HC	

Tabelle 3.7 In PDFlib integrierte PANTONE-Schmuckfarbbibliotheken

Farbbibliothek	Beispielfarbname	Anmerkungen
PANTONE solid to process coated	PANTONE 185 PC	ersetzt durch PANTONE color bridge CMYK PC
PANTONE solid to process coated EURO	PANTONE 185 EC	ersetzt durch PANTONE color bridge CMYK EURO
PANTONE Goe coated	PANTONE 42-1-1 C	2058 Farben; verfügbar seit 2008; nicht empfohlen
PANTONE Goe uncoated	PANTONE 42-1-1 U	2058 Farben; verfügbar seit 2008; nicht empfohlen

HKS®-Farben. Das HKS-Farbsystem ist in Deutschland und anderen europäischen Ländern weit verbreitet. PDFlib bietet volle Unterstützung für HKS-Farben. Sie können alle Farbnamen aus den folgenden digitalen Farbbibliotheken *(Farbfächer)* verwenden (Beispielnamen werden in Klammern angegeben):

- HKS K *(Kunstdruckpapier)*, 88 Farben *(HKS 43 K)*
- HKS N *(Naturpapier)*, 86 Farben *(HKS 43 N)*
- HKS E *(Endlospapier)* beschichtet, 88 Farben *(HKS 43 E)*
- HKS Z *(Zeitungspapier)*, 50 Farben *(HKS 43 Z)*

Inhaber von PDFlib-Lizenzen erhalten von unserem Support auf Anfrage eine Textdatei mit einer Liste aller HKS-Schmuckfarbnamen.

Bei Schmuckfarbnamen wird zwischen Groß- und Kleinschreibung unterschieden; schreiben Sie die Namen deshalb wie in den Beispielen in Großbuchstaben. Wie die Beispiele zeigen, beginnt der Farbname immer mit dem Präfix HKS. Generell müssen HKS-Farbnamen nach folgendem Schema aufgebaut sein:

```
HKS <Id> <Papiersorte>
```

Dabei bezeichnet *<Id>* die Farbe (zum Beispiel 43) und *<Papiersorte>* ist die Abkürzung für die verwendete Papiersorte (zum Beispiel *N* für Naturpapier). Die Namensbestandteile *HKS*, *<Id>* und *<Papiersorte>* werden jeweils durch ein einzelnes Leerzeichen getrennt. Wenn Sie eine Schmuckfarbe abrufen, deren Name zwar mit dem Präfix *HKS* beginnt, aber keine existierende HKS-Farbe bezeichnet, wird eine Warnung protokolliert. Das folgende Codefragment zeigt den Einsatz einer HKS-Farbe mit einem Farbauftrag von 70 Prozent:

```
spot = p.makespotcolor("HKS 38 E");
p.setcolor("fill", "spot", spot, 0.7, 0, 0);
```

Benutzerdefinierte Schmuckfarben. Neben den oben beschriebenen integrierten Schmuckfarben unterstützt PDFlib benutzerdefinierte Schmuckfarben. Diese können mit einem beliebigen Namen (der sich jedoch von den integrierten Namen unterscheiden muss) sowie einer Alternativfarbe versehen sein, die zur Bildschirmausgabe sowie für einfache Druckausgabe, nicht jedoch für hochqualitative Farbseparationen verwendet wird. Der Client ist dafür verantwortlich, geeignete Alternativfarben für benutzerdefinierte Schmuckfarben anzugeben.

Schmuckfarben können mit den Optionen *fillcolor/strokecolor* für Text- und Grafikdarstellung und anderen farbbezogenen Optionen gesetzt werden. Der Alternativfarbe kann direkt in der Definition für die Schmuckfarbe gesetzt werde:

```
fillcolor={spotname={CompanyColor} 1.0 {cmyk 0.2 1.0 0.2 0}}
```

Alternativ können Schmuckfarben mit *PDF_setcolor( )* definiert werden. In diesem Fall wird die aktuelle Füllfarbe als Alternativfarbe verwendet. Außer einem zusätzlichen Aufruf zur Festlegung der Alternativfarbe wird eine benutzerdefinierte Schmuckfarbe im Prinzip wie eine integrierte Schmuckfarbe definiert und eingesetzt:

```
p.setcolor("fill", "cmyk", 0.2, 1.0, 0.2, 0);       /* CMYK-Alternativweerte setzen */
spot = p.makespotcolor("CompanyColor");             /* Schmuckfarbe ableiten */
p.setcolor("fill", "spot", spot, 1, 0, 0);          /* Schmuckfarbe setzen */
```

3.4.3 Füllmuster und Farbverläufe

Alternativ zu flächigen Farben können Objekte mit besonderen Farbarten, und zwar Füllmustern oder Farbverläufen gezeichnet oder gefüllt werden.

Füllmuster. Ein Füllmuster (Pattern) ist durch eine beliebige Anzahl von Operationen zum Auftragen von Farbe definiert, die in einer einzigen Einheit zusammengefasst werden. Diese Einheit kann auf ein beliebiges anderes Objekt angewandt werden, indem sie wiederholt (oder gekachelt) über den gesamten zu füllenden Bereich oder zu zeichnenden Pfad aufgetragen wird. Die Arbeit mit Füllmustern umfasst folgende Schritte:

- Zuerst wird zwischen *PDF_begin_pattern( )* und *PDF_end_pattern( )* das Füllmuster definiert. Dazu können die meisten Grafikoperatoren herangezogen werden.
- Mit dem von *PDF_begin_pattern( )* zurückgegebenen Füllmuster-Handle kann das Füllmuster mit *PDF_setcolor( )* oder der Options *fill/strokecolor* in *PDF_set_graphics_option( )*zur aktuellen Farbe gemacht werden.

Abhängig vom Parameter *painttype* von *PDF_begin_pattern( )* wird die Farbe des Füllmusters festgelegt. Ist *painttype* gleich 1, muss die Füllmusterdefinition eine eigene Farbspezifikation enthalten und sieht damit immer gleich aus; ist *painttype* gleich 2, darf die Füllmusterdefinition keine eigene Farbspezifikation enthalten. Das Füllmuster wird dann in der jeweils aktuellen Füll- oder Zeichenfarbe aufgetragen.

Hinweis *Füllmuster können auch auf Basis eines Farbverlaufs definiert werden (siehe unten).*

Cookbook *Vollständige Codebeispiele hierzu finden Sie in den Cookbook-Topics* `graphics/fill_pattern` *und* `images/tiling_pattern`.

Farbverläufe. Farbverläufe *(Blends* oder *Smooth Shadings)* bilden einen kontinuierlichen Übergang zwischen zwei Farben. Die beiden Farben müssen im selben Farbraum definiert sein. PDFlib unterstützt zwei geometrische Formen für Farbverläufe:

- axiale Verläufe verlaufen entlang einer Geraden;
- radiale Verläufe verlaufen zwischen zwei Kreisen.

Farbverläufe werden als Übergang zwischen zwei Farben definiert. Die erste Farbe entspricht immer der aktuellen Füllfarbe; die zweite Farbe wird in den Parametern *c1, c2, c3* und *c4* von *PDF_shading( )* übergeben. Diese numerischen Werte werden im Farbraum der ersten Farbe gemäß der Beschreibung von *PDF_setcolor( )* interpretiert.

PDF_shading( ) gibt ein Handle auf ein Farbverlaufsobjekt zurück, das zu zwei Zwecken verwendet werden kann:

- Füllen einer Fläche mit *PDF_shfill( )*. Dieses Verfahren kann verwendet werden, wenn die Geometrie des zu füllenden Objekts der Geometrie des Farbverlaufs entspricht.

Trotz ihres Namens füllt diese Funktion nicht nur das Innere des Objekts, sondern wirkt sich auch auf den Außenbereich aus. Dieses Verhalten kann mit *PDF_clip( )* geändert werden.

- Definition eines Pattern für einen Farbverlauf zum Füllen komplexerer Objekte. Dazu muss zunächst mit *PDF_shading_pattern( )* ein Pattern erzeugt werden, das auf dem Farbverlauf basiert. Mit diesem Pattern können dann beliebige Objekte gefüllt oder gezeichnet werden.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `color/color_gradient`.

4 Unicode und andere Encodings

Dieses Kapitel enthält grundlegende Informationen zu Unicode und anderen Encoding-Konzepten. Die Textverarbeitung in PDFlib stützt sich stark auf den Unicode-Standard, unterstützt aber auch ältere Encodings.

4.1 Wichtige Unicode-Konzepte

Zeichen und Glyphen. Beim Arbeiten mit Text ist es wichtig, klar zwischen folgenden Konzepten unterscheiden:

- *Zeichen* sind die kleinsten Einheiten, die in einer Sprache Informationen übermitteln, wie zum Beispiel die Buchstaben im lateinischen Alphabet, chinesische Bildzeichen oder japanische Silben. Zeichen haben eine bestimmte Bedeutung: Sie stellen semantische Einheiten dar.
- *Glyphen* sind unterschiedliche grafische Varianten, die ein oder mehrere Zeichen darstellen. Glyphen haben ein bestimmtes Aussehen: Sie stellen grafische Einheiten dar.

Es gibt keine Eins-zu-Eins-Beziehung zwischen Zeichen und Glyphen. So ist eine Ligatur zum Beispiel eine einzelne Glyphe, die mindestens zwei Zeichen zugeordnet ist. Genauso kann eine Glyphe je nach Kontext zur Darstellung verschiedener Zeichen verwendet werden (verschiedene Zeichen können gleich aussehen, siehe Abbildung 4.1).

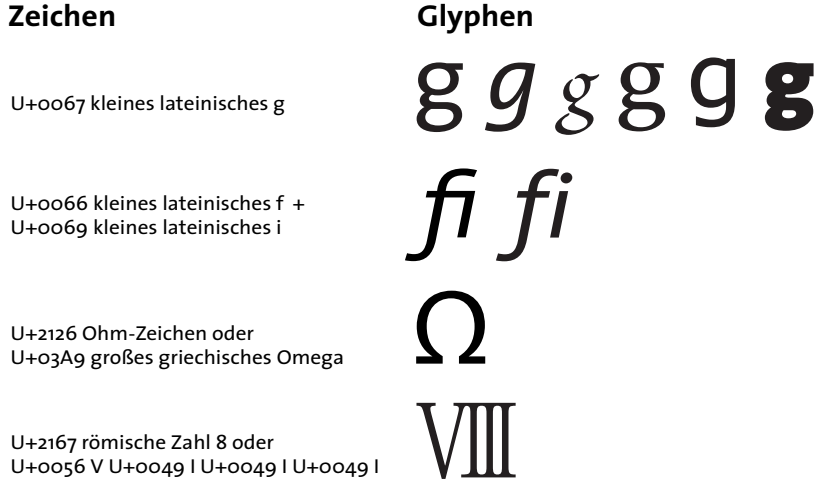


Abb. 4.1 Beziehung zwischen Glyphen und Zeichen

BMP und PUA. Die folgenden Begriffe werden häufig in Unicode-basierten Umgebungen verwendet:

- Die *Basic Multilingual Plane (BMP)* umfasst die Codepunkte im Unicode-Bereich U+0000...U+FFFF. Der Unicode-Standard enthält viele weitere Codepunkte in den ergänzenden Ebenen, d.h. im Bereich U+10000...U+10FFFF.
- Die *Private Use Area (PUA)* besteht aus mehreren Unicode-Bereichen, die für den privaten Gebrauch reserviert sind. PUA-Codepunkte können nicht für den allgemeinen Austausch verwendet werden, da der Unicode-Standard keine Zeichen in diesem Bereich festgelegt. Die Basic Multilingual Plane enthält den PUA-Bereich U+E000...U+F8FF. Ebene fünfzehn (U+F0000...U+FFFFD) und Ebene sechzehn (U+100000...U+10FFFD) sind vollständig dem privaten Gebrauch vorbehalten.

Unicode-Encoding-Formen (UTF-Formate). Der Unicode-Standard ordnet jedem Zeichen eine Nummer (Codepunkt) zu. Um diese Nummern verwenden zu können, müssen sie sich in irgendeiner Form darstellen lassen. Im Unicode-Standard wird dazu eine *Encoding-Form* (früher: Transformationsformat) herangezogen, was nicht mit Font-Encodings zu verwechseln ist. Unicode definiert folgende Encoding-Formen:

- *UTF-8:* Format variabler Breite, in dem die Codepunkte durch 1-4 Bytes repräsentiert werden. ASCII-Zeichen im Bereich U+0000...U+007F werden durch ein einzelnes Byte im Bereich 00...7F dargestellt. Latin-1-Zeichen im Bereich U+00A0...U+00FF werden durch zwei Bytes dargestellt, wobei das erste Byte immer 0xC2 oder 0xC3 ist (diese Werte repräsentieren *Â* und *Ã* in Latin-1).
- *UTF-16:* Codepunkte in der Basic Multilingual Plane (BMP) werden durch einen einzelnen 16-Bit-Wert dargestellt. Codepunkte in den Supplementär-Ebenen, d.h. im Bereich U+10000...U+10FFFF, werden durch zwei 16-Bit-Werte dargestellt. Solche Paare werden Surrogatpaare genannt. Ein Surrogatpaar besteht aus einem höherwertigen Surrogatwert im Bereich D800...DBFF und einem niederwertigen Surrogatwert im Bereich DC00...DFFF. Beide können nur als Bestandteil eines Surrogatpaares und in keinem anderen Zusammenhang auftreten.
- *UTF-32:* Jeder Codepunkt wird durch einen 32-Bit-Wert dargestellt.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `text_output/process_utf8`.

Unicode-Encodings und der Byte Order Mark (BOM). Computerarchitekturen unterscheiden sich in der Bytereihenfolge, d.h. ob die Bytes eines (16 oder 32 Bit großen) Werts mit dem höchstwertigen Byte (Big-Endian) oder niederstwertigen Byte (Little-Endian) zuerst gespeichert werden. PowerPC ist zum Beispiel eine Big-Endian-Architektur, während die x86-Architektur die Bytereihenfolge Little-Endian benutzt. Da UTF-8- und UTF-16-Werte größer als ein Byte sind, muss hier die Bytereihenfolge beachtet werden. Ein Zeichensatz (im Unterschied zur Encoding-Form oben) legt Encoding-Form plus Byte-Reihenfolge fest. UTF-16BE steht beispielsweise für UTF-16 mit Bytereihenfolge Big-Endian. Steht die Bytereihenfolge nicht vorab fest, kann sie mit dem Codepunkt U+FEFF, dem so genannten Byte Order Mark (BOM), festgelegt werden. Der BOM ist in UTF-8 nicht unbedingt erforderlich, kann aber optional vorhanden sein, um eine Bytefolge als UTF-8 zu kennzeichnen. Tabelle 4.1 zeigt die Darstellung des BOM für verschiedene Encoding-Formen.

Tabelle 4.1 Byte Order Marks für verschiedene Unicode-Encoding-Formen

Encoding-Form	***Byte Order Mark (Hex)***	***grafische Darstellung in WinAnsi[1]***
UTF-8	*EF BB BF*	ï»¿
UTF-16 Big-Endian	*FE FF*	þÿ
UTF-16 Little-Endian	*FF FE*	ÿþ
UTF-32 Big-Endian	*00 00 FE FF*	■ ■ þÿ
UTF-32 Little-Endian	*FF FE 00 00*	ÿþ■ ■

1. Das schwarze Quadrat ■ steht für ein Nullbyte.

4.2 Unicode-fähige Sprachbindungen

Einige Merkmale des PDFlib API variieren je nachdem, ob die verwendete Sprach-Bindung Unicode-fähig ist. Dieses Konzept wird in diesem und im nächsten Abschnitt erläutert.

4.2.1 Sprachbindungen mit internen Unicode-Strings

Wenn eine Programmiersprache oder Entwicklungsumgebung intern Unicode-Strings unterstützt, nennen wir die Sprachbindung Unicode-fähig. Folgende PDFlib-Sprachbindungen sind Unicode-fähig:

- C++
- COM
- .NET
- Java
- Objective-C
- Python
- REALbasic
- RPG

In diesen Umgebungen ist die Stringbehandlung einfach: Alle Strings werden an den PDFlib-Kern automatisch als Unicode-Strings im Format UTF-16 übergeben. Vom Client übergebene Unicode-Strings werden von den Sprachwrappern korrekt verarbeitet, die automatisch auch bestimmte PDFlib-Optionen setzen. Dies hat folgende Konsequenzen:

- Alle vom Client übergebene Strings kommen in PDFlib immer im Format *utf16* und im Encoding *unicode* an.
- Die Unterscheidung zwischen den Stringtypen in den API-Beschreibungen ist nicht relevant (Content-Strings, Hypertext-Strings und Name-Strings). Die Optionen *textformat*, *hypertextformat* und *hypertextencoding* sind nicht erforderlich und nicht erlaubt. Die Textflow-Option und *fixedtextformat* wird automatisch auf *true* gesetzt.
- In Unicode-fähigen Sprachen lassen sich Encodings für Seiteninhalte am einfachsten mit dem Encoding *unicode* erstellen. Daneben können bei Bedarf 8-Bit-Encodings und Ein-Byte-Text für Symbolfonts verwendet werden.
- Für chinesischen, japanischen und koreanischen Text dürfen nur Unicode-CMaps verwendet werden, da der Wrapper immer Unicode an den PDFlib-Kern übergibt (siehe Abschnitt 4.5, »Encodings für Chinesisch, Japanisch und Koreanisch«, Seite 110).

Clients können also ohne zusätzliche Konfiguration Unicode-Strings an PDFlib-API-Funktionen übergeben.

4.2.2 Sprachbindungen mit UTF-8-Unterstützung

Programmiersprachen ohne internen Unicode-fähigen Stringtyp können Unicode-Strings im UTF-8-Format trotzdem verarbeiten. Mit der Option *stringformat=utf8* lassen sich die folgenden PDFlib-Sprachbindungen Unicode-fähig machen:

- C
- Cobol
- Perl
- PHP

- Ruby

Wenn Sie mit einer dieser Sprachbindungen arbeiten, empfehlen wir die Verwendung von UTF-8. Mit dem folgenden Funktionsaufruf lässt sich eine Sprachbindung sofort nach der Erstellung ein neues PDFlib-Objekts Unicode-fähig machen:

```
p.set_option("stringformat=utf8");
```

Falls Unicode-Verarbeitung für eine Anwendung erforderlich ist, empfehlen wir den obigen Funktionsaufruf, um die Sprachbindung auf Basis von UTF-8 Unicode-fähig zu machen. Nach diesem Aufruf verhält sich die Sprachbindung wie eine Unicode-fähige Sprachbindung, außer dass der Client an alle API-Funktionen UTF-8-Strings liefern muss. Der Aufruf hat die folgenden zusätzlichen Konsequenzen:

- Das API benötigt alle Strings, also Name-Strings, Content-Strings, Hypertext-Strings und Optionslisten im UTF-8-Format mit oder ohne BOM.
- In der C-Sprachbindung werden Name-Strings als Funktionsparameter weiterhin als UTF-16 erwartet, wenn die Option *length* auf einen Wert größer 0 gesetzt ist.

Unicode-Konvertierung. Wenn Sie auch andere Encodings als Unicode verwenden müssen, müssen Sie sie vor der Übergabe an PDFlib erst nach Unicode im Format UTF-8 oder UTF-16 konvertieren. Für weitere Informationen und Tipps zur Konvertierung von Strings nach Unicode in allen gängigen Sprachumgebungen siehe *Kapitel 2, »Sprachbindungen von PDFlib«, Seite 27*.

4.3 Nicht Unicode-fähige Sprachbindungen

Folgende PDFlib-Sprachbindungen sind nicht Unicode-fähig:

- C (kein integrierter String-Datentyp)
- Cobol (kein integrierter String-Datentyp)
- Perl
- PHP
- Ruby

Wir empfehlen, diese Sprachbindungen mit der Option *stringformat* Unicode-fähig zu machen, siehe Abschnitt 4.2.2, »Sprachbindungen mit UTF-8-Unterstützung«, Seite 100. Der Rest dieses Abschnitts ist nur relevant für Anwendungen in einer der oben genannten Sprachen und unter der Voraussetzung, dass die Option *stringformat=utf8* nicht gesetzt ist.

Unicode-Konvertierung. PDFlib bietet die Konvertierungsfunktionen *PDF_convert _to_ unicode( )*, die Strings zwischen den Formaten UTF-8, UTF-16- und UTF-32 oder von beliebigen Encodings nach Unicode mit einem optionalen BOM konvertiert.

Für C-Benutzer hat das Format UTF-8 mit BOM den Vorteil, dass solche Strings von PDFlib über das BOM automatisch erkannt werden. Damit lassen sich wie bei einem richtigen Unicode-Workflow Fonts mit *encoding=unicode* laden, Hypertext-Strings mit *hypertextencoding=unicode* und Name-Strings mit *usehypertextencoding=true* behandeln.

Für weitere Informationen zu den Konvertierungsmethoden von Unicode-Strings in den jeweiligen Sprachbindungen siehe die entsprechenden Abschnitte in Kapitel 2, »Sprachbindungen von PDFlib«, Seite 27.

Unicode-Verarbeitung und Stringtypen. Auch in nicht Unicode-fähigen Sprachbindungen können Unicode-Strings verwendet werden, die Stringbehandlung ist aber etwas umständlicher und abhängig vom Stringtyp. Das PDFlib-API definiert und verwendet die unten angegebenen Stringtypen. In der PDFlib-Referenz sind Parameter und Optionen als Content-String, Hypertext-String oder Name-String gekennzeichnet (die Bezeichnungen sind historische Fehlbezeichnungen). Die Stringbehandlung ist in Tabelle 4.2 zusammengefasst und wird in den folgenden Abschnitten näher erläutert.

Tabelle 4.2 Übersicht über Stringbehandlung für verschiedene Stringtypen

Stringtyp	*Beispielparameter und -optionen*	*entsprechende Optionen/ Interpretation*
Content- String	*Parameter* `text` *von PDF_fit_textline( ) und PDF_add_ textflow( ).*	`textformat` `encoding`
Hypertext-String	▸ *Option* `fieldname` *von PDF_add_table_cell( )* ▸ *Option* `name` *von PDF_define_layer( )* ▸ *Option* `destname` *von f PDF_create_action( )* ▸ *Parameter* `text` *von PDF_create_bookmark( )*	`hypertextformat` `hypertextencoding`
Name-String	▸ *Parameter* `filename` *von PDF_begin_document( ) und PDF_create_pvf( )* ▸ *Parameter* `fontname` *von PDF_load_font( )* ▸ *Parameter* `profilename` *von PDF_load_iccprofile( )*	`usehypertextencoding`, `filenamehandling`
Strings in Optionslisten		*mit BOM: UTF-8* *ohne BOM: abhängig vom Stringtyp*

Content-Strings. Content-Strings werden zur Erstellung von Seiteninhalt (Seitenbeschreibungen) verwendet, unter Verwendung des Encodings, das vom Benutzer für den jeweiligen Font gewählt wurde. In diese Kategorie gehören alle Funktionsparameter in der PDFlib-Referenz vom Typ *text* für Funktionen, die Seiteninhalte bearbeiten. Da Content-Strings als Glyphen aus einem bestimmten Font dargestellt werden, hängt der Bereich der verwendbaren Zeichen von der Kombination aus Font und Encoding ab.

Die Interpretation von Content-Strings wird mit der Option *textformat* (siehe unten) und *encoding* für *PDF_load_font( )* gesteuert. Ist *textformat=auto* (Standardwert), wird für die Encodings *unicode* und *glyphid* sowie für UCS-2- und UTF-16-CMaps das Format *utf16* verwendet. Für alle anderen Encodings kommt das Format *bytes* zum Einsatz. In der C-Sprachbindung ist die Länge der UTF-16-Strings in einem eigenen Längenparameter zu übergeben.

Hypertext-Strings. Hypertext-Strings werden vorwiegend für interaktive Funktionen wie Lesezeichen und Anmerkungen verwendet und in der PDFlib-Referenz als *Hypertext string* bezeichnet. Unter anderem gehören in diese Kategorie zahlreiche Parameter und Optionen für interaktive Funktionen. Der Bereich der darstellbaren Zeichen hängt von externen Faktoren, etwa den in Acrobat und unter dem Betriebssystem verfügbaren Fonts ab.

Hypertext-Strings werden anhand der Optionen *hypertextformat* und *hypertextencoding* interpretiert (siehe unten). Bei der Voreinstellung *hypertextformat=auto* wird das Format *utf16* für *hypertextencoding=unicode* und sonst das Format *bytes* verwendet. In der C-Sprachbindung ist die Länge der UTF-16-Strings in einem eigenen Längenparameter zu übergeben.

Name-Strings. Name-Strings werden für externe Dateinamen, Fontnamen, Blocknamen usw. verwendet und in der PDFlib-Referenz als *Name string* bezeichnet. Sie unterscheiden sich geringfügig von Hypertext-Strings.

Ein besonderer Fall sind Dateinamen: die Option *filenamehandling* gibt an, wie PDFlib an das API übergebene Dateinamen so konvertiert, dass sie mit dem lokalen Dateisystem verwendet werden können.

Name-Strings werden fast wie Content-Strings interpretiert. Standardmäßig werden Name-Strings im Encoding *host* interpretiert. Beginnt der Name-String jedoch mit einem UTF-8-BOM, wird vom Format UTF-8 ausgegangen (bzw. vom Format EBCDIC UTF-8, wenn der String mit einem EBCDIC UTF-8 BOM beginnt). Ist die Option *usehypertextencoding=true*, wird das in *hypertextencoding* festgelegte Encoding auch auf Name-Strings angewendet. Dies kann zum Beispiel für Font- oder Dateinamen in Shift-JIS genutzt werden. Wenn *hypertextencoding=unicode*, erwartet PDFlib einen UTF-16-String, der mit zwei Nullbytes beendet werden muss.

In der C-Sprachbindung muss der Parameter *length* bei UTF-8-Strings gleich 0 sein. Ist er von 0 verschieden, wird der String als UTF-16 interpretiert. Bei allen anderen nicht Unicode-fähigen Sprachen gibt es in den API-Funktionen keinen Parameter *length*, und Name-Strings müssen immer im Format UTF-8 übergeben werden. Zur Erstellung von Unicode-fähigen Name-Strings müssen Sie sie in das Format UTF-8 konvertieren.

Textformat für Content- und Hypertext-Strings. Unicode-Strings können in PDFlib in den Formaten UTF-8, UTF-16 oder UTF-32 in beliebiger Bytereihenfolge übergeben werden. Dies lässt sich für Text in Seitenbeschreibungen mit der Option *textformat* und für interaktive Elemente mit der Option *hypertextformat* steuern. Tabelle 4.3 zeigt, welche Werte diese beiden Optionen annehmen können. Der Standardwert für die Option *[hyper]textformat* ist *auto*. Mit der Option *usehypertextencoding* stellen Sie das gleiche Verhalten auch für Name-Strings ein. Der Vorgabewert für die Option *hypertextencoding* ist *auto*.

Obwohl die Einstellung *textformat* auf alle Encodings wirkt, ist sie am sinnvollsten für das Encoding *unicode*. Tabelle 4.4 beschreibt die Interpretation von Text-Strings für verschiedenen Kombinationen Encodings und *textformat*. Wenn ein Code- oder Unicode-Wert in einem Content-String nicht durch eine passende Glyphe in dem ausgewählten Font dargestellt werden kann, steuert die Option *glyphcheck* das Verhalten von PDFlib (siehe Abschnitt »Glyphenersetzung«, Seite 126).

Tabelle 4.3 Werte für die Optionen `textformat` *und* `hypertextformat`

[hyper]textformat	*Erklärung*
bytes	*Ein Byte im String entspricht einem Zeichen. Dies eignet sich in erster Linie für 8-Bit-Encodings und Symbolfonts. Beginnt der String mit einem UTF-8-BOM, so wird dieser nach der Auswertung entfernt.*
utf8	*Strings werden im Format UTF-8 erwartet. Ist* `glyphcheck=error`*, lösen ungültige UTF-8-Sequenzen eine Exception aus; andernfalls werden sie gelöscht.*
ebcdicutf8	*Strings werden im Format UTF-8 in EBCDIC-Kodierung erwartet (nur auf iSeries und zSeries).*
utf16	*Strings werden im Format UTF-16 erwartet. Der Unicode Byte Order Mark (BOM) am Anfang des Strings wird ausgewertet und dann entfernt. Ist der BOM nicht vorhanden, wird der String in der Bytereihenfolge des Systems erwartet (auf Intel x86-Architekturen gilt »Little-Endian«-Bytereihenfolge und auf Sparc- oder PowerPC-Systemen »Bid-Endian«).*
utf16be	*Strings werden im UTF-16-Format in »Big-Endian«-Bytereihenfolge erwartet. Es findet keine besondere Behandlung des Byte Order Mark statt.*

Tabelle 4.3 Werte für die Optionen textformat *und* hypertextformat

[hyper]textformat	*Erklärung*
utf16le	*Strings werden im UTF-16-Format in »Little-Endian«-Bytereihenfolge erwartet. Es findet keine besondere Behandlung des Byte Order Mark statt.*
auto	*Content-Strings: Entspricht* bytes *für 8-Bit-Encodings und Nicht-Unicode-CMaps und* utf16 *bei erweiterter Adressierung* (unicode, glyphid *oder UCS2- bzw. UTF16-CMap).* *Hypertext-Strings: UTF-8- und UTF-16-Strings mit BOM werden erkannt (in C müssen UTF-16-Strings mit zweifacher Null enden). Beginnt der String nicht mit einem BOM, wird er gemäß der Option* hypertextencoding *als 8-Bit kodierter String interpretiert.* *In Umgebungen ohne native Unicode-Unterstützung wird Text mittels dieser Einstellung in der Regel korrekt interpretiert.*

Optionslisten. Strings in Optionslisten müssen besonders berücksichtigt werden, da sie sich in nicht Unicode-fähigen Sprachbindungen nicht als Unicode-Strings im Format UTF-16, sondern nur als Byte-Strings ausdrücken lassen. Für Unicode-Optionen wird deshalb UTF-8 verwendet. PDFlib interpretiert die Option abhängig davon, ob sich am Anfang ein BOM befindet. Anhand des BOM wird das Stringformat und anhand des Stringtyps (Content-, Hypertext- oder Name-String) das passende Encoding bestimmt. Im Einzelnen wird eine Stringoption wie folgt interpretiert:

- Beginnt die Option mit einem UTF-8-BOM *(oxEF oxBB oxBF)*, wird sie als String im Format UTF-8 interpretiert. Beginnt sie mit einem EBCDIC-UTF-8 BOM *(ox57 ox8B oxAB)*, wird sie als String im Format EBCDIC-UTF-8 interpretiert. Ist kein BOM vorhanden, wird der String je nach Stringtyp interpretiert:
  - Content-Strings werden entsprechend der jeweils anwendbaren Option *encoding* interpretiert oder entsprechend dem Encoding des entsprechenden Fonts.
  - Hypertext-Strings werden entsprechend der Option *hypertextencoding* interpretiert.
  - Wenn *usehypertextencoding=true* werden Name-Strings entsprechend der Hypertext-Einstellung interpretiert und sonst entsprechend dem Encoding *auto*.

Beachten Sie, dass vor die Zeichen { und } das Zeichen \ gestellt werden muss, um sie in einer Stringoption zu verwenden. Für ältere Encodings wie Shift-JIS besteht nach wie vor die Anforderung, dass den Bytewerten *ox7B* und *ox7D* generell der Wert *ox5C* voranzustellen ist. Sie sollten für Optionen deshalb UTF-8 verwenden (statt Shift-JIS oder anderer älterer Encodings).

Tabelle 4.4 Beziehung von Encoding und Textformat

[hypertext]encoding	*textformat=bytes*	*textformat=utf8, utf16, utf16be oder utf16le*
Alle Stringtypen:		
auto	*siehe Abschnitt »Automatische Auswahl des Encodings«, Seite 107*	
unicode *und UCS2- bzw. UTF-16-CMaps*	*8-Bit-Codes adressieren Unicode-Werte von U+0000 bis U+00FF*	*beliebiger Unicode-Wert, gemäß des gewählten Textformats[1] kodiert*
andere nicht Unicode-basierte CMaps	*beliebige Ein-Byte- oder Multibyte-Codes gemäß der gewählten CMap*	*PDFlib löst eine Exception aus*

Tabelle 4.4 Beziehung von Encoding und Textformat

[hypertext]encoding	***textformat=bytes***	***textformat=utf8, utf16, utf16be oder utf16le***
Nur Content-Strings:		
8-Bit und `builtin`	*8-Bit-Codes*	*Konvertiert Unicode-Werte in 8-Bit-Codes gemäß des gewählten Encoding*[1].
`glyphid`	*8-Bit-Codes adressieren Glyph-IDs zwischen 0 und 255*	*Unicode-Werte werden als Glyph-IDs*[2] *interpretiert Surrogatpaare werden nicht interpretiert.*

1. Ist das Unicode-Zeichen im Font nicht verfügbar, löst PDFlib eine Exception aus oder ersetzt das Zeichen gemäß der Option `glyphcheck`.
2. Ist die Glyph-ID im Font nicht verfügbar, löst PDFlib eine Exception aus oder ersetzt sie gemäß der Option `glyphcheck` *durch die Glyph-ID 0.*

4.4 Ein-Byte- (8-Bit-)Encodings

Hinweis *Die Informationen in diesem Abschnitt werden für Unicode-Workflows nicht benötigt.*

8-Bit-Encodings (auch Ein-Byte-Encodings genannt) bilden einem Bytewert 0x01-0xFF auf ein einzelnes Zeichen mit einem Unicode-Wert mit BMP ab (d.h. U+0000...U+FFFF). Sie sind auf 255 verschiedene Zeichen beschränkt, da der Code 0 (null) für das Zeichen *.notdef* U+0000 reserviert ist. PDFlib enthält interne Definitionen der folgenden Encodings:

```
winansi (identical to cp1252; superset of iso8859-1),
macroman (the original Macintosh character set),
macroman_apple (similar to macroman, but replaces currency with Euro),
ebcdic (EBCDIC code page 1047), ebcdic_37 (EBCDIC code page 037),
pdfdoc (PDFDocEncoding),
iso8859-1, iso8859-2, iso8859-3, iso8859-4, iso8859-5, iso8859-6, iso8859-7, iso8859-8,
iso8859-9, iso8859-10, iso8859-13, iso8859-14, iso8859-15, iso8859-16,s
cp1250, cp1251, cp1252, cp1253, cp1254, cp1255, cp1256, cp1257, cp1258
```

Host-Encoding. Das Encoding *host* spielt eine Sonderrolle, da es sich um kein bestimmtes Encoding handelt. Je nach aktueller Plattform wird es auf ein spezielles 8-Bit-Encoding abgebildet:

- Unter IBM zSeries mit MVS oder USS auf *ebcdic;*
- Unter IBM i5/iSeries auf *ebcdic_37;*
- Unter Windows auf *winansi;*
- Auf allen anderen Systemen auf *iso8859-1;*

Das Encoding *host* eignet sich insbesondere bei plattformunabhängigen Testprogrammen. Es wird jedoch davon abgeraten, es im produktiven Einsatz zu verwenden; stattdessen sollte das benötigte Encoding immer explizit angegeben werden.

Automatische Auswahl des Encodings. PDFlib unterstützt ein Verfahren, mit dem für bestimmte Umgebungen automatisch der jeweils am besten passende Zeichensatz bestimmt wird. Das Schlüsselwort *auto* als Wert für *encoding* legt ein plattform- und umgebungsspezifisches 8-Bit-Encoding für Textfonts wie folgt fest:

- Unter Windows: aktuelle Codepage des Systems (siehe unten)
- Unter Unix und OS X: *iso8859-1* (mit Ausnahme von LWFN-PostScript-Fonts unter OS X, für die *auto* auf *macroman* abgebildet wird)
- Auf IBM i5/iSeries: Encoding des aktuellen Jobs *(IBMCCSIDoooooooooooo)*
- Auf IBM zSeries: *ebcdic* (entspricht Codepage 1047).

Bei Symbolfonts wird das Schlüsselwort *auto* auf das Encoding *builtin* abgebildet (siehe Abschnitt 5.4.2, »Auswahl eines Encodings für Symbolfonts«, Seite 131). Obwohl automatisches Encoding in vielen Situationen bequem sein mag, ist die Portabilität Ihrer PDFlib-Clientprogramme damit aber nicht mehr gewährleistet.

Das Encoding *auto* wird als Standard-Encoding für Name-Strings in nicht Unicode-fähigen Sprachbindungen verwendet, da es das geeignetste Encoding für Dateinamen usw. ist (siehe Abschnitt 4.3, »Nicht Unicode-fähige Sprachbindungen«, Seite 102).

Abrufen von System-Codepages. PDFlib kann Codepage-Definitionen vom Betriebssystem abrufen. Statt an *PDF_load_font( )* den Namen eines integrierten oder benutzerdefinierten Encodings zu übergeben, verwenden Sie einfach einen dem System bekann-

ten Namen. Diese Funktion ist nur auf bestimmten Plattformen verfügbar. Die Syntax des Strings für den Encodingnamen ist entsprechend plattformabhängig:

- Unter Windows lautet der Name des Encodings *cp<nummer>*, wobei *<nummer>* der Nummer einer im System installierten Ein-Byte-Codepage entspricht (Informationen über Multibyte-Codepages in Windows finden Sie in Abschnitt 6.5.1, »Verwendung von CJK-Fonts vom Typ TrueType und OpenType«, Seite 176):

  ```
  font = p.load_font("Helvetica", "cp1250", "");
  ```

  Ein-Byte-Codepages werden in ein internes 8-Bit-Encoding umgewandelt, während Multibyte-Codepages immer auf *unicode* abgebildet werden. Text muss in einem Format übergeben werden, das zur gewählten Codepage kompatibel ist (zum Beispiel SJIS für *cp932)*.
- Auf IBM i5/iSeries kann ein beliebiger *Coded Character Set Identifier* (CCSID) verwendet werden. Der CCSID ist als String zu übergeben, und PDFlib fügt das Präfix *IBMCCSID* an die übergebene Codepage-Nummer an. PDFlib füllt die Nummer außerdem mit führenden Nullzeichen (0) auf, wenn diese über weniger als 5 Zeichen verfügt. Wird 0 (Null) als Codepage-Nummer übergeben, so wird das Encoding des aktuellen Jobs verwendet:

  ```
  font = p.load_font("Helvetica", "273", "");
  ```

- Auf IBM zSeries mit USS oder MVS kann ein beliebiger *Coded Character Set Identifier* (CCSID) verwendet werden. Der CCSID ist als String zu übergeben, und PDFlib leitet den übergebenen Namen der Codepage unverändert ans System weiter:

  ```
  font = p.load_font("Helvetica", "IBM-273", "");
  ```

Benutzerdefinierte 8-Bit-Encodings. Neben vordefinierten Encodings unterstützt PDFlib benutzerdefinierte 8-Bit-Encodings. Diese benötigen Sie, wenn Sie mit einem Zeichensatz arbeiten, der in PDFlib intern nicht verfügbar ist, zum Beispiel eine andere EBCDIC-Codepage, als die von PDFlib intern unterstützte. Zusätzlich zu Zeichensatztabellen mit PostScript-Glyphnamen akzeptiert PDFlib Unicode-basierte Codepage-Tabellen.

Folgende Schritte sind erforderlich, um ein benutzerdefiniertes Encoding in einem PDFlib-Programm zu verwenden (alternativ dazu lässt sich das Encoding auch zur Laufzeit mit *PDF_encoding_set_char( )* aufbauen):

- Legen Sie eine Beschreibung des Encodings in einer einfachen Textdatei an.
- Konfigurieren Sie das Encoding in der PDFlib-Ressourcendatei (siehe Abschnitt 3.1.3, »Ressourcenkonfiguration und Dateisuche«, Seite 65).
- Stellen Sie eine Schrift bereit, die alle im Encoding verwendeten Zeichen enthält.

In der Encoding-Datei werden zeilenweise alle Glyphnamen mit den dazugehörigen Codes aufgelistet. Das folgende Beispiel zeigt den Anfang einer Encoding-Definition:

```
% Encoding-Definition für PDFlib mittels Glyphnamen
% name          code    Unicode (optional)
space           32      0x0020
exclam          33      0x0021
...
```

Ist kein Unicode-Wert festgelegt, sucht PDFlib in seinen internen Tabellen nach einem geeigneten Unicode-Wert. Statt eines Glyphnamens kann ein Unicode-Wert angegeben werden:

```
% Codepage-Definition für PDFlib mittels Unicode-Werten
% Unicode       code
0x0020          32
0x0021          33
...
```

Der Inhalt einer Encoding- oder Codepage-Datei ist nach folgenden Regeln aufgebaut:

- Kommentare beginnen mit einem Prozentzeichen '%' und enden am Zeilenende.
- Der erste Eintrag in einer Zeile ist entweder ein PostScript-Zeichenname oder ein hexadezimaler Unicode-Wert, der sich aus dem Präfix *ox* und vier hexadezimalen Ziffern (in Groß- oder Kleinschreibung) zusammensetzt. Darauf folgen Leer- oder Tabulatorzeichen sowie ein hexadezimaler (0x00–0xFF) oder dezimaler (0–255) Zeichencode. Encoding-Dateien mit Glyphnamen können optional eine dritte Spalte mit dem entsprechenden Unicode-Wert enthalten.
- Zeichencodes, die in der Encoding-Datei nicht vorkommen, gelten als nicht definiert. Alternativ dazu kann für nicht kodierte Zeichen der Unicode-Wert *oxoooo* oder der Zeichenname *.notdef* verwendet werden.
- Alle Unicode-Werte eines Encodings oder einer Codepage-Datei müssen kleiner als U+FFFF sein.

4.5 Encodings für Chinesisch, Japanisch und Koreanisch

Hinweis *Die Informationen in diesem Abschnitt werden für Unicode-Workflows nicht benötigt.*

Von diversen Standardisierungsgremien und Unternehmen wurde über die Jahre hinweg ein breites Spektrum an CJK-Encodings entwickelt. Da ein Encoding für CJK-Text wesentlich komplizierter ist als für lateinischen Text, reicht ein einfaches 8-Bit-Encoding nicht aus. Stattdessen unterstützt PDF das Konzept der *character collections* und *character maps* (CMaps) zur Anordnung der Zeichen in einem Font.

Hinweis *CMaps werden nur für ältere CJK-Encodings verwendet; in Unicode-basierten Workflows sind sie nicht erforderlich. Mit der Funktion PDF_convert_to_unicode( ) lassen sich Strings von älteren CJK-Encodings nach Unicode konvertieren.*

Vordefinierte CMaps für gängige CJK-Encodings. Tabelle 4.5 zeigt die vordefinierten CJK-CMaps. Sie unterstützen die meisten CJK-Encodings, die auf OS X-, Windows- und Unix-Systemen verwendet werden. Außerdem unterstützt werden andere herstellerspezifische Encodings, zum Beispiel Shift-JIS, EUC und ISO 2022 für Japanisch, GB und Big5 für Chinesisch und KSC für Koreanisch. Unicode-fähige CMaps sind auch für alle Sprachumgebungen erhältlich.

Hinweis *Unicode-fähige Sprachbindungen können nur Unicode-kompatible CMaps verwenden (UCS-2 oder UTF-16). Andere CMaps werden nicht unterstützt (siehe Abschnitt 4.6, »Adressierung von Zeichen«, Seite 113).*

CMap-Konfiguration. Um chinesische, japanische oder koreanische (CJK) Textausgabe mit einer vordefinierten CMap zu erzeugen, benötigt PDFlib die passenden CMap-Dateien, so dass der eingehende Text verarbeitet werden kann und CJK-Encodings auf Unicode abgebildet werden können. Die CMap-Dateien sind in einem eigenen Paket verfügbar und wie folgt zu installieren:

- Unter Windows werden die CMap-Dateien automatisch gefunden, wenn Sie sie ins Unterverzeichnis *resource/cmap* des PDFlib-Installationsverzeichnisses kopieren.
- Auf anderen Systemen können Sie die CMap-Dateien in ein beliebiges Verzeichnis stellen, müssen dann aber den *SearchPath* manuell zur Laufzeit festlegen:

```
p.set_option("SearchPath={{/path/to/resource/cmap}}");
```

Alternativ dazu können Sie den Zugriff auf die CJK-CMap-Dateien konfigurieren, indem Sie die Umgebungsvariable *PDFLIBRESOURCEFILE* auf eine UPR-Konfigurationsdatei setzen, die die passende *SearchPath*-Definition enthält.

Tabelle 4.5 Vordefinierte CMaps für Japanisch/Chinesisch/Koreanisch

Sprache	CMap-Name	Encoding und Textformat
Vereinfachtes Chinesisch	UniGB-UCS2-H UniGB-UCS2-V	Unicode-Encoding (UCS-2) für die Character Collection Adobe GB1
	UniGB-UTF16-H UniGB-UTF16-V	Unicode-Encoding (UTF-16BE) für die Character Collection Adobe GB1. Enthält Zuordnungen für alle Zeichen des GB-18030-2000-Zeichensatzes.
	GB-EUC-H GB-EUC-V	Microsoft-Codepage 936 (Charset 134), GB 2312-80-Zeichensatz, EUC-CN-Encoding

Tabelle 4.5 Vordefinierte CMaps für Japanisch/Chinesisch/Koreanisch

Sprache	CMap-Name	Encoding und Textformat
	GBpc-EUC-H GBpc-EUC-V	Macintosh, GB-2312-80-Zeichensatz, EUC-CN-Encoding, Script Manager Code 2
	GBK-EUC-H, -V	Microsoft-Codepage 936 (Charset 134), GBK-Zeichensatz, GBK-Encoding
	GBKp-EUC-H GBKp-EUC-V	Wie GBK.EUC-H, ersetzt aber lateinische Zeichen halber Breite durch proportionale Zeichen und ordnet dem Zeichencode 0x24 das Dollarzeichen ($) statt des Yuan-Symbols (¥) zu.
	GBK2K-H, -V	GB-18030-2000-Zeichensatz, gemischtes 1-, 2- und 4-Byte-Encoding
Traditionelles Chinesisch	UniCNS-UCS2-H UniCNS-UCS2-V	Unicode-Encoding (UCS-2) für die Character Collection Adobe CNS1
	UniCNS-UTF16-H UniCNS-UTF16-V	Unicode-Encoding (UTF-16BE) für die Character Collection Adobe CNS1. Enthält Zuordnungen für alle HKSCS-2001 (2- und 4-Byte-Zeichencodes)
	B5pc-H, -V	Macintosh, Big-Five-Zeichensatz, Big-Five-Encoding, Script Manager Code 2
	HKscs-B5-H HKscs-B5-V	Hong Kong SCS (Supplementary Character Set), eine Erweiterung von Big-Five-Zeichensatz und -Encoding
	ETen-B5-H, -V	Microsoft-Codepage 950 (Charset 136), Big-Five mit ETen-Erweiterungen
	ETenms-B5-H ETenms-B5-V	Wie ETen-B5-H, ersetzt aber lateinische Zeichen halber Breite durch proportionale Zeichen
	CNS-EUC-H, -V	CNS-11643-1992-Zeichensatz, EUC-TW-Encoding
Japanisch	UniJIS-UCS2-H, -V	Unicode-Encoding (UCS-2) für die Character Collection Adobe Japan1
	UniJIS-UCS2-HW-H UniJIS-UCS2-HW-V	Wie UniJIS-UCS2-H, ersetzt aber proportionale lateinische Zeichen durch Zeichen halber Breite
	UniJIS-UTF16-H UniJIS-UTF16-V	Unicode-Encoding (UTF-16BE) für die Character Collection Adobe Japan1. Enthält Zuordnungen für alle Zeichen im JIS-X-0213:1000-Zeichensatz.
	83pv-RKSJ-H	Mac, JIS-X-0208 mit KanjiTalk6-Erweiterungen, Shift-JIS, Script Manager Code 1
	90ms-RKSJ-H 90ms-RKSJ-V	Microsoft-Codepage 932 (Charset 128), JIS-X-0208-Zeichensatz mit NEC- und IBM-Erweiterungen
	90msp-RKSJ-H 90msp-RKSJ-V	Wie 90ms-RKSJ-H, ersetzt aber lateinische Zeichen halber Breite durch proportionale Zeichen
	90pv-RKSJ-H	Mac, JIS-X-0208 mit KanjiTalk7-Erweiterungen, Shift-JIS, Script Manager Code 1
	Add-RKSJ-H, -V	JIS-X-0208-Zeichensatz mit Fujitsu FMR-Erweiterungen, Shift-JIS-Encoding
	EUC-H, -V	JIS-X-0208-Zeichensatz, EUC-JP-Encoding
	Ext-RKSJ-H, -V	JIS-C-6226-Zeichensatz (JIS78) mit NEC-Erweiterungen, Shift-JIS-Encoding
	H, V	JIS-X-0208-Zeichensatz, ISO-2022-JP-Encoding
Koreanisch	UniKS-UCS2-H, -V	Unicode-Encoding (UCS-2) für die Character Collection Adobe Korea1
	UniKS-UTF16-H, -V	Unicode-Encoding (UTF-16BE) für die Character Collection Adobe Korea1
	KSC-EUC-H, -V	KS-X-1001:1992-Zeichensatz, EUC-KR-Encoding
	KSCms-UHC-H KSCms-UHC-V	Microsoft-Codepage 949 (Charset 129), KS-X-1001:1992-Zeichensatz plus 8822 zusätzliche Hangul-Zeichen, Unified-Hangul-Code-Encoding (UHC)

Tabelle 4.5 Vordefinierte CMaps für Japanisch/Chinesisch/Koreanisch

Sprache	CMap-Name	Encoding und Textformat
	KSCms-UHC-HW-H KSCms-UHC-HW-V	Wie KSCms-UHC-H, ersetzt aber proportionale lateinische Zeichen durch Zeichen halber Breite
	KSCpc-EUC-H	Mac, KS-X-1001:1992 mit Mac-OS-KH-Erweiterungen, Script Manager Code 3

Codepages für benutzerdefinierte CJK-Fonts. PDFlib unterstützt die in Tabelle 4.6 aufgeführten Codepages. PDFlib unter Windows unterstützt zusätzlich jede im System installierte Codepage.

Tabelle 4.6 CJK-Codepages (müssen mit `textformat=auto` *oder* `textformat=bytes` *verwendet werden)*

Sprache	Codepage	Format	Encoding
Einfaches Chinesisch	cp936	GBK	GBK
Traditionelles Chinesisch	cp950	Big Five	Big Five mit Microsoft-Erweiterungen
Japanisch	cp932	Shift-JIS	JIS X 0208:1997 mit Microsoft-Erweiterungen
Koreanisch	cp949	UHC	KS X 1001:1992, restliche 8822 Hangul als Erweiterung
	cp1361	Johab	Johab

4.6 Adressierung von Zeichen

In manchen Umgebungen dürfen Programmierer Sourcecode nur mit 8-Bit-Encodings (wie *winansi* oder *ebcdic)* schreiben. Es ist dabei mühsam, einzelne Unicode-Zeichen in 8-Bit-Text aufzunehmen, statt gleich alle Zeichen in Multibyte-Kodierung umzusetzen. Um Entwicklern in dieser Situation zu unterstützen, bietet PDFlib mehrere Ersatzschreibweisen.

4.6.1 Escape-Sequenzen

PDFlib unterstützt ein Verfahren, um beliebige Werte über sogenannte *Escape-Sequenzen* bequem in Textstrings einzubinden (treffender wäre hier vielleicht der Begriff *Backslash-Ersetzung)*. Mit der Sequenz *\t* können beispielsweise Tabulatorzeichen in den Vorgabetext eines Textblocks aufgenommen werden, was durch direkte Tastatureingabe nicht möglich wäre. Außerdem sind Escape-Sequenzen zur Kodierung von Zeichen in Symbolfonts sinnvoll, oder für literale Strings bei Sprachbindungen ohne Escape-Sequenzen.

Eine Escape-Sequenz ist die Anweisung, eine bestimmte Sequenz durch einen Ein-Byte-Wert zu ersetzen. Die Sequenz beginnt mit dem Code für den Backslash '\' im aktuellen Encoding des Strings. Tabelle 4.7 gibt eine Übersicht über die Bytewerte, die sich aus der Ersetzung der Escape-Sequenzen ergeben; manche Werte sind in ASCII und EBCDIC unterschiedlich. Mit Escape-Sequenzen lassen sich nur Bytewerte im Bereich 0-255 ausdrücken.

Anders als bei anderen Programmiersprachen haben Escape-Sequenzen bei PDFlib je nach Typ immer feste Längen. Ein abschließendes Zeichen für die Sequenz wird deshalb nicht benötigt.

Tabelle 4.7 Escape-Sequenzen für Bytewerte

Sequenz	***Länge***	***OS X, Windows, Unix***	***EBCDIC-Plattformen***	***Übliche Interpretation***
`\f`	*2*	*0C*	*0C*	*Seitenvorschub (form feed)*
`\n`	*2*	*0A*	*15/25*	*Zeilenvorschub (line feed)*
`\r`	*2*	*0D*	*0D*	*Wagenrücklauf (carriage return)*
`\t`	*2*	*09*	*05*	*horizontaler Tabulator*
`\v`	*2*	*0B*	*0B*	*Zeilentabulator*
`\\`	*2*	*5C*	*E0*	*Gegenschrägstrich (Backslash)*
`\xNN`	*4*	*zwei Hexadezimalziffern für einen Bytewert, z.B.* `\xFF`		
`\NNN`	*4*	*drei Oktalziffern für einen Bytewert, z.B.* `\377`		

Escape-Sequenzen werden standardmäßig nicht ersetzt. Um Escape-Sequenzen in Strings zu nutzen, müssen Sie die Option *escapesequence* auf *true* setzen:

```
p.set_option("escapesequence=true");
```

Diese globale Option wirkt sich auf alle nachfolgend verwendeten Name-Strings, Hypertext-Strings und Content-Strings aus, was eventuell nicht wünschenswert ist. Um Es-

cape-Sequenzen auf Content-Strings zu beschränken, verwenden Sie stattdessen die folgende Text-Option:

```
p.set_text_option("escapesequence=true");
```

Anstelle der Option *escapesequence* können Sie Escape-Sequenzen in Strings auch mittels *PDF_convert_to_unicode( )* ersetzen.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `fonts/escape_sequences`.

Escape-Sequenzen werden in allen Content-, Hypertext- und Name-Strings nach der BOM-Erkennung, aber vor der Konvertierung in das Zielformat ausgewertet. Bei *textformat=utf16, utf16le* oder *utf16be* müssen Escape-Sequenzen als zwei Bytewerte entsprechend dem gewählten Format ausgedrückt werden. Jedes Zeichen in der Escape-Sequenz wird durch zwei Bytes dargestellt, wobei ein Byte den Wert Null hat. Bei *textformat=utf8* wird der Ergebniscode nicht nach UTF-8 konvertiert.

Ist eine Escape-Sequenz nicht interpretierbar (z.B. *\x* gefolgt von ungültigen hexadezimalen Ziffern), wird eine Exception ausgelöst. Bei Content-Strings wird das Verhalten durch die Einstellungen *glyphcheck* und *errorpolicy* gesteuert.

Wenn Escape-Sequenzen aktiviert sind, sollten Sie bei Windows-Pfadnamen mit Backslash-Zeichen aufpassen.

4.6.2 Character-Referenzen

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `fonts/character_references`.

Eine Character-Referenz ist die Anweisung, eine bestimmte Referenz-Sequenz durch einen Unicode-Wert zu ersetzen. Die Referenz-Sequenz beginnt mit dem Code für das Zeichen '&' im aktuellen Encoding des Strings und endet mit dem Code für ein Semikolon ';'. Es gibt unterschiedliche Möglichkeiten, die Zielwerte in Unicode auszudrücken:

Character-Referenzen wie in HTML. PDFlib unterstützt alle Character-Entity-Referenzen, die in HTML 4.0 definiert sind. Numerische Character-Referenzen können in Dezimal- oder Hexadezimalschreibweise angegeben sein. Für eine vollständige Liste aller HTML-Character-Referenzen siehe:

```
www.w3.org/TR/REC-html40/charset.html#h-5.3
```

Beispiele:

```
&shy;              U+00AD weiches Trennzeichen (soft hyphen)
&euro;             U+20AC Eurozeichen (Entity-Name)
&lt;               U+003C 'kleiner'-Zeichen
&gt;               U+003E 'größer'-Zeichen
&amp;              U+0026 Kaufmännisches 'und'-Zeichen (ampersand)
&Alpha;            U+0391 Alpha-Zeichen
```

Numerische Character-Referenzen. Numerische Character-Referenzen für Unicode-Zeichen sind ebenfalls in HTML 4.0 definiert. Sie benötigen ein Raute-Zeichen '#' und eine Dezimal- oder Hexadezimalzahl, wobei Hexadezimalzahlen mit einem kleinen 'x' oder einem großen 'X' eingeleitet werden, zum Beispiel:

```
&#173;              U+00AD weiches Trennzeichen (soft hyphen)
&#xAD;              U+00AD weiches Trennzeichen (soft hyphen)
&#229;              U+0229 Buchstabe a mit kleinem Kreis darüber (dezimal)
&#xE5;              U+00E5 Buchstabe a mit kleinem Kreis darüber (hexadezimal)
&#Xe5;              U+00E5 Buchstabe a mit kleinem Kreis darüber (hexadezimal)
&#x20AC;            U+20AC Eurozeichen (hexadezimal)
&#8364;             U+20AC Eurozeichen (dezimal)
```

Hinweis *Die Codes 128-159 (dezimal) bzw. 0x80-0x9F (hexadezimal) beziehen sich nicht auf Winansi-Zeichen. Der Unicode-Standard enthält an diesen Positionen keine druckbaren Zeichen, sondern nur Steuerzeichen.*

PDFlib-spezifische Entity-Namen. PDFlib unterstützt spezielle Character-Entity-Referenzen für die folgenden Gruppen von Unicode-Steuerzeichen:

- Steuerzeichen zum Überschreiben des Standardverhaltens beim Shaping, siehe Tabelle 6.4.
- Steuerzeichen zum Überschreiben der Standardformatierung für Bidi, siehe Tabelle 6.5.
- Steuerzeichen für Zeilenumbruch und Formatierung in Textflüssen, siehe Tabelle 8.2.

Beispiele:

```
&linefeed;          U+000A Steuerzeichen für Zeilenvorschub
&hortab;            U+0009 Horizontaler Tabulator
&ZWNJ;              U+200C Nullbreitenzeichen (ZERO WIDTH NON-JOINER)
```

Glyphnamen-Referenzen. Glyphnamen werden aus folgenden Quellen entnommen:

- Nach gängigen Glyphnamen wird in einer internen Liste gesucht.
- Nach fontspezifischen Glyphnamen wird im aktuellen Font gesucht. Diese Art von Character-Referenzen können nur bei Content-Strings verwendet werden, da sie immer einen Font benötigen.

Zur Identifikation von Glyphnamen-Referenzen muss im Namen hinter dem kaufmännischen 'und' ein Punkt '.' stehen, zum Beispiel:

```
&.three;            U+0033 gängiger Glyphname für die Ziffer 3
&.mapleleaf;        (PUA-Unicode-Wert) benutzerdefinierter Glyphname aus dem Font Carta
&.T.swash;          (PUA-Unicode-Wert) der zweite Punkt ist hier Teil des Glyphnamens
```

Glyphnamen-Referenzen sind in folgenden Situationen nützlich:

- Character-Referenzen mit fontspezifischen Glyphnamen können in Content-Strings verwendet werden, um alternative Formen (z.B. Swash-Zeichen) und Glyphen ohne eigenen Unicode-Wert (z.B. Symbole oder Ornamente) auszuwählen. Beachten Sie, dass sich Tabellenziffern und viele andere Funktionen leichter mit OpenType-Funktionen implementieren lassen (siehe Abschnitt 6.3, »OpenType-Layoutfunktionen«, Seite 160).
- Namen aus der Adobe Glyph List (inklusive der Formen *uniXXXX* und *u1XXXX)* sowie einige verbreitete »falsch benannte« Glyphen werden in Content- und Hypertext-Strings generell akzeptiert.

Bytewert-Referenzen. Numerische Werte können ebenfalls in Character-Referenzen übergeben werden, was bei der Adressierung von Glyphen in einem Symbolfont von

Vorteil sein kann. Sie benötigen ein zusätzliches Raute-Zeichen '#' und eine Dezimal- oder Hexadezimalzahl, wobei Hexadezimalzahlen mit einem kleinen 'x' oder einem großen 'X' eingeleitet werden, wie zum Beispiel beim Font Wingdings:

```
&.#x9F;         Aufzählungszeichen im Font Wingdings
&.#159;         Aufzählungszeichen im Font Wingdings
```

Einsatz von Character-Referenzen. Glyphnamen-Referenzen werden standardmäßig nicht ersetzt; um sie in allen Content-Strings verwenden zu können, müssen Sie die Option *charref* auf *true* setzen, zum Beispiel:

```
p.set_option("charref=true");
font = p.load_font("Helvetica", "winansi", "");
if (font == -1) { ... }
p.setfont(font, 24);
p.show_xy("Price: 500&euro;", 50, 500);
```

Diese globale Option wirkt sich auf alle nachfolgend verwendeten Name-Strings, Hypertext-Strings und Content-Strings aus, was eventuell nicht wünschenswert ist. Um Character-Referenzen auf Content-Strings zu beschränken, verwenden Sie stattdessen die folgende Text-Option:

```
font = p.load_font("Helvetica", "winansi", "");
if (font == -1) { ... }
p.set_text_option("charref=true font=" + font + " fontsize=24");
p.show_xy("Price: 500&euro;", 50, 500);
```

Weitere Anmerkungen zum Einsatz von Character-Referenzen:

- Character-Referenzern können in allen Content-Strings, Hypertext-Strings und Name-Strings vorkommen. Die einzige Ausnahme bilden fontspezifische Glyphnamen-Referenzen, die nur, wie oben beschrieben, bei Content-Strings verwendet werden können.
- Character-Referenzen können in Text mit Encoding *builtin* nicht ersetzt werden. Für Symbolfonts können Sie aber Character-Referenzen mit dem Encoding *unicode* verwenden.
- Character-Referenzen werden in Optionslisten nicht ersetzt, aber in Optionen mit dem Datentyp *Unichar* erkannt; in diesem Fall muss die Auszeichnung '&' und ';' entfallen. Diese Erkennung erfolgt immer und wird nicht durch die Option *charref* gesteuert.
- Wenn bei nicht Unicode-fähigen Sprachbindungen das Textformat=*utf16*, *utf16be* oder *utf16le* ist, müssen Character-Referenzen als zwei Bytewerte ausgedrückt werden. Bei *encoding=unicode* und *textformat=bytes* müssen Character-Referenzen in ASCII ausgedrückt werden (selbst auf EBCDIC-basierten Plattformen).
- Wenn eine Character-Referenz nicht aufgelöst werden kann (z.B. *&#* gefolgt von einer ungültigen Dezimalziffer oder *&* gefolgt von einem unbekannten Entity-Namen), wird eine Exception ausgelöst. Bei Content-Strings wird das Verhalten durch die Einstellungen *glyphcheck* und *errorpolicy* gesteuert. Bei *glyphcheck=none* wird die Referenz-Sequenz selbst in der erzeugten Ausgabe angezeigt.

5 Fontverarbeitung

5.1 Fontformate

5.1.1 TrueType-Fonts

TrueType-Dateiformate. PDFlib unterstützt vektorbasierte TrueType-Fonts. PDFlib unterstützt die folgenden Dateiformate für TrueType-Fonts:

- Windows TrueType-Fonts *(*.ttf)*, einschließlich westlichen, Symbol- und CJK-Fonts;
- TrueType-Collections *(*.ttc)* mit mehreren Fonts in einer einzelnen Datei. TTC-Dateien werden üblicherweise zum Gruppieren von CJK-Fonts verwendet, aber auch, um mehrere verschiedene Fonts einer westlichen Fontfamilie in einer einzelnen Datei zusammenzufassen.
- Fonts mit benutzerspezifischen Zeichen *(end-user defined characters, EUDC)*, die mit dem Microsoft-Tool *eudcedit.exe* erstellt wurden *(*.tte);*
- Unter OS X kann jeder auf dem System installierte TrueType-Font (einschließlich *.dfont)* auch in PDFlib verwendet werden.

TrueType-Fontnamen. Wenn Sie Fontdateien einsetzen, können Sie beliebige Alias-Namen verwenden (siehe Abschnitt »Quellen für Fontdaten«, Seite 136). Dieser Name wird zum Laden des Fonts verwendet und kann vom Dateinamen oder dem internen Namen des Fonts abweichen. Der Name eines TrueType-Fonts im generierten PDF-Dokument kann von den in PDFlib (oder Windows) benutzten Namen abweichen. Das ist normal und liegt darin begründet, dass PDF den PostScript-Namen eines TrueType-Fonts verwendet, der nicht unbedingt mit dem ursprünglichen TrueType-Namen übereinstimmt (zum Beispiel *TimesNewRomanPSMT* statt *Times New Roman).*

5.1.2 OpenType-Fonts

Das OpenType-Format vereint PostScript und TrueType. Es ist als Erweiterung des TrueType-Dateiformats implementiert und bietet ein einheitliches Format. OpenType-Fonts können optionale Tabellen enthalten, die zur Verbesserung der Textausgabe verwendet werden können, z.B. Ligaturen und Swash-Zeichen (siehe Abschnitt 6.3, »OpenType-Layoutfunktionen«, Seite 160), sowie Tabellen für das Shaping komplexer Schriftsysteme (siehe Abschnitt 6.4, »Ausgabe komplexer Schriftsysteme«, Seite 167).

Obwohl OpenType-Fonts ein einheitliches Container-Format für alle Plattformen bieten, kann es nützlich sein, die folgenden OpenType-Varianten zu verstehen, die manchmal zu Verwirrung führen:

- Zeichenbeschreibungsformat: OpenType-Fonts können Glyphenbeschreibungen enthalten, die auf TrueType oder PostScript basieren. Die PostScript-Variante wird auch CFF *(Compact Font Format)* oder Type 2 genannt und normalerweise mit der Endung **.otf* verwendet. Der Windows-Explorer stellt OpenType-Fonts immer mit dem Logo »O« dar.

- TrueType-Fonts und OpenType-Fonts mit TrueType-Zeichenbeschreibungen werden leicht verwechselt, da beide die Endung **.ttf* verwenden. Deshalb wendet der Windows-Explorer zur Unterscheidung folgendes Kriterium an: Wenn ein *.ttf*-Font eine digitale Signatur enthält, wird er mit dem Logo »O« dargestellt, ansonsten mit dem Logo »TT«. Da jedoch die digitale Signatur in OpenType-Fonts nicht zwingend erforderlich ist, kann sie nicht als verlässliches Unterscheidungskriterium verwendet werden.
- Die CID-Architektur *(Character ID)* wird für CJK-Fonts verwendet. Moderne CID-Fonts werden als OpenType-Fonts **.otf* mit PostScript-Zeichenbeschreibungen ausgeliefert. Aus praktischer Sicht sind sie nicht von einfachen OpenType-Fonts zu unterscheiden. Der Windows-Explorer stellt OpenType-CID-Fonts immer mit dem Logo »O« dar.

Beachten Sie, dass weder die Dateinamenserweiterung noch das vom Windows-Explorer angezeigte Logo etwas über die An- oder Abwesenheit von OpenType-Layoutfunktionen in einem Font aussagt. Für weitere Informationen siehe Abschnitt 6.3, »OpenType-Layoutfunktionen«, Seite 160.

5.1.3 WOFF-Fonts

WOFF *(Web Open Font Format)* ist ein einfaches, komprimiertes Dateiformat für TrueType- und OpenType-Fonts. Man kann es als neues Container-Format für bestehende Fontformate sehen, das es keine neuen typografischen Features bietet. WOFF wurde für die Verwen-dung im Web konzipiert und bietet Features zur Komprimierung sowie für Fontuntergruppen (Subsetting), um Fontdateien möglichst klein zu halten. WOFF soll demnächst eine W3C-Empfehlung werden; die WOFF-Spezifikation ist zu finden unter

`www.w3.org/TR/WOFF`

WOFF-Fonts verwenden üblicherweise die Dateinamenserweiterung *.woff*.

PDFlib unterstützt WOFF-Fonts unter der Voraussetzung, dass der zugrundeliegende TrueType- oder OpenType-Font unterstützt wird. Als WOFF-Font verpackte TrueType-Bitmap-Fonts werden zum Beispiel nicht unterstützt. Da die Betriebssysteme Windows und OS X WOFF-Fonts nicht unterstützen, können Sie nicht als Host-Fonts verwendet werden.

5.1.4 SVG-Fonts

Ein SVG-Font ist eine normale SVG-Grafikdatei, die eine SVG-Fontdefinition enthält (siehe auch Abschnitt 7.2, »SVG Graphics«, Seite 179). SVG-Fonts werden im allgemeinen als Standalone-Ressourcen in SVG-Grafiken verwendet. Sie können aber wie jedes andere Format in PDFlib geladen werden, auch wenn sie nicht aus einer SVG-Grafik heraus benutzt werden.

PDFlib verwendet das erste Element *font* in der SVG-Datei und ignoriert eventuell vorhandene grafische Inhalte in der Datei. Wird ein Font gefunden, wird ein Type-3-Font generiert, der intern den in SVG angegebenen Namen trägt.

Der sich hieraus ergebende Font wird wie benutzerdefinierte Type-3-Fonts behandelt (siehe Abschnitt 5.1.8, »Type-3-Fonts«, Seite 120). Eine Ressource namens *FontnameAlias* wird automatisch erzeugt. Sie verbindet den benutzerdefinierten Ressourcennamen

mit dem internen SVG-Fontnamen, so dass beide Namen zum Laden des Fonts verwendet werden können. SVG-Fonts verwenden normalerweise die Dateinamenserweiterung *.svg*.

5.1.5 PostScript-Type-1-Fonts

PostScript-Font- und Metrikdateien. PostScript-Type-1-Fonts bestehen aus zwei Teilen: der eigentlichen Zeichenbeschreibung und den Metrikdaten. PDFlib unterstützt folgende Formate für PostScript-Type-1-Font- und -Metrikdaten:

- Das plattformunabhängige Format AFM (Adobe Font Metrics) und das Windows-spezifische Format PFM (Printer Font Metrics) für Fontmetrikdaten.
- Das plattformunabhängige Format PFA *(Printer Font ASCII)* und das Windows-spezifische Format PFB *(Printer Font Binary)* für Zeichenbeschreibungen im PostScript-Type-1-Format.
- Unter OS X werden auch ressourcenbasierte PostScript-Type-1-Fonts, zum Beispiel LWFN-Fonts (LaserWriter Font), unterstützt. Diesen liegt ein Zeichensatzkoffer (FOND-Ressource oder FFIL) mit Metrikdaten bei (sowie Bildschirmfonts, die von PDFlib ignoriert werden).
  Beim Arbeiten mit PostScript-Systemfonts muss die LWFN-Datei im selben Ordner wie der Zeichensatzkoffer liegen und entsprechend der 5+3+3-Regel benannt sein.

PostScript-Fontnamen. Wenn Sie Fontdateien von der Festplatte einsetzen, können Sie beliebige Alias-Namen verwenden (siehe Abschnitt »Quellen für Fontdaten«, Seite 136). Es gibt mehrere Möglichkeiten, den korrekten Namen für einen PostScript-Font herauszufinden:

- Öffnen Sie die Zeichenbeschreibungsdatei *(*.pfa* oder **.pfb)* und finden Sie den String nach dem Eintrag */FontName*. Entfernen Sie den Schrägstrich / am Anfang und verwenden Sie den Rest als Schriftnamen.
- Unter Windows oder OS X können Sie auf die Fontdatei doppelklicken, um ein Fontbeispiel sowie den PostScript-Namen des Fonts anzuzeigen.
- Öffnen Sie die AFM-Metrikdatei und suchen Sie nach dem Eintrag *FontName*.

Hinweis *Der PostScript-Name des Fonts kann sich vom Windows-Namen deutlich unterscheiden. So erscheint »AvantGarde-Demi« (PostScript-Name) im Windows-Fontmenü zum Beispiel als »AvantGarde, Bold«.*

5.1.6 SING-Fonts (Glyphlets)

SING-Fonts *(Smart Independent Glyphlets)* sind technisch gesehen eine Erweiterung des OpenType-Fontformats. SING-Fonts wurden als Lösung für das Gaiji-Problem mit CJK-Text entwickelt, d.h. für benutzerdefinierte Glyphen, die nicht in Unicode oder einem anderen gängigen Standard-Encoding für CJK kodiert sind. Weitere Informationen über die Architektur von SING finden Sie im *Adobe Glyphlet Development Kit (GDK) for SING Gaiji Architecture,* das Sie hier herunterladen können:

`www.adobe.com/devnet/opentype/gdk/topic.html`

SING-Fonts enthalten normalerweise nur eine einzelne Glyphe (sie können zusätzlich eine vertikale Variante enthalten). Der Unicode-Wert dieser Haupt-Glyphe kann mit

PDFlib über die Glyph-ID und anschließend über den Unicode-Wert dieser Glyph-ID abgefragt werden:

```
maingid = (int) p.info_font(font, "maingid", "");
uv = (int) p.info_font(font, "unicode", "gid=" + maingid);
```

Wir empfehlen, SING-Fonts als Fallback-Font mit *PDF_load_font( )* und der Unteroption *gaiji* der Option *forcechars* von *fallbackfonts* zu verwenden; für weitere Information siehe Abschnitt 6.5.2, »EUDC- und SING-Fonts für Gaiji-Zeichen«, Seite 177.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `fonts/starter_fallback`.

Mit dem preiswerten Tool SigMaker von FontLab können Sie SING-Fonts erzeugen, die auf vorhandenen Bildern oder Glyphen eines anderen Fonts basieren:

```
www.fontlab.com/font-utility/sigmaker/
```

5.1.7 CEF-Fonts

Das Fontformat CEF *(Compact Embedded Font)* ist den SING-Fonts sehr ähnlich. Es umfasst eine Teilmenge eines OpenType-Fonts mit PostScript-Zeichenbeschreibungen. Anders als reguläre OpenType-Fonts enthalten CEF-Fonts jedoch nicht die gängigen TrueType-Fonttabellen. CEF-Fonts werden hauptsächlich von Adobe-Anwendungen verwendet. Während CEF-Fonts meist in SVG-Grafiken eingebettete werden, kommen sie manchmal auch als Standalone-Dateien vor. In diesem Fall wird normalerweise die Dateinamenserweiterung *.cef* verwendet. PDFlib behandelt CEF-Fonts im Wesentlichen genauso wie OpenType-Fonts.

5.1.8 Type-3-Fonts

Im Gegensatz zu den üblichen Fontformaten werden benutzerdefinierte Type-3-Fonts nicht aus einer Datei eingebettet, sondern müssen zur Laufzeit mit den Grafikfunktionen von PDFlib definiert werden. Type-3-Fonts können zu folgenden Zwecken eingesetzt werden:

- Bitmap-Fonts
- Benutzerdefinierte Grafiken wie Logos können schnell mit einfachen Textoperatoren gedruckt werden
- Japanische Gaiji-Zeichen (benutzerdefinierte Zeichen), die in den vordefinierten Fonts und Zeichensätzen nicht verfügbar sind.

Da alle PDF-Funktionen für Vektorgrafik, Rasterbilder und sogar Textausgabe in Type-3-Fontdefinitionen verwendet werden können, gibt es keinerlei Beschränkungen hinsichtlich der Zeicheninhalte in einem Type-3-Font. In Kombination mit der PDF-Importbibliothek PDI können Sie selbst komplexe Zeichnungen als PDF-Seite importieren und dann zur Definition eines Zeichens in einem Type-3-Font verwenden. Type-3-Fonts werden dabei meist für Bitmap-Glyphen eingesetzt, da es das einzige Fontformat in PDF ist, das Rasterbilder für Glyphen unterstützt. Das folgende Beispiel zeigt die Definition eines einfachen Type-3-Fonts:

```
p.begin_font("Fuzzyfont", 0.001, 0.0, 0.0, 0.001, 0.0, 0.0, "");

p.begin_glyph_ext(-1, "glyphname=circle width=1000 boundingbox={0 0 1000 1000}");
p.arc(500, 500, 500, 0, 360);
p.fill();
```

```
p.end_glyph();

p.begin_glyph_ext(-1, "glyphname=ring width=400 boundingbox={0 0 400 400}");
p.arc(200, 200, 200, 0, 360);
p.stroke();
p.end_glyph();

p.end_font();
```

Cookbook *Vollständige Codebeispiele hierzu finden Sie in den Cookbook-Topics* `fonts/starter_type3font`, `fonts/type3_bitmaptext`, `fonts/type3_rasterlogo` *und* `fonts/type3_vectorlogo`.

Der Font wird in PDFlib registriert und sein Name kann gemeinsam mit einem Encoding, das die Namen der Glyphen in dem Type-3-Font enthält, an *PDF_load_font( )* übergeben werden. Bei der Definition und Verwendung von Type-3-Fonts ist Folgendes zu beachten:

- Wurde der Font mit *encoding=unicode* geladen, können die Glyphen mit ihrem Unicode-Wert oder mit Glyphnamen-Referenzen der Form *&.<glyphname>;* adressiert werden, zum Beispiel *&.circle;*
- Wurde der Font mit *encoding=builtin* geladen, können Zeichencodes zur Adressierung von Glyphen verwendet werden, wobei der Code jeder Glyphe der Reihenfolge entspricht, in der die Glyphen erzeugt wurden; die Glyphe *.notdef* hat immer den Code 0.
- Wenn nur Unicode-Werte, aber keine Glyphnamen angegeben wurden, generiert PDFlib Glyphnamen in der Form *GXXX,* wobei *XXX* die Dezimalzahl der generierten Glyphe darstellt.
- Wir empfehlen, die Bildoption *inline* zur Definition von Bitmaps in Type-3.Fonts zu verwenden. Die Option *interpolate* für Rasterbilder kann zur Verbesserung des Bildschirm- und Druckbildes von Type-3-Fonts für Bitmaps verwendet werden.
- Werden normale Rasterbilddaten zur Definition von Zeichen verwendet, dann werden im Rasterbild nicht gesetzte Bildpunkte unabhängig vom Hintergrund weiß gedruckt. Um dies zu vermeiden und die ursprüngliche Hintergrundfarbe durchscheinen zu lassen, verwenden Sie den Parameter *mask* zur Konstruktion des Rasterbilds.
- Aufgrund von Einschränkungen in PDF-Viewern müssen alle Zeichen, die in der Textausgabe verwendet werden, auch tatsächlich im Font definiert sein: Wenn der Zeichencode *x* mit einer Textausgabe-Funktion angezeigt werden soll und das Encoding an Position *x* den Eintrag *glyphname* enthält, dann muss *glyphname* über *PDF_begin_glyph( )* definiert worden sein.
- Manche PDF-Viewer benötigen eine Glyphe namens *.notdef* für Codes, für die im Font kein entsprechender Glyphname definiert ist. Die Glyphe *.notdef* muss zwar vorhanden sein, die zugehörige Glyphenbeschreibung kann aber leer sein.
- Definitionen von Type-3-Glyphen enthalten keinerlei typografische Eigenschaften wie Oberlänge, Unterlänge usw. Diese können jedoch mit entsprechenden Optionen in *PDF_load_font( )* gesetzt werden.

5.2 Unicode-Zeichen und Glyphen

5.2.1 Glyph-IDs

Ein Font ist eine Sammlung von Glyphen, wobei jede Glyphe von ihrer geometrischen Kontur definiert wird. PDFlib weist jeder Glyphe aus dem Font eine Zahl zu. Diese Zahl wird die Glyph-ID oder GID genannt. GID 0 (Null) bezieht sich in allen Fontformaten auf die Glyphe *.notdef*. Das visuelle Erscheinungsbild der Glyphe *.notdef* variiert je nach Fontformat und Anbieter; typische Implementierungen sind das Leerzeichen oder ein leeres oder durchgestrichenes Rechteck. Die höchste GID ist um eins kleiner als die Anzahl der Zeichen im Font, die mit dem Schlüsselwort *numglyphs* von *PDF_info_font( )* abgefragt werden kann.

Die Zuordnung der Glyph-IDs hängt vom Fontformat ab:

- Da TrueType- und OpenType-Fonts bereits internen GIDs enthalten, verwendet PDFlib diese GIDs.
- Bei OpenType-CJK-Fonts mit CID werden CIDs als GIDs verwendet.
- Bei anderen Fonttypen nummeriert PDFlib die Glyphen nach der Reihenfolge der zugehörigen Zeichenbeschreibung im Font.

PDFlib unterstützt die Glyphenauswahl über GID als Alternative zu Unicode- und anderen Encodings (siehe Abschnitt »Glyphid-Encoding«, Seite 131). Direkte GID-Adressierung ist nur bei besonderen Anwendungen sinnvoll, zum Beispiel um Font-Übersichtstabellen durch Abfragen der Anzahl an Glyphen und Iteration über alle Glyph-IDs zu erstellen.

5.2.2 Unicode-Zuordnung für Glyphen

Unicode-Zuordnung. PDFlib weist jeder GID einen eindeutigen Unicode-Wert zu. Diese Zuordnung hängt vom Fontformat ab und wird in den folgenden Abschnitten für die unterstützten Fonts beschrieben. Obwohl jeder GID ein eindeutiger Unicode-Wert zugewiesen wird, ist dies umgekehrt nicht notwendigerweise der Fall, d.h. eine bestimmte Glyphe kann mehrere Unicode-Werte verkörpern. Gängige Beispiele in vielen TrueType- und OpenType-Fonts sind die leere Glyphe, die das Leerzeichen U+0020 darstellt, ebenso wie das geschützte Leerzeichen U+00A0 und eine Glyphe, die sowohl das Ohm-Zeichen U+2126 als auch den griechischen Großbuchstaben Omega U+03A9 darstellt. Wenn mehrere Unicode-Werte auf die gleiche Glyphe in einem Font weisen, ordnet PDFlib den ersten im Font gefundenen Unicode-Wert zu.

Nicht zugeordnete Glyphen und die Private Use Area (PUA). In manchen Fällen kann der Font keinen Unicode-Wert für eine bestimmte Glyphe liefern. In diesem Fall weist PDFlib der Glyphe einen Wert aus der Private Use Area (PUA) von Unicode zu (siehe Abschnitt 4.1, »Wichtige Unicode Konzepte«, Seite 93). Solche Zeichen werden als nicht zugeordnete Glyphen (*unmapped glyphs)* bezeichnet. Die Anzahl der nicht zugeordneten Glyphen in einem Font kann mit dem Schlüsselwort *unmappedglyphs* von *PDF_info_font( )* abgefragt werden. Nicht zugeordnete Glyphen werden als Unicode-Ersatzzeichen U+FFFD in der ToUnicode-CMap des Fonts dargestellt, die die Auffindbarkeit und Textextraktion steuert. Deshalb können nicht zugeordnete Glyphen nicht korrekt als Text aus dem generierten PDF extrahiert werden.

Wenn PDFlib nicht zugeordneten Glyphen PUA-Werte zuweist, werden aufsteigende Werte aus dem folgenden Pool verwendet:

- Die Basis bildet der Unicode-PUA-Bereich der Basic Multilingual Plane (BMP), also der Bereich U+E000 bis U+F8FF. Bei Bedarf werden zusätzliche PUA-Werte in Ebene 15 (U+F0000 bis U+FFFFD) verwendet.
- Zur Erzeugung neuer PUA-Werte werden keine Werte verwendet, die bereits intern im Font zugeordnet werden.
- PUA-Werte im Adobe-Bereich U+F600-F8FF werden nicht verwendet.

Die generierten PUA-Werte sind innerhalb eines Font eindeutig. Die Zuordnung erzeugter PUA-Werte zu Glyphen in einem Font ist von anderen Fonts unabhängig.

Unicode-Zuordnung für TrueType-, OpenType- und SING-Fonts. PDFlib behält die Unicode-Zuordnungen bei, die in den entsprechende *cmap*-Tabellen des Fonts gefunden wurden (die Auswahl der *cmap* richtet sich nach dem in *PDF_load_font( )* angegebenen Encoding). Wenn eine bestimmte Glyphe für mehrere Unicode-Werte verwendet wird, verwendet PDFlib den ersten im Font gefundenen Unicode-Wert.

Wenn die *cmap* keine Unicode-Zuordnung für eine Glyphe bietet, überprüft PDFlib die Glyphnamen in der *post*-Tabelle (falls im Font vorhanden) und bestimmt die Unicode-Zuordnungen anhand der Glyphnamen wie unten für Type-1-Fonts beschrieben.

In einigen Fällen liefern weder die Tabellen *cmap* noch *post* Unicode-Werte für alle Glyphen im Font. Dies trifft für Glyphvarianten (z.B. Swash-Zeichen), erweiterte Ligaturen und nicht-textuelle Symbole außerhalb des Unicode-Standards zu. In diesem Fall ordnet PDFlib den betreffenden Glyphen PUA-Werte zu, siehe »Nicht zugeordnete Glyphen und die Private Use Area (PUA)«, Seite 122.

Unicode-Zuordnung für Type-1-Fonts. Type-1-Fonts enthalten keine expliziten Unicode-Zuordnungen, sondern weisen jeder Glyphe einen eindeutigen Namen zu. PDFlib versucht mit Hilfe einer internen Zuordnungstabelle mit Unicode-Zuordnungen für mehr als 7 000 gängige Glyphnamen für eine Vielzahl von Sprachen und Schriften, auf diesen Glyphnamen basierende Unicode-Werte zuzuordnen. Die Zuordnungstabelle enthält circa 4 200 Glyphnamen aus der Adobe Glyph List (AGL)[1]. Allerdings können Type-1-Fonts Glyphnamen enthalten, die nicht in der internen Zuordnungstabelle enthalten sind; dies gilt insbesondere für Symbolfonts. In diesem Fall ordnet PDFlib den betreffenden Glyphen PUA-Werte zu, siehe »Nicht zugeordnete Glyphen und die Private Use Area (PUA)«, Seite 122.

Wenn die Metrik für einen Type-1-Font aus einer PFM-Datei geladen wird und keine PFB- oder PFA-Zeichenbeschreibungsdatei verfügbar ist, sind PDFlib die Glyphnamen des Fonts nicht bekannt. In diesem Fall weist PDFlib auf dem Encoding-Eintrag (*charset*) in der PFM-Datei basierende Unicode-Werte zu.

Unicode-Zuordnung für Type-3-Fonts. Da Type-3-Fonts ebenfalls auf Glyphnamen basieren, werden sie wie Type-1-Fonts behandelt. Ein wichtiger Unterschied ist jedoch, dass die Glyphnamen für Type-3-Fonts vom Benutzer angegeben werden (direkt über den Parameter *uv* oder indirekt über die Option *glyphname* von *PDF_begin_glyph_ext( )*). Wir empfehlen daher, entweder geeignete Unicode-Werte oder entsprechende AGL-Glyphnamen für die Glyphen in benutzerdefinierten Type-3-Fonts anzugeben. Dadurch

1. Die AGL ist abrufbar unter partners.adobe.com/public/developer/en/opentype/glyphlist.txt

wird sichergestellt, dass die richtigen Unicode-Werte von PDFlib zugeordnet werden können, so dass die generierten PDF-Dokumente durchsuchbar sind.

5.2.3 Unicode-Steuerzeichen

Steuerzeichen sind Unicode-Werte, die keine Glyphe darstellen, sondern bestimmte Formatierungsinformationen transportieren. PDFlib verarbeitet die folgenden Gruppen von Unicode-Steuerzeichen:

- Steuerzeichen für das Überschreiben des Standardverhaltens von Shaping (siehe Tabelle 6.4) und für das Überschreiben der Standardformatierung von bidirektionalem Text (siehe Tabelle 6.5) steuern komplexes Schrift-Shaping und die Verarbeitung von OpenType-Layoutfunktionen in Textline und Textflow. Nach der Auswertung werden diese Steuerzeichen entfernt.
- Steuerzeichen zur Formatierung von Zeilenumbruch und Textflow sind in Tabelle 8.1 aufgeführt. Nach der Auswertung werden diese Steuerzeichen entfernt.
- Weitere Unicode-Steuerzeichen in den Bereichen U+0001-U0019 und U+007F-U+009F werden durch das Zeichen *replacementchar* ersetzt.

Selbst wenn ein Font eine Glyphe für ein Steuerzeichen enthält, ist diese in der Regel nicht sichtbar, da PDFlib die Steuerzeichen entfernt (als Ausnahmen zu dieser Regel werden * * und *&SHY;* nicht entfernt). Mit *encoding=glyphid* können Steuerzeichen jedoch im Text erhalten bleiben und sichtbare Ausgabe produzieren.

5.3 Die Textverarbeitungs-Pipeline

Die Client-Anwendung liefert Text für die Seitenausgabe an PDFlib. Dieser Text wird gemäß einiger anwendungsspezifischer Encodings und Formate kodiert. Während die interne Verarbeitung von PDFlib auf dem Unicode-Standard basiert, benötigt die finale Textausgabe fontspezifische Glyph-IDs. PDFlib verarbeitet daher eingehende Strings für Seiteninhalte in einer dreistufigen Textverarbeitungs-Pipeline:

- Normalisierung von Eingabecodes zu Unicode-Werten; dieser Prozess wird durch das gewählte Encoding beschränkt.
- Konvertierung von Unicode-Werten zu fontspezifischen Glyph-IDs; dieser Prozess wird durch die verfügbaren Glyphen im Font beschränkt.
- Umwandlung von Glyph-IDs; dieser Prozess wird durch das Encoding der Ausgabe beschränkt.

Diese drei Stufen der Textverarbeitungs-Pipeline bestehen aus mehreren Teilprozessen, die über Optionen gesteuert werden können.

5.3.1 Normalisierung von Eingabe-Strings zu Unicode

Die folgenden Schritte werden für alle Encodings außer *encoding=glyphid* und nicht Unicode-basierte CMaps durchgeführt:

- Unicode-fähige Sprachbindungen: Wurde ein Ein-Byte-Encoding angegeben, wird UTF-16-basierter Text durch Weglassen der höchstwertigen Bytes zu Ein-Byte-Text konvertiert.
- Windows: Konvertierung von Multibyte-Text (z.B. *cp932)* zu Unicode.
- Escape-Sequenzen (siehe Abschnitt 4.6.1, »Escape-Sequenzen«, Seite 113) durch die entsprechenden numerischen Werte ersetzen.
- Character-Referenzen auflösen und durch die entsprechenden numerischen Werte ersetzen (siehe Abschnitt 4.6.2, »Character-Referenzen«, Seite 114, sowie Abschnitt »Character-Referenzen mit Glyphnamen«, Seite 125).
- Ein-Byte-Encoding: Konvertierung von Ein-Byte-Text zu Unicode entsprechend des festgelegten Encodings.
- Normalisierung von Text zu einer der Normalisierungsformen von Unicode (z.B. NFC) entsprechend der Option *normalize*.

Für weitere Informationen zu den Unicode-Zuordnungen für verschiedene Fontformate und Zeichentypen siehe auch Abschnitt 5.2.2, »Unicode-Zuordnung für Glyphen«, Seite 122.

Character-Referenzen mit Glyphnamen. Ein Font kann Glyphen enthalten, die nicht direkt zugänglich sind, da die entsprechenden Unicode-Werte nicht im Voraus bekannt sind (PDFlib ordnet PUA-Werte erst zur Laufzeit zu). Als Alternative für die Adressierung dieser Glyphen können Character-Referenzen mit Glyphnamen verwendet werden; für eine Beschreibung der Syntax siehe Abschnitt 4.6.2, »Character-Referenzen«, Seite 114. Diese Referenzen werden durch die entsprechenden Unicode-Werte ersetzt.

Wenn eine Charakter-Referenz in einem Content-String verwendet wird, versucht PDFlib, die angegebene Glyphe im aktuellen Font zu finden und die Referenz durch den Unicode-Wert der Glyphe zu ersetzen. Wenn eine Glyphe mit dem angegebenen Namen nicht im Font zur Verfügung steht, durchsucht PDFlib seine internen Glyphnamen-Tabelle nach einem entsprechenden Unicode-Wert. Dieser Unicode-Wert wird erneut zur

Überprüfung der Verfügbarkeit einer geeigneten Glyphe im Font wiederverwendet. Kann keine passende Glyphe gefunden werden, wird das Verhalten durch die Einstellungen *glyphcheck* und *errorpolicy* gesteuert. Character-Referenzen können nicht mit den Encodings *glyphid* oder *builtin* verwendet werden.

5.3.2 Konvertierung von Unicode-Werten zu Glyph-IDs

Die durch die Schritte in den vorigen Abschnitten bestimmten Unicode-Werte müssen unter Umständen aus verschiedenen Gründen geändert werden. Die unten aufgeführten Schritte werden für alle Encodings außer *encoding=glyphid* und nicht Unicode-basierte CMaps durchgeführt. Diese Ausnahmen werden folgendermaßen behandelt:

- Für nicht Unicode-basierte CMaps: für ungültige Code-Sequenzen wird eine Exception ausgelöst.
- Für *encoding=glyphid:* ungültige Glyph-IDs werden durch *replacementchar* ersetzt (bei *glyphcheck=replace)* oder durch die Glyph-ID 0 (*glyphcheck=none).* Bei *glyphcheck=error* wird eine Exception ausgelöst.

Zeichenvergabe aus Fallback-Fonts. Ersetzen der Unicode-Werte entsprechend der Unteroption *forcechars* der Option *fallbackfonts* und Bestimmung der Glyph-ID des entsprechenden Fallback-Fonts. Für weitere Information siehe Abschnitt 5.4.6, »Fallback-Fonts«, Seite 143.

Auflösen von Variantensequenzen. Bei einigen Fonts können auf Unicode-Zeichen Variantenselektoren folgen, die eine bestimmte Glyphvariante des Zeichens auswählen (siehe Abschnitt 6.5.4, »Variantenselektoren und Variantensequenzen von Unicode«, Seite 180). Wenn der Font eine Glyphvariante für die Variantensequenz enthält, wird die Glyph-ID der Variante anstelle der ursprünglichen Glyph-ID verwendet.

Konvertierung zu Glyph-IDs. Konvertierung der Unicode-Werte zu Glyph-IDs entsprechend der ermittelten Zuordnungen, siehe Abschnitt 5.2.2, »Unicode-Zuordnung für Glyphen«, Seite 122. Wenn keine entsprechende Glyph-ID für einen Unicode-Wert im Font gefunden werden konnte, hängen die nächsten Schritte von der Option *glyphcheck* ab:

- *glyphcheck=none:* Die Glyph-ID 0 wird verwendet, das heißt, die Glyphe *.notdef* wird in der Textausgabe verwendet. Wenn die Glyphe *notdef* eine sichtbare Form hat (oft ein leeres oder durchgestrichenes Rechteck), lässt sie die problematischen Zeichen auf der PDF-Seite sichtbar werden, was eventuell nicht gewünscht ist.
- *glyphcheck=replace* (Standardeinstellung): Ein Warnhinweis wird protokolliert und PDFlib versucht, den nicht zuzuordnenden Unicode-Wert durch das unten beschriebenen Verfahren zur Glyphen-Ersetzung zu ersetzen.
- *glyphcheck=error:* PDFlib gibt einen Fehler aus. Bei *errorpolicy=return* wird der Funktionsaufruf beendet, ohne eine Textausgabe zu erzeugen; *PDF_add/create_textflow( )* geben *-1* (in PHP: 0) aus. Bei *errorpolicy=exception* wird eine Exception ausgelöst.

Glyphenersetzung. Bei *glyphcheck=replace* werden nicht zuzuordnende Unicode-Werte rekursiv wie folgt ersetzt:

- Die beim Laden des Master-Fonts angegebenen Fallback-Fonts werden nach Glyphen für den Unicode-Wert durchsucht. Dies kann sich auf beliebig viele Fonts beziehen,

da für jeden Font mehr als ein Fallback-Font angegeben werden kann. Wird eine Glyphe in einem der Fallback-Fonts gefunden, wird sie verwendet.

- Selektion einer entsprechend dem Unicode-Wert ähnlichen Glyphe aus der PDFlib-internen Ersetzungstabelle. Der folgende Auszug aus der internen Liste enthält einige Ersatzglyphen. Wenn das erste Zeichen in der Liste nicht in einem Font verfügbar ist, wird es durch das zweite Zeichen ersetzt:

U+00A0 (geschütztes Leerzeichen)	U+0020 (Leerzeichen)
U+00AD (weiches Trennzeichen)	U+002D (Minuszeichen)
U+2010 (Bindestrich)	U+002D (Minuszeichen)
U+03BC (griechischer Kleinbuchstabe MU)	U+00C5 (Mikro-Zeichen)
U+212B (Ångström-Zeichen)	U+00B5 (lateinischer Großbuchstabe A mit Kreis über dem A: Å)
U+220F (Produkt-Zeichen)	U+03A0 (griechischer Großbuchstabe PI)
U+2126 (Ohm-Zeichen)	U+03A9 (griechischer Großbuchstabe OMEGA)

  Neben der internen Tabelle werden die Fullwidth-Zeichen U+FF01 bis U+FF5E durch entsprechende ISO 8859-1-Zeichen (d.h. U+0021 bis U+007E) ersetzt, wenn sie im Font nicht verfügbar sind.
- Zerlegung von Unicode-Ligaturen in ihre Glyphen-Bestandteile (zum Beispiel Ersetzen von U+FB00 *Latin small ligature ff* durch die Sequenz U+0066 *f,* U+0066 *f).*
- Selektion von Glyphen mit der gleichen Unicode-Semantik entsprechend ihres Glyphnamens. Suffixe in Glyphnamen, die durch einen Punkt vom Namen getrennt sind, werden entfernt, wenn die entsprechende Glyphe nicht verfügbar ist (so würde *A.swash* zum Beispiel durch *A* oder *g.alt* durch *g* ersetzt).

Wenn durch keine dieser Maßnahmen eine Glyphe für den Unicode-Wert gefunden wird, wird die Option *replacementchar* wie folgt ausgewertet:

- Bei *replacementchar=auto* (Standardeinstellung) werden die Zeichen U+00A0 (geschütztes Leerzeichen) und U+0020 (Leerzeichen) geprüft. Wenn diese auch nicht verfügbar sind, wird die Glyph-ID 0 verwendet (Symbol für fehlende Glyphe).
- Wenn ein Unicode-Zeichen als *replacementchar* angegeben wurde, wird es anstelle des ursprünglichen Zeichens verwendet.
- Bei *replacementchar=drop* wird das Zeichen aus dem Text entfernt und keine Ausgabe erzeugt.
- Bei *replacementchar=error* wird eine Exception ausgelöst. Dies kann zur Vermeidung von unlesbarer Textausgabe verwendet werden und wird für PDF/A und PDF/X-4/5 erzwungen.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `fonts/glyph_replacement`.

5.3.3 Umwandlung von Glyph-IDs

Die ermittelten Glyph-IDs sind noch nicht final, da mehrere Transformationen angewandt werden müssen, bevor die endgültige Ausgabe erstellt werden kann. Die Details dieser Transformationen sind abhängig vom Font und mehreren Optionen. Die folgenden Schritte werden für alle Encodings durchgeführt, ausgenommen nicht Unicode-fähige CMaps mit *keepnative=true*.

Vertikale Glyphen. Bei Fonts in vertikaler Schreibrichtung können einige Glyphen durch ihre vertikalen Pendants ersetzt werden. Dies ist nur möglich, wenn die OpenType-Layout-Tabelle *vert* im Font vorhanden ist.

OpenType-Layoutfunktionen. OpenType-Funktionen können Ligaturen, Swash-Zeichen, Kapitälchen und viele andere typografische Varianten erzeugen, indem eine oder mehrere Glyph-IDs durch andere Werte ersetzt werden. Für Informationen zu OpenType-Funktionen siehe Abschnitt 6.3, »OpenType-Layoutfunktionen«, Seite 160). OpenType-Layoutfunktionen sind nur für geeignete Fonts relevant (siehe »Voraussetzungen für OpenType Layout-Funktionen«, Seite 155) und werden entsprechend der Option *features* angewendet.

Shaping für komplexe Schriftsysteme. Shaping ordnet den Text neu an und bestimmt die geeignete Glyphvariante anhand der Position eines Zeichens (zum Beispiel Anfangs-, mittlere, letzte oder isolierte Form von arabischen Zeichen). Für Informationen zum Thema Shaping siehe Abschnitt 6.4, »Ausgabe komplexer Schriftsysteme«, Seite 167. Shaping ist nur für geeignete Fonts relevant (siehe Abschnitt »Voraussetzungen für das Shaping«, Seite 169) und wird entsprechend der Option *shaping* angewendet.

5.4 Laden von Fonts

5.4.1 Auswahl eines Encodings für Textfonts

Fonts können explizit mit der Funktion *PDF_load_font( )* oder implizit durch Übergabe der Optionen *fontname* und *encoding* an bestimmte Funktionen wie *PDF_add/create_textflow( )* oder *PDF_fill_textblock( )* geladen werden. Ungeachtet der Methode, die zum Laden eines Fonts verwendet wird, muss ein geeignetes Encoding festgelegt werden. Das Encoding bestimmt Folgendes:

- in welchem Textformat PDFlib den übergebenen Text erwartet;
- welche Glyphen aus einem Font verwendet werden können;
- wie der Text auf der Seite und die Glyphinformationen im Font im PDF-Ausgabedokument gespeichert werden.

PDFlib unterstützt den Unicode-Standard[1], der im Wesentlichen ISO 10646 entspricht. Da Unicode von den meisten modernen Entwicklungsumgebungen unterstützt wird, möchten wir die Verwendung von Unicode-Strings zur PDF-Ausgabe möglichst einfach gestalten. Entwickler, die nicht mit Unicode arbeiten, brauchen ihre Anwendungen aber nicht auf Unicode umzustellen, sondern können mit anderen Encodings arbeiten.

Die Wahl des Encodings ist abhängig vom Font, den verfügbaren Textdaten und einigen Aspekten der Programmierung. Um Sie bei der Auswahl eines geeigneten Encodings zu unterstützen, geben die folgenden Abschnitte einen Überblick über die verschiedenen Klassen von Encodings.

Unicode-Encoding. Mit *encoding=unicode* können Sie Unicode-Strings an PDFlib übergeben. Dieses Encoding wird für alle Fontformate unterstützt. Je nach Sprachbindung kann der von der Programmiersprache (z.B. Java) unterstützte Datentyp Unicode-String oder aber Byte-Arrays mit Unicode in einem der Formate UTF-8, UTF-16 oder UTF-32 mit Little- oder Big-Endian-Bytereihenfolge verwendet werden (z.B. C).

Mit *encoding=unicode* können alle Glyphen eines Fonts adressiert werden; Shaping für komplexe Schriftsysteme und OpenType-Layoutfunktionen werden unterstützt. PDFlib prüft, ob der Font eine Glyphe für den gewünschten Unicode-Wert enthält. Ist keine Glyphe vorhanden, kann eine Ersatzglyphe aus demselben oder einem anderen Font verwendet werden (siehe Abschnitt 5.4.6, »Fallback-Fonts«, Seite 143).

In nicht Unicode-fähigen Sprachbindungen erwartet PDFlib standardmäßig UTF-16-kodierten Text. Sie können jedoch unter Angabe von *textformat=bytes* auch Ein-Byte-Strings übergeben. In diesem Fall stellen die Byte-Werte die Zeichen U+0001 - U+00FF dar, das heißt, den ersten Unicode-Block mit Basic Latin (wie ISO 8859-1). Mit Character-Referenzen können jedoch auch Unicode-Werte außerhalb dieses Bereiches in Ein-Byte-Text verwendet werden.

Einige Fonttypen in PDF (Type-1-, Type-3- und auf Glyphnamen basierende OpenType-Fonts) unterstützen nur Ein-Byte-Text. Mit PDFlib können aber selbst bei diesen Fonttypen mehr als 255 verschiedenen Zeichen verwendet werden.

Der Nachteil bei *encoding=unicode* ist, dass Text in gängigen Ein- oder Multibyte-Encodings (außer ISO 8859-1) nicht verwendet werden kann.

1. Siehe www.unicode.org

Ein-Byte-Encodings. 8-Bit Encodings (auch Ein-Byte-Encodings genannt) bilden jedes Byte eines Text-Strings auf ein einzelnes Zeichen ab und sind damit auf jeweils 255 unterschiedliche Zeichen beschränkt (der Wert 0 ist nicht verfügbar). Diese Art des Encoding wird für alle Fontformate unterstützt. PDFlib prüft, ob der Font Glyphen enthält, die zu dem ausgewählten Encoding passen. Wird keine Mindestanzahl an brauchbaren Glyphen gefunden, protokolliert PDFlib einen Warnhinweis. Ist überhaupt keine brauchbare Glyphe für das ausgewählte Encoding im Font vorhanden, wird der Font nicht geladen und die folgende Meldung ausgegeben: *font doesn't support encoding*. PDFlib prüft, ob der Font eine Glyphe für den gewünschten Eingabewert enthält. Ist keine Glyphe vorhanden, kann eine Ersatzglyphe aus demselben oder einem anderen Font herangezogen werden (siehe Abschnitt 5.4.6, »Fallback-Fonts«, Seite 143).

In nicht Unicode-fähigen Sprachbindungen erwartet PDFlib standardmäßig Ein-Byte-kodierten Text. Sie können jedoch unter Angabe von *textformat=utf8* oder *utf16* auch UTF-8- oder UTF-16-Strings übergeben.

Für ausführliche Informationen zu 8-Bit-Encodings siehe Abschnitt 4.4, »Ein-Byte-(8-Bit-)Encodings«, Seite 107. 8-Bit-Encodings können aus verschiedenen Quellen entnommen werden:

- Vordefinierte Encodings, von denen eine große Auswahl zur Verfügung steht (siehe Abschnitt 4.4, »Ein-Byte- (8-Bit-)Encodings«, Seite 107). Diese umfassen die wichtigsten Encodings, die auf verschiedensten Systemen und in unterschiedlichsten Sprachräumen im Einsatz sind.
- Benutzerdefinierte Encodings, die in einer externen Datei bereitgestellt oder zur Laufzeit dynamisch mit *PDF_encoding_set_char( )* zusammengestellt werden. Diese Encodings können auf Glyphnamen oder Unicode-Werten basieren.
- Encodings, die vom Betriebssystem bezogen werden. Diese so genannten System-Encodings werden nur auf Windows und den Systemen IBM i5/iSeries und zSeries unterstützt.

Der Nachteil bei Ein-Byte-Encodings ist, dass nur eine begrenzte Menge von Zeichen und Glyphen vorhanden ist. Aus diesem Grund werden Shaping komplexer Schriftsystem und OpenType-Layoutfunktionen nicht für Single-Byte-Encodings unterstützt.

Builtin-Encoding. Unter anderem können Sie *encoding=builtin* angeben, um Single-Byte-Codes für nicht-textuelle Glyphen aus Symbolfonts zu verwenden. Das Format des internen Encodings eines Fonts hängt vom Fonttyp ab:

- TrueType: Das Encoding wird basierend auf der symbolischen *cmap* des Fonts erzeugt, also dem *(3, 0)*-Eintrag in der cmap-Tabelle.
- OpenType-Fonts können ein Encoding in der CFF-Tabelle enthalten.
- PostScript-Type-1-Fonts enthalten immer ein Encoding.
- Bei Type-3-Fonts wird das Encoding durch die ersten 255 Glyphen eines Fonts bestimmt.

Wenn der Font kein Builtin-Encoding enthält, kann er nicht geladen werden (zum Beispiel OpenType-CJK-Fonts). Sie können das Schlüsselwort *symbolfont* von *PDF_info_font( )* verwenden. Wird *false* zurückgegeben, handelt es sich bei dem Font um einen Textfont, der auch mit einem der üblichen Ein-Byte-Encodings geladen werden kann. Wenn das Schlüsselwort *symbolfont* den Wert *true* zurückgibt, ist dies jedoch nicht möglich. Die Glyphen in diesen Symbolfonts können nur verwendet werden, wenn Sie den entsprechenden Code für jede Glyphe kennen (siehe Abschnitt 5.4.2, »Auswahl eines Encodings für Symbolfonts«, Seite 131).

In nicht Unicode-fähigen Sprachbindungen erwartet PDFlib standardmäßig Ein-Byte-Text. Dadurch können Sie auch Ein-Byte-Werte verwenden, die üblicherweise zur Adressierung einiger Symbolfonts verwendet wurden; mit anderen Encodings ist dies nicht möglich. Sie können allerdings auch Text in einem Unicode-Format übergeben, zum Beispiel mit *textformat=utf16*.

Der Nachteil bei *encoding=builtin* ist, dass in Ein-Byte-kodiertem Text keine Character-Referenzen verwendet werden können.

Multibyte-Encodings. Dieser Typ von Encoding wird von CJK-Fonts unterstützt, das sind TrueType- und OpenType-CID-Fonts mit chinesischen, japanischen oder koreanischen Zeichen. Verschiedene Encoding-Schemas wurden für diese Schriften entwickelt, zum Beispiel Shift-JIS und EUC für Japanisch, GB und Big5 für Chinesisch und KSC für Koreanisch. Multibyte-Encodings werden durch die CMaps von Adobe oder Codepages von Windows festgelegt (siehe Abschnitt 4.5, »Encodings für Chinesisch, Japanisch und Koreanisch«, Seite 110).

Diese veralteten Encodings werden nur in nicht Unicode-fähigen Sprachbindungen unterstützt, ausgenommen Unicode CMaps; diese sind äquivalent zu *encoding=unicode*.

In nicht Unicode-fähigen Sprachbindungen erwartet PDFlib standardmäßig Multibyte-kodierten Text *(textformat=bytes)*.

Bei Multibyte-Encodings wird der Text im PDF genauso ausgegeben wie vom Benutzer geliefert, sofern die Option *keepnative* auf *true* gesetzt ist.

Der Nachteil bei Multibyte-Encodings ist, dass PDFlib den eingegebenen Text nur auf gültige Syntax überprüft, jedoch nicht, ob eine Glyphe für den übergebenen Text im Font vorhanden ist. Ebenso kann kein Unicode-Text übergeben werden, da PDFlib die Unicode-Werte nicht in die entsprechenden Multibyte-Sequenzen konvertieren kann. Außerdem können Charakter-Referenzen, OpenType-Layoutfunktionen und Shaping komplexer Schriftsysteme nicht verwendet werden.

Glyphid-Encoding. PDFlib unterstützt für alle Fontformate *encoding=glyphid*. Bei diesem Encoding können alle Glyphen in einem Font adressiert werden, und zwar mit dem Nummerierungsschema, das in Abschnitt 5.2.1, »Glyph-IDs«, Seite 122 erklärt wird. Numerische Glyph-IDs laufen von 0 zu einem theoretischen Maximalwert von 65 565 (Fonts mit einer solch hohen Anzahl von Glyphen sind jedoch nicht verfügbar). Der Maximalwert für Glyph-IDs kann mit dem Schlüsselwort *maxcode* von *PDF_info_font( )* abgefragt werden.

In nicht Unicode-fähigen Sprachbindungen benötigt PDFlib standardmäßig Double-Byte-kodierten Text *(textformat=utf16)*.

PDFlib überprüft, ob die übergebene Glyph-ID für den Font gültig ist. Komplexes Schrift-Shaping und OpenType-Layoutfunktionen werden nicht unterstützt.

Da Glyph-IDs spezifisch für den jeweiligen Font sind und manchmal sogar von PDFlib erzeugt werden, ist *encoding=glyphid* generell nicht für reguläre Text-Ausgabe geeignet. Dieses Encoding wird hauptsächlich zum Drucken vollständiger Font-Tabellen mit allen Glyphen verwendet.

5.4.2 Auswahl eines Encodings für Symbolfonts

Symbolfonts sind Fonts, die Symbole, Logos, Piktogramme oder andere nicht-textuelle Glyphen enthalten. Sie werfen mehrere Fragen auf, die bei Textfonts keine Rolle spielen. Das zugrunde liegende Problem ist, dass der Unicode-Standard per Definition in der

Regel Symbolglyphen nicht kodiert (Ausnahmen von dieser Regel sind zum Beispiel die Glyphen des gängigen Fonts ZapfDingbats). Um Symbolfonts überhaupt in Unicode-Workflows verwenden zu können, weisen TrueType- und OpenType-Fonts ihren Glyphen in der Regel Unicode-Werte in der Private Use Area (PUA) zu. Bei PostScript-Type-1-Fonts kann dieses Verfahren wegen fehlender Tabellen für die Unicode-Zuordnung nicht angewendet werden. Sie verwenden zur Glyphenauswahl daher im Allgemeinen die Codes von lateinischen Zeichen. In allen Fontformaten haben Symbolglyphen normalerweise benutzerdefinierte Glyphnamen.

Für die Auswahl von Glyphen aus Symbolfonts hat dies die folgenden Konsequenzen:

- Symbolfonts vom Typ TrueType und OpenType lassen sich am besten laden mit *encoding=unicode*. Wenn Sie die den Glyphen zugewiesenen PUA-Werte kennen, können Sie diese zur Auswahl der Symbolglyphen im Text übergeben. Dies setzt voraus, dass Sie die PUA-Zuweisungen im Font bereits kennen.
- Da PDFlib den Symbolfonts vom Typ PostScript Type 1 die PUA-Werte intern zuweist, sind diese im Voraus nicht bekannt.
- Wenn Sie zur Adressierung von Glyphen in einem Symbolfont lieber 8-Bit-Codes verwenden, können Sie den Font mit *encoding=builtin* laden und die 8-Bit-Codes im Text übergeben. Die Ziffer 4 (Code 0x34) würde zum Beispiel das Häkchen-Symbol im Font ZapfDingbats auswählen.

Um Symbolfonts mit *encoding=unicode* zu verwenden, müssen passende Unicode-Werte für den Text verwendet werden:

- Die Zeichen im Font *Symbol* haben alle richtige Unicode-Werte.
- Die Zeichen im Font *ZapfDingbats* haben Unicode-Werte im Bereich U+2007 - U+27BF.
- Die Symbolfonts von Microsoft, wie Wingdings und Webdings, verwenden PUA-Unicode-Werte im Bereich U+F020 - U+F0FF (obwohl die Windows-Anwendung *Zeichentabelle* sie mit Ein-Byte-Codes darstellt).
- Bei anderen Fonts muss der Unicode-Wert für einzelne Glyphen im Font im Voraus bekannt sein oder zur Laufzeit mit *PDF_info_font( )* bestimmt werden, zum Beispiel durch Übergabe der Glyphnamen in PostScript-Type-1-Fonts.

Steuerzeichen. Die Unicode-Steuerzeichen im Bereich U+0001 - U+001F, siehe Tabelle 8.1, werden bei Textflow selbst mit *encoding=builtin* unterstützt. Codes < 0x20 werden als Steuerzeichen interpretiert, wenn der Symbolfont keine Glyphen für den Code enthält. Dies gilt für den Großteil der Symbolfonts.

Da der Code für das Zeilenvorschub-Zeichen bei ASCII und EBCDIC unterschiedlich ist, empfehlen wir, das Literalzeichen 0x0A auf EBCDIC-Systemen zu vermeiden und stattdessen die Escape-Sequenz von PDFlib *\n* mit der Option *escapesequence=true* zu verwenden. Beachten Sie, dass *\n* beim PDFlib-API ankommen muss, in C ist zum Beispiel die Sequenz *\\n* erforderlich.

Character-Referenzen. In Symbolfonts werden Character-Referenzen unterstützt. Symbolfonts enthalten jedoch normalerweise keine Glyphen für das Et-Zeichen U+0026 '&', das die Character-Referenzen einleitet. Der Code 0x26 kann auch nicht verwendet werden, da er einer bestehenden Glyphe im Font zugeordnet sein könnte. Deshalb sollten Symbolfonts bei der Verwendung von Character-Referenzen mit *encoding=unicode* geladen werden. Character-Referenzen können nicht mit *encoding=builtin* verwendet werden.

5.4.3 Beispiel: Auswahl einer Glyphe aus dem Symbolfont Wingdings

Lassen Sie uns einen Blick auf ein Beispiel werfen, da es viele verschiedene Möglichkeiten gibt, Zeichen aus einem Symbolfont auszuwählen und manche nicht zu den gewünschten Ergebnissen führen.

Die Zeichen im Font verstehen. Lassen Sie uns zunächst anhand der Windows-Anwendung *charmap* (Windows-Zeichentabelle) einige Informationen über das Zielzeichen im Font sammeln:

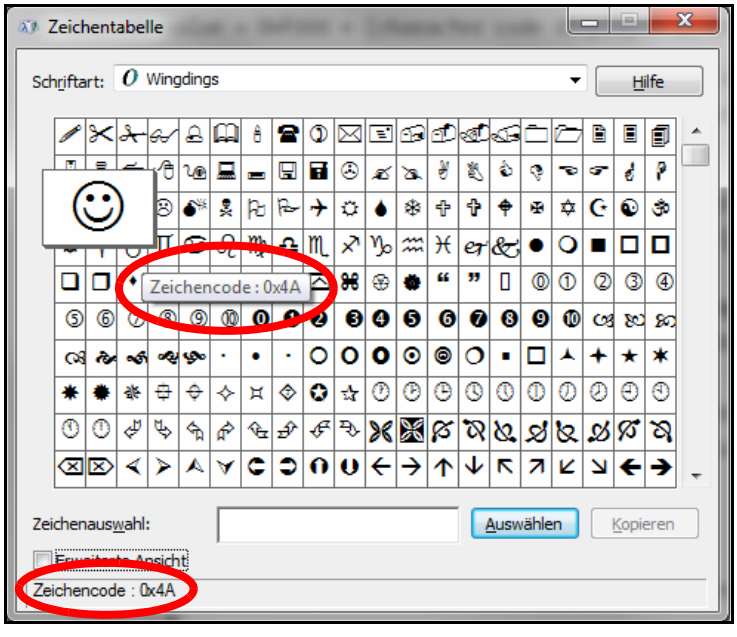


Abb. 5.1
Windows-Zeichentabelle mit Font Wingdings

- Die *Zeichentabelle* stellt die Glyphen im Font Wingdings dar, bietet aber in der *Erweiterten Ansicht* keinen Zugriff auf den Unicode-Wert. Das liegt daran, dass der Font Symbol-glyphen enthält, für die keine standardisierten Unicode-Werte registriert sind. Stattdessen verwenden die Glyphen im Font Unicode-Dummywerte in der Private Use Area (PUA). In der Windows-Zeichentabelle werden diese Werte nicht angezeigt.
- In der linken unteren Ecke des Dialogs oder wenn Sie mit dem Mauszeiger über das Smiley-Symbol *(smileface)* fahren, wird *Zeichencode: 0x4A* angezeigt. Dies ist der Bytecode der Glyphe.
  Dieser Code stimmt mit dem Großbuchstaben *J* im Winansi-Encoding überein. Wenn Sie das Zeichen zum Beispiel in die Zwischenablage kopieren, entsteht der entsprechende Unicode-Wert U+004A, also das Zeichen *J*, aus dem Einfügen des Inhalts der Zwischenablage in einen Texteditor. Dennoch ist dies nicht der Unicode-Wert des Zeichens, und somit kann U+004A oder *J* nicht zur Auswahl in Unicode-Workflows verwendet werden.
- Das intern im Font verwendete Unicode-Zeichen wird in der Windows-Zeichentabelle nicht angezeigt. Symbolfonts von Microsoft verwenden jedoch die folgende, einfache Regel:

```
Unicode-Wert = U+F000 + (in der Zeichentabelle angezeigter Zeichencode)
```

Für die Glyphe *smileface* ergibt dies den Unicode-Wert U+F04A.

- Der zugehörige Glyphname kann mit einem Fonteditor und ähnlichen Tools ermittelt werden. In unserem Beispiel ist es *smileface*.

Symbolzeichen mit PDFlib adressieren. Abhängig von den verfügbaren Informationen zum Zielzeichen können Sie die Glyphe *smileface* aus dem Font Wingdings auf verschiedenen Arten ansprechen:

- Wenn Sie den PUA-Unicode-Wert kennen, der dem Zeichen im Font zugeordnet ist, können Sie eine numerische Character-Referenz verwenden (siehe Abschnitt »Numerische Character-Referenzen«, Seite 114):

  

  Wenn Sie mit *textformat=utf8* arbeiten, können Sie die entsprechende Drei-Byte-UTF-8- Sequenz verwenden:

  \xEF\x81\x8A

  Unicode-Werte können nicht in der Kombination von *encoding=builtin* mit *textformat=bytes* verwendet werden.
- Wenn Sie den Zeichencode kennen, können Sie eine Bytewert-Referenz verwenden (siehe Abschnitt »Bytewert-Referenzen«, Seite 115):

  ```
  &.#x4A;
  ```

  In nicht Unicode-fähigen Sprachbindungen kann der Zeichencode direkt angegeben werden, wenn *encoding=builtin* und *textformat=bytes* ist:

  ```
  J
  \x4A
  ```

- Wenn Sie den Glyphnamen kennen, können Sie eine Glyphnamen-Referenz verwenden (siehe Abschnitt »Glyphnamen-Referenzen«, Seite 115):

  ```
  &.smileface;
  ```

  Glyphnamen können nicht in der Kombination von *encoding=builtin* mit *textformat=bytes* verwendet werden.

Tabelle 5.1 zeigt die Methoden für Unicode-fähige Sprachbindungen wie Java und .NET.

Tabelle 5.1 Adressierung der Glyphe `smileface` *im Font Wingdings bei Unicode-fähigen Sprachbindungen (z.B. Java)*

Encoding	*weitere Optionen*	*Eingabe-String*	*sichtbares Ergebnis auf der Seite*
unicode		*\uF04A* [1]	☺
	charref	**	☺
	charref	*&.#x4A;*	☺
	charref	*&.smileface;*	☺
		J [2]	*(nichts)*
	escapesequence	*\x4A*	*(nichts)*
builtin	*(siehe oben unter* encoding=unicode)		

1. Stringsyntax für U+F04A *in Java und vielen anderen Unicode-fähigen Sprachen*
2. Winansi-Zeichen für den Bytecode \x4A

Tabelle 5.2 zeigt die Methoden für nicht Unicode-fähige Sprachbindungen wie C.

Tabelle 5.2 Adressierung der Glyphe smileface *im Font Wingdings mit nicht Unicode-fähigen Sprachbindungen (z.B. C)*

Encoding	*Textformat*	*weitere Optionen*	*Eingabe-String*	*sichtbares Ergebnis auf der Seite*
unicode	utf16		*\xFo\x4A* [1]	☺
	utf8	charref	**	☺
		charref	*&.#x4A;*	☺
		charref	*&.smileface;*	☺
			ï•Š [2]	☺
		escapesequence[3]	*\xEF\x81\x8A* [4]	☺
			J [5]	*(nichts)*
		escapesequence	*\x4A*	*(nichts)*
	bytes	charref	**	☺
		bytes	*&.#x4A;*	☺
		charref	*&.smileface;*	☺
			J	*(nichts)*
		escapesequence	*\x4A*	*(nichts)*
builtin	utf16, utf8	*(siehe oben unter* encoding=unicode)		
	bytes	charref	**	🕮✂☒☞🗀🗐✌▭
		charref	*&.#x4A;*	🕮🖗✂☒🗐✌▭
		charref	*&.smileface;*	🕮🖗♦❍♓●♏♐♋♍♏▭
			J	☺
		escapesequence	*\x4A*	☺

1. Muss je nach Bytereihenfolge als \xF0\x4A *oder* \x4A\xF0 *angegeben werden; Beachten Sie, dass* \x *die Escape-Syntax von C darstellt*
2. Winansi-Zeichen für die Drei-Byte-Sequenz \xEF \x81 \x8A
3. Die Option escapesequence *wird nur benötigt, falls die Programmiersprache keine Syntax für direkte Bytewerte anbietet.*
4. Drei-Byte UTF-8-Sequenz für U+F04A
5. Winansi-Zeichen für den Bytecode \x4A

5.4.4 Suche nach Fonts

Quellen für Fontdaten. In PDFlib können Fonts explizit mit der Funktion *PDF_load_font( )* oder implizit durch Übergabe der Optionen *fontname* und *encoding* an verschiedene Textausgabefunktionen geladen werden. Zur Suche der Fontdaten können Sie den Originalnamen der Schrift verwenden oder mit beliebigen benutzerdefinierten Fontnamen arbeiten. Benutzerdefinierte Fontnamen sollten innerhalb eines Dokuments eindeutig sein. Der benutzerdefinierte Fontname kann mit *PDF_info_font( )* und dem Schlüsselwort *apiname* abgefragt werden.

Bei weiteren Aufrufen von *PDF_load_font( )* mit den gleichen Fontnamen wird dasselbe Font-Handle zurückgegeben, wenn alle Optionen den im ersten Funktionsaufruf übergebenen Optionen entsprechen. Andernfalls wird für denselben Fontnamen ein neues Font-Handle erzeugt. PDFlib bezieht Fontdaten aus folgenden Quellen:

- Fontdateien im Dateisystem oder virtuelle Fontdateien
- Systemschriften *(host fonts)* vo Windows oder OS X
- PDF-Standardfonts: diese umfassen eine kleine Zahl lateinischer Fonts und CJK-Fonts mit festen Namen
- Type-3-Fonts, die mit *PDF_begin_font( )* und verwandten Funktionen definiert wurden

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `fonts/font_resources`.

Mit der Option *enumeratefonts* kann PDFlib angewiesen werden, alle auf einem Suchpfad zugänglichen Fonts zu sammeln (siehe Abschnitt »Dateisuche und Ressourcenkategorie SearchPath«, Seite 66). Mit der Option *saveresources* kann die Liste der aktuellen PDFlib-Ressourcen im Dateisystem gespeichert werden:

```
/* Fontverzeichnis zum Suchpfad hinzufügen */
p.set_option("searchpath={{C:/fonts}}");

/* Auflisten aller Fonts im Suchpfad und Erzeugen einer UPR-Datei */
p.set_option("enumeratefonts saveresources={filename=C:/fonts/pdflib.upr}");
```

Aliasnamen für Fonts. Jeder Font kann beliebig viele Aliasnamen haben. Dies kann nützlich sein, wenn Fonts über einen künstlichen oder virtuellen Namen angefordert werden, der einem physikalischen Font zugeordnet sein muss. Aliasnamen für Fonts können mit der Ressourcenkategorie *FontnameAlias* erzeugt werden, wie das folgende Beispiel zeigt (siehe auch Tabelle 3.1, Seite 65):

```
p.set_option("FontnameAlias={sans Helvetica}");
```

Der Aliasname links kann beliebig gewählt werden und kann zum Laden des Fonts unter seinem neuen Aliasnamen verwendet werden. Der rechte Name muss eine gültiger API-Name des Fonts sein, zum Beispiel der Name eines Host-Fonts oder ein Font, der mit einer der Font-Ressourcenkategorien FontOutline zu einer Font-Ressource verbunden wurde.

Suchreihenfolge für Fonts. Der an PDFlib übergebene Fontname ist ein Name-String. Wenn der angegebene Fontname ein Aliasname ist, wird er durch den entsprechenden API-Fontnamen ersetzt. PDFlib verwendet diesen, um nach unterschiedlichen Fonttypen in der unten beschriebenen Reihenfolge zu suchen. Die Suche wird beendet, sobald bei einem der Schritte ein brauchbarer Font gefunden wurde:

- Der Fontname stimmt mit dem Namen eines CJK-Standardfonts überein, und das angegebene Encoding ist der Name einer voreingestellten CMap (siehe Abschnitt 6.5.5, »Standard-CJK-Fonts«, Seite 181).
- Der Fontname stimmt mit dem Namen eines zuvor im selben Dokument mit *PDF_begin_font( )* erzeugten Type-3-Fonts überein (siehe Abschnitt 5.1.8, »Type-3-Fonts«, Seite 120).
- Der Fontname stimmt mit dem Namen in einer Ressource *FontOutline* überein, die den Fontnamen mit dem Namen einer TrueType- oder OpenType-Fontdatei verbindet.
- Der Fontname stimmt mit dem Namen in einer Ressource *FontAFM* oder *FontPFM*, die den Fontnamen mit dem Namen einer Font-Metrikdatei vom Typ PostScript-Type-1 verbindet.
- Der Fontname stimmt mit dem Namen in einer Ressource *FontOutline* überein, die den Fontnamen mit dem Namen einer SVG-Fontdatei verbindet und mit keinem Namen in einer *HostFont*-Ressource übereinstimmt.
- Der Fontname stimmt mit dem Namen in einer Ressource *HostFont* überein, die den Fontnamen mit dem Namen eines auf dem System installierten Fonts verbindet.
- Der Fontname stimmt mit dem Namen eines lateinischen Standardfonts überein (siehe Abschnitt »Lateinische Standardfonts«, Seite 138).
- Der Fontname stimmt mit dem Namen eines auf dem System installierten Host-Fonts überein (siehe Abschnitt »Windows-Fontstilnamen«, Seite 141).
- Der Fontname stimmt mit dem Basisnamen einer Fontdatei überein (also dem Dateinamen ohne Suffix).

Wenn kein Font gefunden wurde, wird das Laden des Fonts mit folgender Fehlermeldung abgebrochen:

```
Font file (AFM, PFM, TTF, OTF etc.) or host font not found
```

Für weitere Informationen zu den Ressourcenkategorien siehe Abschnitt 3.1.3, »Ressourcenkonfiguration und Dateisuche«, Seite 65. In den folgenden Abschnitten finden Sie weiterführende Informationen zum Laden von Fonts der verschiedenen Fontklassen.

TrueType-, OpenType- und WOFF-Fonts. Der Fontname muss mit dem Namen der gewünschten Fontdatei über die Ressource *FontOutline* verbunden sein:

```
p.set_option("FontOutline={Arial=/usr/fonts/Arial.ttf}");
font = p.load_font("Arial", "unicode", "embedding");
```

Der Aliasname auf der linken Seite des Gleichheitszeichens (API-Name des Fonts) kann beliebig gewählt werden:

```
p.set_option("FontOutline={f1=/usr/fonts/Arial.ttf}");
font = p.load_font("f1", "unicode", "embedding");
```

Alternativ zur Laufzeitkonfiguration mit *PDF_set_option( )* kann die Ressource *FontOutline* in einer UPR-Datei gespeichert werden (siehe Abschnitt 3.1.3, »Ressourcenkonfiguration und Dateisuche«, Seite 65). Um absolute Dateinamen zu vermeiden, können Sie die Ressourcenkategorie *SearchPath* verwenden (auch diese kann alternativ in einer UPR-Datei gespeichert werden), zum Beispiel:

```
p.set_option("SearchPath={{/usr/fonts}}");
p.set_option("FontOutline={f1=Arial.ttf}");
font = p.load_font("f1", "unicode", "");
```

TrueType-Collection. Zur Auswahl eines Fonts aus einer TrueType-Collection (*TTC*, siehe Abschnitt 6.5.1, »Verwendung von CJK-Fonts vom Typ TrueType und OpenType«, Seite 176), geben Sie den Fontnamen direkt an:

```
p.set_option("FontOutline={MS-Gothic=msgothic.ttc}");
font = p.load_font("MS-Gothic", "unicode", "embedding");
```

Der Fontname wird mit den Namen aller Fonts in der TTC-Datei verglichen. Eine andere Möglichkeit besteht darin, den *n*-ten Font in einer TTC-Datei zu selektieren, indem Sie nach dem Fontnamen einen Doppelpunkt und dann die Zahl *n* angeben. Hierbei kann der API-Name des Fonts auf der linken Seite des Gleichheitszeichens beliebig gewählt werden:

```
p.set_option("FontOutline={f1=msgothic.ttc}");
font = p.load_font("f1:0", "unicode", "");
```

PostScript-Type-1-Fonts. Der Fontname muss mit dem Namen der gewünschten Font-Metrikdatei über eine der Ressourcenkategorien *FontAFM* oder *FontPFM* dem Typ der Metrikdatei entsprechend verbunden werden:

```
p.set_option("FontPFM={lucidux=LuciduxSans.pfm}");
font = p.load_font("lucidux", "unicode", "");
```

Soll ein PostScript-Font eingebettet werden, muss der Fontname zusätzlich mit der entsprechenden Zeichenbeschreibungsdatei (PFA oder PFB) über die Ressourcenkategorie *FontOutline* verbunden werden:

```
p.set_option("FontPFM={lucidux=LuciduxSans.pfm}");
p.set_option("FontOutline={lucidux=LuciduxSans.pfa}");
font = p.load_font("lucidux", "unicode", "embedding");
```

Beachten Sie, dass bei PostScript-Type-1-Fonts die Ressource *FontOutline* allein nicht ausreicht. Da immer eine Metrikdatei erforderlich ist, muss eine AFM- oder PFM-Datei vorhanden sein, damit der Font geladen werden kann.

Die Verzeichnisse, die nach Font-Metrikdateien und Zeichenbeschreibungsdateien durchsucht werden, können über die Ressourcenkategorie *SearchPath* angegeben werden.

Lateinische Standardfonts. PDF-Viewer unterstützen standardmäßig 14 Fonts, die immer vorhanden sind. PDFlib enthält alle Metrikdaten für die Standardschriften, so dass keine zusätzlichen Fontdateien erforderlich sind (es sei denn, der Font soll eingebettet werden). Die Standardschriften lauten:

Courier, Courier-Bold, Courier-Oblique, Courier-BoldOblique,
Helvetica, Helvetica-Bold, Helvetica-Oblique, Helvetica-BoldOblique,
Times-Roman, Times-Bold, Times-Italic, Times-BoldItalic,
Symbol, ZapfDingbats

Wenn ein Font nicht über Ressourcen mit einem Dateinamen verbunden ist, sucht PDFlib den Font in der Liste der lateinischen Standardfonts. Dieser Schritt wird über-

sprungen, wenn die Option *embedding* angegeben wird oder eine Ressource *FontOutline* für diesen Font vorhanden ist. Das folgende Codefragment lädt einen Standardfont ohne jede Konfiguration:

```
font = p.load_font("Times-Roman", "unicode", "");
```

Standardfonts in der internen Liste werden nie eingebettet. Um einen dieser Fonts einzubetten, müssen Sie eine Outline-Datei konfigurieren.

Host-Fonts (Systemfonts). Wenn ein Fontname nicht über Ressourcen mit einem Dateinamen verbunden ist, sucht PDFlib den Font unter den auf Windows oder OS X. installierten Fonts. Solche auf dem Betriebssystem installierten Fonts heißen *Host-Fonts* (Systemfonts). Namen von Host-Fonts müssen in ASCII kodiert sein. Unter Windows kann auch Unicode verwendet werden. Für weitere Informationen siehe Abschnitt »Windows-Fontstilnamen«, Seite 141. Zum Beispiel:

```
font = p.load_font("Verdana", "unicode", "");
```

Unter Windows kann an den Fontnamen nach einem Komma ein optionaler Fontstil angefügt werden (diese Syntax kann auch bei lateinischen Standardfonts verwendet werden):

```
font = p.load_font("Verdana,Bold", "unicode", "");
```

Um einen Host-Font mit dem Namen eines Standardfonts zu laden, muss der Fontname über die Ressourcenkategorie *HostFont* mit dem gewünschten Namen des Host-Fonts verbunden werden. Das folgende Codefragment zeigt, wie Metrik und Zeichenbeschreibungen des Host-Fonts *Symbol* und nicht die in PDFlib integrierten Daten verwendet werden:

```
p.set_option("HostFont={Symbol=Symbol}");
font = p.load_font("Symbol", "unicode", "embedding");
```

Der API-Name des Fonts auf der linken Seite des Gleichheitszeichens kann beliebig gewählt werden. Normalerweise wird der Name des Host-Fonts auf beiden Seiten des Gleichheitszeichens verwendet.

Suche mit Namenserweiterung für Fontdateien. Alle Fonttypen außer Type-3-Fonts können mit dem angegebenen Fontnamen als Basisnamen (ohne Dateiendung) einer Font-Metrikdatei oder Zeichenbeschreibungsdatei gesucht werden. Konnte PDFlib unter dem angegebenen Namen keinen Font finden, durchsucht es alle Einträge in der Ressourcenkategorie *SearchPath*, wobei es den Fontnamen bei jedem Suchlauf mit einer anderen Dateinamenserweiterung versieht. Der Suchalgorithmus verläuft wie folgt:

- Der Fontname wird um die folgenden Endungen ergänzt und mit jedem der resultierenden Namen die Suche nach der Font-Metrikdatei (oder der Fontdatei im Falle von TrueType- und OpenType-Fonts) erneut begonnen:

  ```
  .tte .ttf .otf .gai .woff .cef .afm .pfm .ttc, svg, svgz,
  .TTE .TTF .OTF .GAI .WOFF .CEF .AFM .PFM .TTC, SVG, SVGZ
  ```

- Soll ein PostScript-Font eingebettet werden, werden nacheinander die folgenden Endungen an den Fontnamen angefügt, um die Fontdatei zu ermitteln:

```
.pfa .pfb
.PFA .PFB
```

Wenn kein Font gefunden wurde, wird das Laden des Fonts mit folgender Fehlermeldung abgebrochen:

```
Font cannot be embedded (PFA or PFB font file not found)
```

- Alle der oben beschriebenen Namen werden erst so zur Suche herangezogen, wie sie konstruiert wurden und dann mit allen Verzeichnisnamen aus der Ressourcenkategorie *SearchPath*.

Damit findet PDFlib einen Font ohne jede manuelle Konfiguration, sofern der Name der Fontdatei aus dem Fontnamen sowie der dem Fonttyp entsprechenden Standard-Dateinamenserweiterung besteht und sich in einem der in *SearchPath* festgelegten Verzeichnisse befindet.

Die beiden folgenden Anweisungspaare haben denselben Effekt bei der Suche nach der passenden Fontdatei:

```
p.set_option("FontOutline={Arial=/usr/fonts/Arial.ttf}");
font = p.load_font("Arial", "unicode", "");
```

und

```
p.set_option("SearchPath={{/usr/fonts}}");
font = p.load_font("Arial", "unicode", "");
```

Standard-CJK-Fonts. Acrobat unterstützt verschiedene Standardfonts für CJK-Text. Weitere Informationen sowie eine Liste der Standard-CJK-Fonts finden Sie in Abschnitt 6.5.5, »Standard-CJK-Fonts«, Seite 181. PDFlib findet einen Standard-CJK-Font gleich zu Begin der Fontsuche, wenn der angegebene Fontname mit dem Namen eines Standard-CJK-Fonts übereinstimmt, wenn das angegebene Encoding dem Namen einer der vordefinierten CMaps entspricht und wenn die Option *embedding* nicht angegeben wurde. In der internen Liste gefundene Standard-CJK-Fonts werden nur eingebettet, wenn eine Zeichenbeschreibungsdatei konfiguriert wurde.

Type-3-Fonts. Type-3-Fonts müssen zur Laufzeit erstellt und deren Glyphen mit den PDFlib-Grafikfunktionen definiert werden (siehe Abschnitt 5.1.8, »Type-3-Fonts«, Seite 120). Wenn der an *PDF_begin_font( )* übergebene Fontname mit dem Fontnamen für *PDF_load_font( )* übereinstimmt, wird der Font am Anfang der Fontsuche ausgewählt.
Zum Beispiel:

```
p.begin_font("PDFlibLogoFont", 0.001, 0.0, 0.0, 0.001, 0.0, 0.0, "");

...
p.end_font();
...
font = p.load_font("PDFlibLogoFont", "logoencoding", "");
```

5.4.5 Host-Fonts unter Windows und OS X

Auf Windows und OS X hat PDFlib Zugriff auf alle TrueType-, OpenType- und PostScript-Fonts, die auf dem Betriebssystem installiert sind. Wir bezeichnen solche Fonts als *Host-Fonts* (Systemfonts). Statt die Fontdateien manuell zu konfigurieren, werden sie einfach im System installiert (meist durch bloßes Kopieren in das entsprechende Verzeichnis), und schon stehen sie PDFlib zur Verfügung.

Bei Host-Fonts ist unbedingt auf den korrekten Fontnamen einschließlich Groß- und Kleinschreibung zu achten. Da die Namen von entscheidender Bedeutung sind, werden im Folgenden Methoden zur Ermittlung der richtigen Fontnamen beschrieben. Weitere Informationen finden Sie in Abschnitt 5.1.5, »PostScript-Type-1-Fonts«, Seite 119, und Abschnitt 5.1.1, »TrueType-Fonts«, Seite 117. Die Suche nach Host-Fonts können Sie mit der Option *usehostfonts* von *PDF_set_option( )* deaktivieren.

Ermitteln der Namen von Host-Fonts unter Windows. Um den Namen eines installierten Fonts herauszufinden, doppelklicken Sie auf die Fontdatei. Der vollständige Fontname wird im Fenstertitel (Windows Vista/7/8) oder der ersten Zeile des entsprechenden Fensters (Windows XP) angezeigt. Teile des Fontnamens können je nach verwendeter Windows-Version lokalisiert sein. So kann zum Beispiel der gängige Bestandteil *Bold* auf einem deutschen System als *Fett* erscheinen. Um die Daten der Host-Fonts vom Windows-System zu beziehen, müssen Sie in PDFlib die übersetzte Variante des Fontnamens (zum Beispiel *Arial Fett)* oder den Fontstilnamen verwenden (siehe Abschnitt »Windows-Fontstilnamen«, Seite 141). Dagegen müssen Sie die generische (nicht lokalisierte) Variante des Fontnamens benutzen (zum Beispiel *Arial Bold)*, wenn Sie die Fontdaten direkt aus der Fontdatei abrufen möchten.

Hinweis *Sie können dieses Internationalisierungsproblem vermeiden, indem Sie Fontstilnamen (zum Beispiel »*`,Bold`*«, siehe unten) anstelle des lokalisierten Fontnamens verwenden.*

Zur genaueren Untersuchung von TrueType-Fonts können Sie sich die von Microsoft kostenlos erhältliche Software »font properties extension«[1] besorgen, die zahlreiche Einträge der TrueType-Tabellen eines Fonts in lesbarer Form anzeigt.

Windows-Fontstilnamen. Beim Laden von Host-Fonts unter Windows können Sie eine Funktion nutzen, die vom Font-Selektionsmechanismus von Windows zur Verfügung gestellt wird: Für das Gewicht und die Neigung lässt sich ein Stilname übergeben, zum Beispiel:

```
font = p.load_font("Verdana,Bold", "unicode", "");
```

Windows wird damit angewiesen, nach einer bestimmten fetten, kursiven oder anderen Variante des Basisfonts zu suchen. Aus den vorhandenen Fonts selektiert Windows denjenigen, der dem angeforderten Stil am nächsten kommt (es wird keine neue Fontvariante erstellt). Der von Windows selektierte Font kann vom angeforderten Font abweichen, welcher wiederum nicht mit dem Fontnamen im generierten PDF übereinstimmen muss; PDFlib hat keinerlei Kontrolle über die Fontselektion von Windows. Stilnamen für Fonts funktionieren außerdem nur bei Host-Fonts und nicht bei Fonts, die über eine Fontdatei auf der Festplatte spezifiziert werden.

1. Siehe www.microsoft.com/typography/property/TrueTypeProperty21.mspx

Zur Festlegung des Fontgewichts können folgende Schlüsselwörter (durch ein Komma getrennt) an den Basisnamen des Fonts angehängt werden:

```
none, thin, extralight, ultralight, light, normal, regular, medium,
semibold, demibold, bold, extrabold, ultrabold, heavy, black
```

Bei den Schlüsselwörtern wird zwischen Groß- und Kleinschreibung unterschieden. Das Schlüsselwort *italic* kann alternativ oder zusätzlich zu obigen Schlüsselwörtern angehängt werden. Zwei Stilnamen werden durch Komma voneinander getrennt, zum Beispiel:

```
font = p.load_font("Verdana,Bold,Italic", "unicode", "");
```

Numerische Werte für das Fontgewicht können als zusätzliche Alternative zu Stilnamen für Fonts verwendet werden:

```
0 (none), 100 (thin), 200 (extralight), 300 (light), 400 (normal), 500 (medium), 600
(semibold), 700 (bold), 800 (extrabold), 900 (black)
```

Im folgenden Beispiel wird die Variante *bold* eines Fonts ausgewählt:

```
font = p.load_font("Verdana,700", "unicode", "");
```

Hinweis *Windows-Stilnamen für Fonts sind nützlich, wenn Sie mit lokalisierten Fontnamen zu tun haben, da sie eine universelle Methode für den Zugriff auf Fontvariationen unabhängig von ihren lokalisierten Namen bieten.*

Font-Ersetzung unter Windows. Windows kann basierend auf bestimmten Registry-Einträgen Fonts automatisch ersetzen. Dies wirkt sich auch auf den Host-Font-Mechanismus von PDFlib aus, wird jedoch vollständig vom Windows-Betriebssystem kontrolliert. Wenn zum Beispiel der Font Helvetica angefordert wird, kann Windows entsprechend des folgenden Registry-Eintrags stattdessen Arial verwenden:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontSubstitutes
```

Weitere Informationen zur Font-Ersetzung von Windows können Sie der Microsoft-Dokumentation entnehmen.

Ermitteln von Namen für Host-Fonts unter OS X. Mit dem Hilfsprogramm *Font Book*, das mit OS X ausgeliefert wird, können Sie die Namen installierter Host-Fonts ermitteln. Zur systematischen Ermittlung der Fontnamen aller installierten Fonts empfehlen wir jedoch die Font Tool Suite[1] von Apple, eine Sammlung von Kommandozeilen-Programmen inklusive des Programms *ftxinstalledfonts* zur Bestimmung der exakten Namen aller installierten Fonts. PDFlib unterstützt verschiedene Varianten von Namen für Host-Fonts:

- »Eindeutige« Fontnamen *(unique)*, die sich in Unicode kodieren lassen (z.B. für ostasiatische Fonts). Um eindeutige Fontnamen zu ermitteln, führen Sie in einem Terminalfenster den folgenden Befehl aus (manchmal enthält die Ausgabe Einträge mit einem ':', die entfernt werden müssen):

  ```
  ftxinstalledfonts -u
  ```

1. See developer.apple.com/fonts

- PostScript-Fontnamen: Um diese zu ermitteln, führen Sie in einem Terminalfenster den folgenden Befehl aus:

```
ftxinstalledfonts -p
```

Mögliches Problem beim Zugriff auf Host-Fonts unter OS X. Beim Testen haben wir festgestellt, dass neu installierte Fonts für Anwendungen ohne eine grafische Benutzeroberfläche wie PDFlib manchmal erst dann erreichbar sind, wenn sich der Benutzer vom System ab- und wieder angemeldet hat.

5.4.6 Fallback-Fonts

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/starter_fallback`.

Den Schwächen von Fonts und Encodings begegnet PDFlib mit einem leistungsfähigen Mechanismus, den sogenannten Fallback-Fonts. Damit lässt sich in vielen Situationen die Textausgabe vereinfachen, da notwendige Fontänderungen automatisch von PDFlib durchgeführt werden. Dieser Mechanismus erweitert einen bestimmten Font (den sogenannten Basisfont) durch Einblenden von Glyphen aus anderen Fonts in den Basisfont. Genauer: die Fonts werden nicht wirklich verändert, vielmehr führt PDFlib alle notwendigen Fontänderungen in der PDF-Seitenbeschreibung automatisch durch. Fallback-Fonts bieten die folgenden Funktionen:

- Nach Glyphen, die im Basisfont fehlen, wird automatisch in einem oder mehreren Fallback-Fonts gesucht. Glyphen lassen sich also zu einem Font hinzufügen. Da mehrere Fallback-Fonts zum Basisfont hinzugefügt werden können, können Sie alle Unicode-Zeichen nutzen, für die mindestens ein Font eine passende Glyphe enthält.
- Glyphen aus dem Basisfont können mit Glyphen aus einem bestimmten Fallback-Font überschrieben, also ersetzt werden. Sie können einzelne Glyphen ersetzen oder sogar ganze Bereiche von Unicode-Zeichen bestimmen, die ersetzt werden sollen.

Größe und vertikale Position von Glyphen eines Fallback-Fonts können auf den Basisfont abgestimmt werden. Sogar der Basisfont selbst kann als Fallback-Font verwendet werden (mit demselben oder einem anderen Encoding). Damit lassen sich folgende Besonderheiten leicht implementieren:

- Mit dem Basisfont als Fallback-Font können Sie die Größe oder Position einiger oder aller Glyphen einiger oder aller Fonts anpassen.
- Sie können auch Zeichen außerhalb des eigentlichen Encodings des Basisfonts hinzufügen.

Fallback-Fonts werden durch die Option *fallbackfonts* zum Laden von Fonts gesteuert und betreffen alle Funktionen für die Textausgabe. Wie bei allen Optionen zum Laden von Fonts können Sie die Option *fallbackfonts* explizit mit *PDF_load_font( )* setzen oder implizit über Optionslisten zum Laden von Fonts. Da Sie für einen Basisfont mehr als einen Fallback-Font angeben können, benötigt die Option *fallbackfonts* eine Liste von Optionslisten (das heißt, Sie müssen einen zusätzlichen Satz von Klammern angeben).

Mit *PDF_info_font( )* lassen sich die Ergebnisse des Einsatzes von Fallback-Fonts abfragen (siehe Abschnitt 5.6.3, »Abfrage von Codepage-Abdeckung und Fallback-Fonts«, Seite 153).

Vorsichtsmaßnahmen. Beachten Sie bei der Arbeit mit Fallback-Fonts Folgendes:

- Nicht alle Fontkombinationen führen zu typografisch ansprechenden Ergebnissen. Sie sollten nur Fallback-Fonts verwenden, bei denen die Gestaltung der Glyphen zu der des Basis-Fonts passt.
- Optionen zum Laden von Fonts müssen in der Optionsliste *fallbackfonts* für Fallback-Fonts separat angegeben werden. Wenn für den Basisfont zum Beispiel *embedding* angegeben ist, heißt das nicht, dass die Fallback-Fonts automatisch mit eingebettet werden.
- Fallback-Fonts müssen korrekte Unicode-Werte enthalten, um erfolgreich verarbeitet werden zu können. Die Ersatzglyphen müssen die gleichen Unicode-Werte wie die zu ersetzenden Glyphen haben.
- Schrift-Shaping (Optionen *shaping, script, locale)* und OpenType-Funktionen (Optionen *features, script, language)* werden nur auf Glyphen innerhalb eines Fonts angewendet, aber nicht über Fontgrenzen hinweg auf Glyphen aus dem Fallback-Font und andere aus dem Basisfont.
- Vorsicht ist geboten beim Unterstreichen, Überstreichen und Durchstreichen von Text, sowie der Option *ascender* und ähnlichen typografischen Werten und gleichzeitigem Einsatz von Fallback-Fonts. Die im Basisfont definierte Stärke oder Position des Unterstrichs kann vom Fallback-Font abweichen. Dadurch können sich diese unerwünscht verändern. Eine einfache Abhilfe gegen solche Artefakte ist die Angabe eines einheitlichen Werts in den Optionen *underlineposition* und *underlinewidth* von *PDF_fit_textline( )* und *PDF_add/create_textflow( )*. Dieser Wert sollte so gewählt werden, dass er mit dem Basisfont und allen Fallback-Fonts funktioniert.

Die folgenden Abschnitte beschreiben einige wichtige Anwendungsfälle von Fallback-Fonts und zeigen die entsprechenden Optionslisten.

Hinzufügen mathematischer Zeichen zu einem Textfont. Als grobe Lösung bei fehlenden mathematischen Glyphen können Sie für die Option *fallbackfonts* die folgende Optionsliste zum Laden von Fonts verwenden, um mathematische Glyphen aus dem Font *Symbol* in einen Textfont aufzunehmen:

```
fallbackfonts={{fontname=Symbol encoding=unicode}}
```

Fontkombinationen für verschiedene Schriftsysteme. Manchmal ist die Schrift der Texteingabe nicht im Voraus bekannt. Zum Beispiel kann eine Datenbank lateinische, griechische und kyrillische Texte enthalten, die verfügbaren Fonts decken aber nur eine dieser Schriften ab. Statt das jeweilige Schriftsystem zu ermitteln und einen dazu passenden Font auszuwählen, können Sie eine Kombination aus mehreren Fonts konstruieren und damit die Obermenge aller Schriftsysteme abdecken. Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts lassen sich griechische und kyrillische Fonts zu einem lateinischen Font hinzufügen:

```
fallbackfonts={
 {fontname=Times-Greek encoding=unicode embedding forcechars={U+0391-U+03F5}}
 {fontname=Times-Cyrillic encoding=unicode embedding forcechars={U+0401-U+0490}}
}
```

Erweiterung von 8-Bit-Encodings. Auch wenn die Texteingabe auf ältere 8-Bit-Encodings beschränkt ist, ermöglicht Ihnen der Einsatz von Fallback-Fonts (bei denen der Basisfont selbst als Fallback-Font dient) und das Verfahren der Character-Referenzen von PDFlib, Zeichen außerhalb dieser älteren Encodings zu verwenden. Wenn Sie bei-

spielsweise den Font Helvetica mit *encoding=iso8859-1* geladen haben (das Euro-Zeichen fehlt in diesem Encoding), können Sie mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts die Euro-Glyphe zum Font hinzufügen:

```
fallbackfonts={{fontname=Helvetica encoding=unicode forcechars=euro}}
```

Da das Eingabe-Encoding kein Euro-Zeichen enthält, können Sie es nicht mit einem 8-Bit-Wert adressieren. Als einfache Abhilfe verwenden Sie deshalb am besten Character- oder Glyphnamen-Referenzen, zum Beispiel *€* (siehe Abschnitt 4.6.2, »Character-Referenzen«, Seite 114).

Verwendung von Euro-Glyphen aus anderen Fonts. Wenn der Basisfont keine Euro-Glyphe enthält, können Sie mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts die Euro-Glyphe aus einem anderen Font entnehmen:

```
fallbackfonts={{fontname=Helvetica encoding=unicode forcechars=euro textrise=-5%}}}
```

Mit der Unteroption *textrise* wird die Euro-Glyphe leicht nach unten verschoben.

Vergrößern von Glyphen in einem Font. Mit Fallback-Fonts lassen sich Glyphen in einem Font auch vergrößern, ohne die Fontgröße zu verändern. In diesem Beispiel wird ebenfalls der Basisfont als Fallback-Font verwendet. Mit dieser Funktion lassen sich unterschiedlich gestaltete Fonts optisch kompatibel machen, ohne die Fontgröße im Code anpassen zu müssen. Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts können Sie alle Glyphen im festgelegten Bereich auf 120 Prozent vergrößern:

```
fallbackfonts={
 {fontname=Times-Italic encoding=unicode forcechars={U+0020-U+00FF} fontsize=120%}
}
```

Hinzufügen vergrößerter Piktogramme. Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts können Sie ein Symbol aus dem Font ZapfDingbats hinzufügen:

```
fallbackfonts={
 {fontname=ZapfDingbats encoding=unicode forcechars=.a12 fontsize=150% textrise=-15%}
}
```

Mit den Unteroptionen *fontsize* und *textrise*e werden Größe und Position des Symbols an den Basisfont angepasst.

Glyphenersetzung in CJK-Fonts. Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts können Sie lateinische Zeichen im ASCII-Bereich durch solche aus einem anderen Font ersetzen:

```
fallbackfonts={
 {fontname=Courier-Bold encoding=unicode forcechars={U+0020-U+007E}}
}
```

Hinzufügen lateinischer Zeichen zu Arabischen Fonts. Für diesen Anwendungsfall siehe Abschnitt 6.4.5, »Arabische Textformatierung«, Seite 174.

Identifizierung fehlender Glyphen. Der frei verfügbare Font *Unicode BMP Fallback SIL* stellt statt der tatsächlichen Glyphe den hexadezimalen Wert jedes Unicode-Zeichens dar. Dieser Font kann für die Diagnose fontbezogener Probleme im Workflow sehr nützlich sein. Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts können Sie jeden Font um diesen speziellen Fallback-Font erweitern, um fehlende Zeichen zu visualisieren:

```
fallbackfonts={{fontname={Unicode BMP Fallback SIL} encoding=unicode}}
```

Hinzufügen von Gaiji-Zeichen zu einem Font. Für diesen Anwendungsfall siehe Abschnitt 6.5.2, »EUDC- und SING-Fonts für Gaiji-Zeichen«, Seite 177.

5.5 Fonteinbettung und Fontuntergruppen (Subsetting)

5.5.1 Fonteinbettung

PDF-Fonteinbettung und Fontersetzung in Acrobat. PDF-Dokumente können Fontdaten in verschiedenen Formaten einbinden, um eine korrekte Textanzeige sicherzustellen. Alternativ dazu kann ein Fontdeskriptor eingebettet werden, der nur die Zeichenmetriken (ohne die eigentlichen Zeichenbeschreibungen) sowie allgemeine Fontinformationen enthält. Wird ein Font nicht ins PDF-Dokument eingebettet, versucht Acrobat, ihn vom Zielsystem zu beziehen, sofern vorhanden und konfiguriert *(»Lokale Fonts verwenden«)* oder versucht, gemäß den Angaben des Fontdeskriptors einen Ersatzfont zu konstruieren. Die Verwendung von Ersatz-Fonts erzeugt lesbaren Text, aber die Glyphen sehen möglicherweise anders aus als der Original-Font. Ersatz-Fonts funktionieren auch nicht, wenn Shaping komplexer Schriftsysteme oder OpenType-Layout-Funktionen verwendet wurden. Aus diesen Gründen wird Fonteinbettung generell empfohlen, es sei denn, Sie wissen, dass die Anzeige der Dokumente auf den Zielsystemen auch ohne eingebettete Fonts akzeptabel ist. Solche PDF-Dokumente sind von Natur aus nicht portierbar, wenngleich sie in beschränktem Rahmen wie zum Beispiel beim innerbetrieblichen Dokumentenaustausch sinnvoll sein können, wo alle Fonts auf allen Rechnern verfügbar sind.

Fonteinbettung mit PDFlib. Die Fonteinbettung wird über die Option *embedding* beim Laden eines Fonts gesteuert (in manchen Situationen erzwingt PDFlib jedoch die Fonteinbettung).

```
font = p.load_font("WarnockPro", "winansi", "embedding");
```

Tabelle 5.3 zeigt die verschiedenen Situationen bei der Fontverwendung und die jeweils von PDFlib benötigten Font- und Metrikdateien. Um einen Standard- oder speziellen CJK-Font mit einer der Standard-CMaps zu verwenden, müssen darüber hinaus passende CMap-Dateien vorhanden sein (und eventuell eine CMap zur Unicode-Zuordnung für die jeweilige *character collection*, z.B. *Adobe-Japan1-UCS2)*.

Tabelle 5.3 Unterschiedliche Verwendung von Fonts und erforderliche Dateien

Verwendung von Fonts	***Metrikdatei erforderlich?***	***Fontdatei erforderlich?***
Einer der 14 Standardfonts	*nein*	*nur wenn* `embedding=true` *gesetzt ist und* `skipembedding={latincore}` *nicht gesetzt ist*
Unter Windows oder OS X installierte Host-Fonts vom Typ TrueType, OpenType oder Type 1	*nein*	*nein*
Nicht-Standardfonts vom Typ Type 1	*ja*	*nur wenn* `embedding=true`
TrueType-Fonts	*n/a*	*ja*
OpenType- und SING-Fonts	*n/a*	*ja*

Rechtliche Aspekte der Fonteinbettung. Es ist wichtig anzumerken, dass der bloße Besitz einer Fontdatei nicht unbedingt dazu berechtigt, diese in ein PDF-Dokument einzubetten, selbst wenn eine gültige Fontlizenz vorhanden ist. Zahlreiche Font-Anbieter lassen die Fonteinbettung nur eingeschränkt zu, manche verbieten die Einbettung in PDF vollständig, und wieder andere offerieren besondere Online- oder Einbettungslizenzen. Schließlich gibt es auch noch Font-Anbieter, die Einbettung nur bei Verwendung von Fontuntergruppen erlauben. Überprüfen Sie deshalb die rechtlichen Aspekte, bevor Sie Fonts mit PDFlib einbetten. PDFlib hält sich an Einbettungsbeschränkungen, falls diese in einem TrueType- oder OpenType-Font spezifiziert sind. Dies wird über ein Einbettungsflag bewerkstelligt, das auf *no embedding*[1] gesetzt sein kann. In diesem Fall kommt PDFlib der Aufforderung des Herstellers nach und weist jeden Versuch zurück, den Font einzubetten.

Die rechtliche Warnung oben sollte vor allem für Web-Fonts im Auge behalten werden, da die meisten Anbieter von Web-Fonts keine Einbettung solcher Fonts in PDF-Dokumenten zulassen.

5.5.2 Fontuntergruppen (Subsetting)

Um die Größe der PDF-Ausgabe zu reduzieren, ist PDFlib in der Lage, lediglich diejenigen Zeichen eines Fonts einzubetten, die im Dokument auch tatsächlich verwendet werden. Dieser Vorgang wird Bildung von Fontuntergruppen genannt. Dabei wird ein neuer Font mit entsprechend weniger Glyphen als im Originalfont erzeugt, so dass alle für die PDF-Anzeige überflüssigen Informationen weggelassen werden. Fontuntergruppen sind bei CJK-Fonts besonders wichtig. PDFlib unterstützt Untergruppen für folgende Fontformate:

- TrueType-Fonts
- OpenType-Fonts mit PostScript- oder TrueType-Zeichenbeschreibungen
- Type-3-Fonts (müssen besonders behandelt werden, siehe »Untergruppenbildung bei Type-3-Fonts«, Seite 149)

Wird ein Font, für den die Untergruppenbildung angefordert wurde, in einem Dokument verwendet, so merkt sich PDFlib, welche Zeichen in der Textausgabe verwendet wurden. Die Untergruppenbildung lässt sich auf verschiedene Art beeinflussen:

- Die Option *autosubsetting* bestimmt das Standardverhalten bezüglich Untergruppenbildung. Ist die Option gleich *true*, so wird die Untergruppenbildung für alle Fonts aktiviert, für die eine Untergruppenbildung möglich ist (eine Ausnahme bilden Type-3-Fonts, die besonders behandelt werden, siehe unten). Der Standardwert ist *true*.
- Bei *autosubsetting=true:* Die Option *subsetlimit* enthält einen Prozentwert. Werden in einem Dokument prozentual zur Gesamtzahl der Glyphen im Font mehr Glyphen verwendet, als durch den Prozentsatz vorgegeben ist, wird die Untergruppenbildung für den Font deaktiviert und dieser komplett eingebettet. Dies spart Rechenzeit, die Ausgabe wird jedoch etwas größer. Die folgende Fontoption setzt den Grenzwert der Untergruppe auf 75 %:

```
subsetlimit=75%
```

1. Genauer gesagt: wenn das Flag fsType in der OS/2-Tabelle des Fonts den Wert 2 hat.

Der Standardwert für *subsetlimit* beträgt 100 Prozent. Anders ausgedrückt: die Option *subsetting* von *PDF_load_font( )* wird immer berücksichtigt, außer der Client setzt das Limit explizit auf weniger als 100 Prozent.

- Bei *autosubsetting=true:* Mit der Option *subsetminsize* lässt sich die Untergruppenbildung für kleine Fontdateien vollständig deaktivieren. Ist die ursprüngliche Fontdatei kleiner als der Wert von *subsetminsize* in KB, wird die Untergruppenbildung für diesen Font deaktiviert.

Einbettung und Untergruppenbildung bei TrueType-Fonts. Wird ein TrueType-Font mit einem von *winansi* und *macroman* verschiedenen Encoding verwendet, wird er für die PDF-Ausgabe standardmäßig in einen CID-Font konvertiert. Bei Encodings, die nur Zeichen aus der Adobe Glyph List (AGL) enthalten, lässt sich dieses Verhalten unterbinden, indem die Option *autocidfont* auf *false* gesetzt wird.

Bestimmen der initialen Fontuntergruppe. Fontuntergruppen enthalten Zeichenbeschreibungen für alle in einem Dokument verwendeten Glyphen. Die erzeugten Fontuntergruppen variieren also je nach Dokument, da in jedem Dokument normalerweise ein unterschiedlicher Satz von Zeichen (und damit Glyphen) verwendet wird. Unterschiedliche Fontuntergruppen können beim Zusammenführen mehrerer kleiner Dokumente mit eingebetteten Fontuntergruppen zu einem großen Dokument zu Problemen führen: die eingebetteten Untergruppen können nicht entfernt werden, da sie alle unterschiedlich sind.

Für diesen Fall können Sie in PDFlib mit der Option *initialsubset* von *PDF_load_font( )* den initialen Inhalt einer Fontuntergruppe bestimmen. Während PDFlib entsprechend der erzeugten Textausgabe Glyphen zu einer standardmäßig leeren Fontuntergruppe hinzufügt, kann mit der Option *initialsubset* eine nicht leere Untergruppe bestimmt werden. Wenn zum Beispiel nur Textausgabe vom Typ Latin-1 erzeugt wird und der Font noch viele andere Glyphen enthält, können Sie den ersten Unicode-Block als initiale Untergruppe bestimmen:

```
initialsubset={U+0020-U+00FF}
```

Damit werden die Glyphen für alle Unicode-Zeichen im angegebenen Bereich in der Untergruppe aufgenommen. Wenn dieser Bereich so gewählt wurde, dass er den gesamten Text im generierten Dokument abdeckt, sind die erzeugten Fontuntergruppen in allen Dokumenten identisch. Wenn diese Dokumente später zu einem einzigen PDF-Dokument zusammengefasst werden, können die identischen Fontuntergruppen mit der Option *optimize* von *PDF_begin_document( )* entfernt werden.

Untergruppenbildung bei Type-3-Fonts. Bevor Type-3-Fonts in einem Dokument verwendbar sind, müssen sie definiert und eingebettet werden (da die Zeichenbreiten benötigt werden). Eine Untergruppenbildung ist aber erst möglich, nachdem alle Seiten erstellt wurden (da erst dann bekannt ist, welche Glyphen benutzt und damit in die Untergruppe aufgenommen werden). Um diesen Konflikt zu vermeiden, unterstützt PDFlib Type-3-Fonts, die nur Metrikdaten erhalten *(widths-only Type3 fonts)*. Um einen Type-3-Font als Untergruppe einzubetten, müssen Sie den Font in zwei Durchgängen definieren:

- Der erste Durchgang mit der Option *widthsonly* von *PDF_begin_font( )* erfolgt vor der eigentlichen Verwendung des Fonts und definiert lediglich die Font- und Zeichenmetriken; die Fontmatrix muss in *PDF_begin_font( )* und *wx* sowie die Bounding-Box

der Glyphen müssen in *PDF_begin_glyph_ext( ))* übergeben werden. Diese Werte müssen die Glyphen korrekt beschreiben. Für jede Glyphe sind *PDF_begin_glyph_ext( )* und *PDF_end_glyph( )* erforderlich, aber keinerlei Aufruf zur Definition der eigentlichen Glyphenform. Werden zwischen dem Anfang und Ende der Glyphendefinition weitere Funktionen aufgerufen, so haben sie keine Auswirkung auf die PDF-Ausgabe und lösen auch keine Exception aus.

- Der zweite Durchgang muss nach der Erstellung des gesamtes Texts in diesem Font erfolgen und definiert die Zeichenbeschreibungen oder Bitmaps. Die Font- und Zeichenmetriken werden ignoriert, da sie bereits aus dem ersten Durchgang bekannt sind. Nach der Erstellung der letzten Seite weiß PDFlib, welche Glyphen im Dokument verwendet werden und bettet nur die Beschreibungen derjenigen Zeichen ein, die zur Erstellung der Untergruppe benötigt werden.

In Durchgang 1 und 2 muss dieselbe Menge von Glyphen übergeben werden. Ein Type-3-Font mit Untergruppenbildung kann nur einmal mit *PDF_load_font( )* geladen werden.

Cookbook Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic `fonts/type3_subsetting`.

5.6 Abfragen von Fontinformationen

Mit *PDF_info_font( )* können Sie nützliche Informationen über Fonts, Encodings, Unicode und Glyphen abfragen. Je nach Abfragetyp kann ein gültiges Font-Handle als Option für diese Funktion erforderlich sein. In allen Beispielen unten werden die Variablen aus Tabelle 5.4 verwendet.

Tabelle 5.4 Variablen für die Beispiele mit PDF_info_font( )

Variable	***Kommentar***
`int uv;`	*Numerische Unicode-Werte; alternativ können Glyphnamen-Referenzen ohne die Verzierung '&' und ';' in den Optionslisten verwendet werden, z.B.* `unicode=euro`*. Für weitere Informationen siehe die Beschreibung des Datentyps der Optionsliste* `Unichar` *in der PDFlib-Referenz.*
`int c;`	*8-Bit-Zeichencode*
`int gid;`	*Glyph-ID*
`int cid;`	*CID-Wert*
`String gn;`	*Glyphname*
`int gn_idx;`	*String-Index für einen Glyphnamen; wenn* `gn_idx` *ungleich* `-1`*, kann der entsprechende String folgendermaßen abgefragt werden:* `gn = p.get_string(gn_idx, "");`
`String enc;`	*Name des Encodings*
`int font;`	*gültiges Font-Handle, erzeugt mit PDF_load_font()*

Wenn die gewünschte Kombination aus Schlüsselwort und Optionen nicht verfügbar ist, gibt *PDF_info_font( )* den Wert -1 aus. Dies muss durch die Client-Anwendung überprüft werden und kann verwendet werden, um zu überprüfen, ob eine erforderliche Glyphe in einem Font vorhanden ist.

Die Codezeilen in den Beispielen unten können auch einzeln verwendet werden, da sie nicht voneinander abhängig sind.

5.6.1 Fontunabhängige Abfrage von Encoding, Unicode und Glyphnamen

Abfrage von Encodings. Für eine Abfrage des Encodings ist kein gültiges Font-Handle erforderlich, das heißt, der Wert *-1* (in PHP: 0) kann an die Option *font* von *PDF_info_font( )* übergeben werden. Nur PDFlib intern bekannte Glyphnamen können mit *gn* übergeben werden, jedoch keine fontspezifischen Glyphnamen.

Abfrage des 8-Bit-Codes eines Unicode-Zeichens oder einer benannten Glyphe in einem 8-Bit-Encoding:

```
c = (int) p.info_font(-1, "code", "unicode=" + uv + " encoding=" + enc);
c = (int) p.info_font(-1, "code", "glyphname=" + gn + " encoding=" + enc);
```

Abfrage des Unicode-Werts eines 8-Bit-Codes oder einer benannten Glyphe in einem 8-Bit-Encoding:

```
uv = (int) p.info_font(-1, "unicode", "code=" + c + " encoding=" + enc);
uv = (int) p.info_font(-1, "unicode", "glyphname=" + gn + " encoding=" + enc);
```

Abfrage des registrierten Glyphnamens eines 8-Bit-Codes oder des Unicode-Werts in einem 8-Bit-Encoding:

```
gn_idx = (int) p.info_font(-1, "glyphname", "code=" + c + " encoding=" + enc);
gn_idx = (int) p.info_font(-1, "glyphname", "unicode=" + uv + " encoding=" + enc);

/* Abfrage des eigentlichen Glyphnamens mit dem String-Index */
gn = p.get_string(gn_idx, "");
```

Abfrage von Unicode- und Glyphnamen. Mit *PDF_info_font( )* lassen sich auch Abfragen durchführen, die von einem bestimmten 8-Bit-Encoding unabhängig sind, sich aber auf die Unicode-Werte und PDFlib intern bekannten Glyphnamen auswirken. Da diese Abfragen fontunabhängig sind, ist eine gültiges Font-Handle nicht erforderlich.

Abfrage des Unicode-Werts eines intern bekannten Glyphnamens:

```
uv = (int) p.info_font(-1, "unicode", "glyphname=" + gn + " encoding=unicode");
```

Abfrage des internen Glyphnamens eines Unicode-Werts:

```
gn_idx = (int) p.info_font(-1, "glyphname", "unicode=" + uv + " encoding=unicode");

/* Abfrage des eigentlichen Glyphnamens mit dem String-Index */
gn = p.get_string(gn_idx, "");
```

5.6.2 Fontspezifische Abfrage von Encoding, Unicode und Glyphnamen

Die folgenden Abfragen beziehen sich auf einen bestimmten Font, der durch ein gültiges Font-Handle ausgewählt werden muss. Mit der Variable *gn* können sowohl intern bekannte, als auch fontspezifische Glyphnamen übergeben werden. In allen Beispielen unten bedeutet der Ausgabewert -1, dass der Font die gesuchte Glyphe nicht enthält.

Abfrage des 8-Bit-Codes für Unicode-Wert, Glyph-ID, benannte Glyphe oder CID in einem Font, der mit einem 8-Bit-Encoding geladen wurde:

```
c = (int) p.info_font(font, "code", "unicode=" + uv);
c = (int) p.info_font(font, "code", "glyphid=" + gid);
c = (int) p.info_font(font, "code", "glyphname=" + gn);
c = (int) p.info_font(font, "code", "cid=" + cid);
```

Abfrage des Unicode-Werts für Code, Glyph-ID, benannte Glyphe oder CID in einem Font:

```
uv = (int) p.info_font(font, "unicode", "code=" + c);
uv = (int) p.info_font(font, "unicode", "glyphid=" + gid);
uv = (int) p.info_font(font, "unicode", "glyphname=" + gn);
uv = (int) p.info_font(font, "unicode", "cid=" + cid);
```

Abfrage der Glyph-ID für Code, Unicode-Wert, benannte Glyphe oder CID in einem Font:

```
gid = (int) p.info_font(font, "glyphid", "code=" + c);
gid = (int) p.info_font(font, "glyphid", "unicode=" + uv);
gid = (int) p.info_font(font, "glyphid", "glyphname=" + gn);
gid = (int) p.info_font(font, "glyphid", "cid=" + cid);
```

Abfrage der Glyph-ID für Code, Unicode-Wert oder benannte Glyphe in einem Font in Bezug auf ein beliebiges 8-Bit-Encoding:

```
gid = (int) p.info_font(font, "glyphid", "code=" + c + " encoding" + enc);
gid = (int) p.info_font(font, "glyphid", "unicode=" + uv + " encoding=" + enc);
gid = (int) p.info_font(font, "glyphid", "glyphname=" + gn + " encoding=" + enc);
```

Abfrage des fontspezifischen Namens einer Glyphe unter Angabe von Code, Unicode-Wert, Glyph-ID oder CID:

```
gn_idx = (int) p.info_font(font, "glyphname", "code=" + c);
gn_idx = (int) p.info_font(font, "glyphname", "unicode=" + uv);
gn_idx = (int) p.info_font(font, "glyphname", "glyphid=" + gid);
gn_idx = (int) p.info_font(font, "glyphname", "cid=" + cid);

/* Abfrage des eigentlichen Glyphnamens mit dem String-Index */
gn = p.get_string(gn_idx, "");
```

Überprüfen der Glyphenverfügbarkeit. Mit *PDF_info_font( )* können Sie überprüfen, ob ein bestimmter Font die für Ihre Anwendung benötigten Glyphen enthält. Der folgende Code überprüft zum Beispiel, ob die Euro-Glyphe in einem Font enthalten ist:

```
/* Hier könnte auch "unicode=U+20AC" verwendet werden */
if (p.info_font(font, "code", "unicode=euro") == -1)
{
        /* keine Glyphe für Euro-Zeichen im Font verfügbar */
}
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `fonts/glyph_availability`.

Alternativ dazu können Sie mit *PDF_info_textline( )* die Anzahl der im übergebenen Textstring nicht zugeordneten Zeichen überprüfen, d.h. die Anzahl der Zeichen im String, für die keine passende Glyphe im Font verfügbar ist. Das folgende Codefragment fragt die Ergebnisse zu einem String ab, der lediglich ein einzelnes Euro-Zeichen enthält (das als Glyphnamen-Referenz ausgedrückt wird). Wird ein nicht zugeordnetes Zeichen gefunden, so bedeutet dies, dass der Font keine Glyphe für das Euro-Zeichen enthält:

```
String optlist = "font=" + font + " charref";

if (p.info_textline("&euro;", "unmappedchars", optlist) == 1)
{
        /* keine Glyphe für Euro-Zeichen im Font verfügbar */
}
```

5.6.3 Abfrage von Codepage-Abdeckung und Fallback-Fonts

Mit *PDF_info_font( )* lässt sich prüfen, ob ein Font zum Erzeugen von Textausgabe in einer bestimmten Sprache oder Schrift geeignet ist, sofern die für den Text benötigte Codepage bekannt ist. Die Codepage-Abdeckung ist in der OS/2-Tabelle des Fonts kodiert. Beachten Sie, dass der Font-Designer selbst entscheiden muss, was genau es bedeutet, dass ein Font eine bestimmte Codepage unterstützt. Selbst wenn ein Font angeblich eine bestimmte Codepage unterstützt, muss dies nicht unbedingt heißen, dass er Glyphen für alle Zeichen in dieser Codepage enthält. Wenn Sie genauere Informationen zur Abdeckung benötigen, können Sie die Verfügbarkeit aller benötigten Zeichen abfragen, siehe Abschnitt 5.6.2, »Fontspezifische Abfrage von Encoding, Unicode und

Glyphnamen«, Seite 152.

Prüfen, ob ein Font eine Codepage unterstützt. Das folgende Codefragment prüft, ob ein Font eine bestimmte Codepage unterstützt:

```
String cp="cp1254";

result = (int) p.info_font(font, "codepage", "name=" + cp);

if (result == -1)
        System.err.println("Codepage-Abdeckung unbekannt");
else if (result == 0)
        System.err.println("Codepage durch diesen Font nicht unterstützt");
else
        System.err.println("Codepage durch diesen Font unterstützt");
```

Abruf einer Liste aller unterstützten Codepages. Mit dem folgenden Codefragment lässt sich eine Liste aller Codepages abrufen, die von einem TrueType- oder OpenType-Font unterstützt werden:

```
cp_idx = (int) p.info_font(font, "codepagelist", "");

if (cp_idx == -1)
        System.err.println("Codepage list unknown");
else
{
        System.err.println("Codepage list:");
        System.err.println(p.get_string(cp_idx, ""));
}
```

Dies erzeugt die folgende Liste für den gängigen Font Arial:

```
cp1252 cp1250 cp1251 cp1253 cp1254 cp1255 cp1256 cp1257 cp1258 cp874 cp932 cp936 cp949
cp950 cp1361
```

Abfrage von Ersatzglyphen. Mit *PDF_info_font()* können Sie die Ergebnisse des Einsatzes von Fallback-Fonts abfragen (siehe Abschnitt 5.4.6, »Fallback-Fonts«, Seite 143). Mit dem folgenden Codefragment lässt sich der Name des Basis- oder Fallback-Fonts bestimmen, der zur Darstellung des jeweiligen Unicode-Zeichens verwendet wurde:

```
result = p.info_font(basefont, "fallbackfont", "unicode=U+03A3");
/* Bei result==basefont wurde der Basisfont verwendet und kein Fallback-Font benötigt */
if (result == -1)
{
        /* Zeichen kann weder mit Basisfont noch mit Fallback-Fonts dargestellt werden */
}
else
{
        idx = p.info_font(result, "fontname", "api");
        fontname = p.get_string(idx, "");
}
```

6 Textausgabe

6.1 Methoden der Textausgabe

PDFlib unterstützt die Textausgabe auf mehreren Ebenen:

- Einfache Textausgabe mit *PDF_show( )* und ähnlichen Funktionen;
- Einzeilige Textausgabe mit *PDF_fit_textline( );* Diese Funktion unterstützt auch Text auf einem Pfad.
- Mehrzeilige Textausgabe formatiert mit Textflow *(PDF_fit_textflow( )* und ähnlichen Funktionen); Mit dem Textflow-Formatierer kann Text auch innerhalb oder außerhalb vektorbasierter Formen umbrochen werden.
- Text in Tabellen; der Tabellenformatierer unterstützt Inhalte vom Typ Textline und Textflow in Tabellenzellen.

Einfache Textausgabe. Mit Funktionen wie *PDF_show( )* kann Text ohne Formatierungshilfsmittel an einer bestimmten Stelle auf der Seite platziert werden. Dies empfiehlt sich nur bei Anwendungen mit sehr geringen Anforderungen an die Textausgabe (z.B. reine Textdateien nach PDF konvertieren), oder bei Anwendungen mit vollständigen Informationen zur Textplatzierung (z.B. ein Treiber, der eine Datei in einem anderen Format nach PDF konvertiert). Das folgende Codefragment erzeugt einfache Textausgabe:

```
font = p.load_font("Helvetica", "unicode", "");
p.setfont(font, 12);
p.set_text_pos(50, 700);
p.show("Hello world!");
p.continue_text("(says Java)");
```

Formatierte einzeilige Textausgabe mit Textlines. *PDF_fit_textline( )* erzeugt Textausgabe, die aus einzelnen Zeilen besteht und eine Vielzahl von Formatierungsmöglichkeiten bietet. Die Position der einzelnen Textzeilen muss dabei durch die Client-Anwendung bestimmt werden.

Das folgende Codefragment erzeugt Textausgabe mithilfe von Textlines. Da Font, Encoding und Fontgröße als Optionen angegeben werden können, muss *PDF_load_font( )* vorab nicht aufgerufen werden:

```
p.fit_textline(text, x, y, "fontname=Helvetica encoding=unicode fontsize=12");
```

Für weitere Informationen siehe Abschnitt 8.1, »Platzieren und Einpassen von einzeiligem Text«, Seite 215.

Mehrzeilige Textausgabe mit Textflow. Mit *PDF_fit_textflow( )* lässt sich Textausgabe mit einer beliebigen Anzahl von Zeilen erzeugen und Text über mehrere Spalten oder Seiten hinweg platzieren. Der Textflow-Formatierer unterstützt umfangreiche Formatierungsfunktionen. Das folgende Codefragment erzeugt Textausgabe mithilfe von Textflow:

```
tf = p.add_textflow(tf, text, optlist);
result = p.fit_textflow(tf, llx, lly, urx, ury, optlist);
p.delete_textflow(tf);
```

Für weitere Informationen siehe Abschnitt 8.2, »Mehrzeilige Textflows«, Seite 224.

Text in Tabellen. Mit Textlines und Textflows lässt sich Text auch in Tabellenzellen platzieren. Für weitere Informationen siehe Abschnitt 8.3, »Tabellenformatierung«, Seite 245.

6.2 Textmetrik und Textvarianten

6.2.1 Font- und Zeichenmetriken

Textposition. PDFlib verwaltet eine aktuelle Textposition, die unabhängig von der aktuellen Position beim Zeichnen von Grafik ist. Die aktuelle Textposition kann über die Option *textx/texty* und die aktuelle Textposition über *currentx/currenty* abgefragt werden.

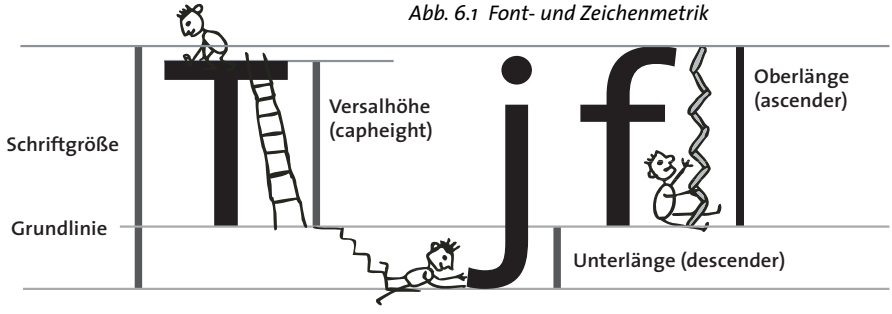
Glyphenmetrik. PDFlib verwendet dasselbe System für Glyphen- und Fontmetrik wie PostScript und PDF. Es soll hier kurz beschrieben werden.

Die Fontgröße, die von PDFlib-Anwendern angegeben werden muss, ist der Abstand zwischen zwei aufeinander folgenden Textzeilen, der minimal erforderlich ist, damit Zeichen nicht überlappen. Die Fontgröße ist gewöhnlich höher als die einzelnen Zeichen des Fonts, da sie auch Ober- und Unterlängen und möglicherweise einen Zwischenraum zwischen den Zeilen umfasst.

Der Zeilenabstand *(leading)* bezeichnet den vertikalen Abstand zwischen den Grundlinien benachbarter Textzeilen. Er wird standardmäßig auf den Wert der Fontgröße gesetzt. Die Versalhöhe *(capheight)* bezeichnet die Höhe von Großbuchstaben wie *T* oder *H* in den meisten lateinischen Fonts. Die x-Höhe *(xheight)* bezeichnet die Höhe von Kleinbuchstaben wie *x* in den meisten lateinischen Fonts. Die Oberlänge *(ascender)* bezeichnet die Höhe von Kleinbuchstaben wie *f* oder *d* in den meisten lateinischen Fonts. Die Unterlänge *(descender)* ist der Abstand von der Grundlinie zum unteren Ende von Kleinbuchstaben wie *j* oder *p* in den meisten lateinischen Fonts. Sie ist in der Regel negativ. Die Werte für x-Höhe, Versalhöhe, Oberlänge und Unterlänge werden als Bruchteil der Fontgröße gemessen und müssen deshalb vor der Verwendung mit der nötigen Fontgröße multipliziert werden.

Die Property *gaplen* ist nur in TrueType- und OpenType-Fonts verfügbar (bei anderen Fontformaten wird sie geschätzt). Der Wert für *gaplen* wird aus der Fontdatei gelesen und spezifiziert die Differenz zwischen dem empfohlenen Abstand zwischen Grundlinien und der Summe von Ober- und Unterlänge.

Abb. 6.1 Font- und Zeichenmetrik



PDFlib muss manchmal einen oder mehrere der Werte schätzen, da sie in der Font- oder Metrikdatei nicht immer vorhanden sind. Um herauszufinden, ob tatsächliche oder geschätzte Werte verwendet werden, können Sie mit *PDF_info_font( )* und der Option *faked* die x-Höhe *(xheight)* ermitteln. Die Zeichenmetriken für einen bestimmten Font können von PDFlib wie folgt abgefragt werden:

```
font = p.load_font("Times-Roman", "unicode", "");

capheight = p.info_font(font, "capheight", "");
ascender = p.info_font(font, "ascender", "");
descender = p.info_font(font, "descender", "");
xheight = p.info_font(font, "xheight", "");
```

Hinweis *Die Position und Größe von hochgestellten und tiefgestellten Zeichen können von PDFlib nicht abgefragt werden.*

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `fonts/font_metrics_info`.

CPI-Berechnungen. Die meisten Fonts besitzen variable Zeichenbreiten, nur bei den sogenannten äquidistanten (nichtproportionalen) Fonts ist die Laufweite aller Zeichen gleich. Um die PDF-Fontmetrik auf die Bemaßung Zeichen pro Zoll *(characters per inch, CPI)* abzubilden, die häufig beim Hochleistungsdruck verwendet wird, mögen einige Rechenbeispiele für den äquidistanten Font Courier hilfreich sein. In Courier haben alle Zeichen eine Breite von 600 Einheiten, bezogen auf den vollen Zeichenbereich von 1000 Einheiten pro Punkt (dieser Wert lässt sich aus der entsprechenden AFM-Metrikdatei ermitteln). Bei Text der Größe 12 Punkt haben zum Beispiel alle Zeichen eine absolute Breite von

```
12 Punkt * 600/1000 = 7,2 Punkt
```

bei einem optimalen Zeilenabstand von 12 Punkt. Da ein Zoll (inch) aus 72 Punkt besteht, passen genau 10 Zeichen von 12 Punkt Courier in einen Zoll. Mit anderen Worten stellt 12 Punkt Courier eine 10-cpi-Schrift dar. Für 10 Punkt Text beträgt die Zeichenbreite 6 Punkt, was einen Font von 72/6 = 12 cpi ergibt; 8 Punkt Courier ergibt 15 cpi.

6.2.2 Kerning

Manche Zeichenkombinationen sehen unter Umständen nicht sehr gut aus. Beispielsweise sieht zweimal *V* hintereinander manchmal wie ein *W* aus. Ebenso muss der Abstand zwischen *T* und *e* oft verringert werden, damit kein hässlicher Leerraum entsteht. Dieses Ausgleichen wird als Unterschneidung oder Kerning bezeichnet. Viele Fonts enthalten umfangreiche Kerning-Informationen über die Anpassung des Abstands bei kritischen Buchstabenkombinationen. PDFlib bezieht die Kerning-Informationen aus folgenden Quellen:

- TrueType- und OpenType-Fonts: Kerning-Paare aus der Tabelle *kern;*
- OpenType-Fonts: auf Paaren und Klassen basierende Kerning-Informationen ermittelt über die Funktion *kern* und die Tabelle *GPOS;*
- PostScript-Type-1-Fonts: Kerning-Paare aus den AFM- und PFM-Dateien;
- SVG-Fonts: Kerning-Paare aus dem Element *hkern;*
- Kerning-Paare für die PDF-Basisfonts werden von PDFlib intern bereitgestellt.

Abb. 6.2 Kerning

Kerning wird über die Fontoption *readkerning* und die Textoption *kerning* gesteuert. Standardmäßig ist Kerning aktiviert.

Eine temporäre Deaktivierung von Kerning kann zum Beispiel bei Tabellenziffern sinnvoll sein, bei denen die Kerningdaten aus bestimmten Ziffernpaaren bestehen und so keine einheitliche Anordnung der Zahlen möglich ist. Beachten Sie, dass es in modernen TrueType- und OpenType-Fonts besondere Ziffern für diesen Zweck gibt, die mit den Layout-Funktionen *Tabular Figures* und der Option *features={tnum}* verwendet werden können.

Kerning erfolgt zusätzlich zu Zeichenabstand, Wortabstand und eventuell aktivierter horizontaler Skalierung. In PDFlib gibt es keine Beschränkung für die Anzahl der Kerning-Paare in einem Font.

6.2.3 Textvariationen

Simulation von Fettschrift. PDFlib bietet einen Mechanismus zur Erzeugung von unechtem fetten Text. Dieser lässt sich für einzelne Textstrings mit der Option *fakebold* steuern.

Diese Methode simuliert einen fetten Font mittels Durchziehen der Glyphenkonturen; bei Type-3-Fonts wird der Text mit verschiedenen Offsets mehrfach platziert. Zur Auszeichnung sollten Sie aber unbedingt richtige Fettschrift verwenden, denn die Option *fakebold* erzeugt im Vergleich nur minderwertige Textausgabe und die Textextraktion kann unerwünschte Ergebnisse liefern.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `fonts/simulated_fontstyles`.

Hinweis *Mit der Fontoption* `fontstyle=bold[italic]` *lässt sich die Simulation mit fakebold für den gesamten in einem bestimmten Font formatierten Text aktivieren.*

Simulation kursiver Fonts. Wenn nur der reguläre Font verfügbar ist, lässt sich ein kursiver Font mit der Option *italicangle* simulieren. Dabei wird ein unechter kursiver Font erzeugt, indem der reguläre Font in einem vom Benutzer festgelegten Winkel geneigt wird. Negative Werte neigen dabei den Text nach rechts. Es sei jedoch darauf hingewiesen, dass ein echter Kursivfont eine wesentlich gefälligere Ausgabe ergibt. Ist dieser

nicht vorhanden, kann er mit der Option *italicangle* simuliert werden. Nützlich ist dies bei CJK-Fonts. Übliche Werte für die Option *italicangle* liegen zwischen -12 und -15 Grad.

Hinweis *Mit der Fontoption* `fontstyle=bold[italic]` *lässt sich die Simulation mit* `fakebold` *für den gesamten in einem bestimmten Font formatierten Text aktivieren.*

Hinweis *PDFlib passt die Glyphenbreite nicht auf die neue Bounding-Box der geneigten Glyphe an. Bei Blocksatz können die kursiven Glyphen dann beispielsweise über die Fitbox hinausragen.*

Schattentext. Mit PDFlib lässt sich ein Schatteneffekt erzeugen, indem mehrere Instanzen des selben Textes immer leicht versetzt platziert werden. Hierzu kann die Option *shadow* von *PDF_fit_textline( )* und *PDF_add/create_textflow( )* verwendet werden. Schattenfarbe und Schattenposition in Bezug zum Haupttext und Grafikoptionen können in Unteroptionen festgelegt werden.

Unterstreichen, Überstreichen und Durchstreichen von Text. PDFlib kann Linien unter, über oder auf Text zeichnen. Die Breite des Strichs und sein Abstand zur Grundlinie werden der Metrikdatei des Fonts entnommen. Außerdem fließen in die Berechnung der Strichstärke die aktuellen Werte des horizontalen Skalierungsfaktors sowie der Textmatrix ein. Unter-, Über- und Durchstreichen lässt sich mit den entsprechenden Optionen für *PDF_set_textoption( )* und den jeweiligen Textausgabefunktionen ein- und ausschalten. Mit den Optionen *underlineposition* und *underlinewidth* können genauere Einstellungen vorgenommen werden.

Die Option *strokecolor* wird zum Zeichnen der Linien verwendet; die aktuelle Option für Linienenden wird ignoriert. Warnung für Ästheten: In den meisten Fonts werden beim Unterstreichen Unterlängen berührt und beim Überstreichen diakritische Zeichen über den Oberlängen.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/starter_textline`.

Modi der Textdarstellung. PDFlib unterstützt mehrere Darstellungsmodi *(text rendering modes)*, die das Erscheinungsbild von Text beeinflussen. Dazu gehört Text, der durch Umrisslinien dargestellt wird, sowie die Möglichkeit, Text als Beschneidungspfad zu verwenden. Text lässt sich auch unsichtbar darstellen. Damit lässt sich zum Beispiel Text gut auf eingescannten Bildern platzieren, so dass er zwar indiziert und durchsucht werden kann, aber nicht direkt sichtbar ist. Die Darstellungsmodi werden in der *PDFlib-Referenz* beschrieben und lassen sich mit der Option *textrendering* einstellen.

Beim Zeichnen von Umrisslinien werden Zustandsoptionen für Text wie *linewidth* oder *color* auf die Glyphe angewendet. Der Darstellungsmodus hat keine Auswirkungen auf Text in einem Type-3-Font.

Cookbook *Vollständige Codebeispiele hierzu finden Sie in den Cookbook-Topics* `text_output/text_as_clipping_path` *und* `text_output/invisible_text`.

Textfarbe. Text wird mit der aktuellen Füllfarbe dargestellt, die mit der Option *fillcolor* gesetzt werden kann. Bei einem von 0 verschiedenen Darstellungsmodus wirken sich *strokecolor* bzw. *fillcolor* abhängig vom gewählten Darstellungsmodus auf den Text aus.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/starter_textline`.

6.3 OpenType-Layoutfunktionen

Cookbook *Vollständige Codebeispiele hierzu finden Sie in den Cookbook-Topics* `text_output/starter_opentype` *und* `font/opentype_feature_tester`.

6.3.1 Unterstützte OpenType-Layoutfunktionen

PDFlib unterstützt verbesserte Textausgabe anhand zusätzlicher Informationen in einigen Fonts. Diese Fonterweiterungen werden OpenType-Layoutfunktionen *(features)* genannt. Zum Beispiel kann ein Font die Funktion *liga* enthalten mit der Information, dass die Glyphen *f, f* und *i* zu einer Ligatur kombiniert werden können. Andere häufige Beispiele sind Kapitälchen in der Funktion *smcp*, also Großbuchstaben, die kleiner als die regulären Großbuchstaben sind, und Mediävalziffern in der Funktion *onum* mit Ober-und Unterlängen (im Gegensatz zu Versalziffern, die immer auf der Grundlinie platziert sind). Obwohl Ligaturen eine sehr häufige OpenType-Funktion sind, sind sie nur eine von vielen Dutzend möglichen Funktionen. Eine Übersicht über die OpenType-Format- und Funktionstabellen finden Sie unter

```
www.microsoft.com/typography/developers/opentype/default.htm
```

PDFlib unterstützt die folgenden Gruppen von OpenType-Features:

- OpenType-Funktionen für westliche Typographie (siehe Tabelle 6.1); diese werden mit der Option *features* gesteuert.
- OpenType-Funktionen für chinesische, japanische und koreanische Textausgabe (siehe Tabelle 6.6); diese werden ebenfalls mit der Option *features* gesteuert; für weitere Informationen siehe Abschnitt 6.5.3, »OpenType-Layoutfunktionen für erweiterte CJK-Textausgabe«, Seite 178.
- OpenType-Funktionen für das Shaping komplexer Schriftsysteme und vertikale Textausgabe; diese werden automatisch entsprechend der Optionen *shaping* und *script* ausgewertet (siehe Abschnitt 6.4, »Ausgabe komplexer Schriftsysteme«, Seite 167). Die Funktion *vert* wird über die Fontoption *vertical* gesteuert.
- OpenType-Tabellen für Kerning; PDFlib behandelt Kerning jedoch nicht als OpenType-Funktion, da Kerning-Informationen auch mit anderen Mitteln als OpenType-Tabellen dargestellt werden können. Verwenden Sie zur Steuerung von Kerning stattdessen die Fontoption *readkerning* und die Textoption *kerning* (siehe Abschnitt 6.2.2, »Kerning«, Seite 157).

Für weitere Informationen zu OpenType-Layoutfunktionen siehe

```
www.microsoft.com/typography/otspec/featuretags.htm
```

Bestimmen von OpenType-Funktionen. Sie können OpenTyp-Funktionstabellen mit den folgenden Tools bestimmen:

- Mit dem Fonteditor FontLab lassen sich Fonts erzeugen und bearbeiten. Mit der kostenlosen Demo-Version *(www.fontlab.com)* lassen sich OpenType-Funktionen anzeigen.
- Mit der kostenlosen Anwendung DTL OTMaster Light *(www.fonttools.org)* können Fonts inklusive ihrer OpenType-Funktionstabellen dargestellt und analysiert werden.

- Mit der kostenlosen »font properties extension«[1] von Microsoft (nur für 32-Bit-Systeme erhältlich) kann man eine Liste von verfügbaren OpenType-Funktionen in einem Font anzeigen lassen (siehe Abbildung 6.3).
- Mit der Schnittstelle *PDF_info_font( )* von PDFlib lassen sich auch unterstützte OpenType-Funktionen abfragen (siehe Abschnitt »Abfrage von OpenType-Funktionen per Programm«, Seite 165).

Tabelle 6.1 Unterstützte OpenType-Funktionen für westliche Typografie (Funktionen für CJK-Text siehe Tabelle 6.6)

Schlüsselwort	Name	Beschreibung
`_none`	*all features disabled*	*Deaktiviert alle OpenType-Funktionen aus Tabelle 6.1 und Tabelle 6.6.*
`afrc`	*alternative fractions*	*Ersetzt durch Schrägstrich getrennte Ziffern durch alternative Formen*
`c2pc`	*petite capitals from capitals*	*Wandelt Großbuchstaben in kleinere Kapitälchen (petite caps) um.*
`c2sc`	*small capitals from capitals*	*Wandelt Großbuchstaben in Kapitälchen um.*
`calt`	*contextual alternates*	*Ersetzt Standard-Glyphen durch alternative Formen, die sich besser miteinander verbinden lassen; wird in Schreibschriften verwendet, in denen manche oder alle Glyphen miteinander verbunden sind.*
`case`	*case-sensitive forms*	*Verschiebt verschiedene Satzzeichen auf eine Position, die zu einer Sequenz von Großbuchstaben oder Versalziffern passt; wandelt auch Mediävalziffern in Versalziffern um.*
`ccmp`	*glyph composition/ decomposition*	*Um die Anzahl der alternativen Glyphen zu minimieren, ist es manchmal erwünscht, ein Zeichen in zwei Glyphen zu zerlegen oder zur besseren Verarbeitung der Glyphen zwei Zeichen zu einer einzelnen Glyphe zusammenzufügen. Dieses Feature erlaubt beides.*
`clig`	*contextual ligatures*	*Ersetzt aus typografischen Gründen eine Glyphensequenz durch eine einzelne Glyphe. Anders als andere Ligatur-Features legt* `clig` *den Kontext fest, in dem die Ligatur empfehlenswert ist. Diese Funktionalität ist für einige Schreibschriften und für Swash-Ligaturen wichtig.*
`cswh`	*contextual swash*	*Ersetzt in einem festgelegten Kontext die Standard-Glyphe durch die entsprechende Swash-Glyphe.*
`dlig`	*discretionary ligatures*	*Ersetzt mehrere Glyphen durch eine einzelne Glyphe.*
`dnom`	*denominators*	*Ersetzt Ziffern hinter einem Schrägstrich durch Nennerziffern.*
`frac`	*fractions*	*Ersetzt durch Bruchstrich getrennte Ziffern durch die gängigen (diagonalen) Brüche.*
`hist`	*historical forms*	*Ersetzt die aktuelle Standardform durch eine historische Alternative. Einige früher übliche Formen wirken heute anachronistisch.*
`hlig`	*historical ligatures*	*Ersetzt die (aktuellen) Standardligaturen durch historische Alternativen.*
`liga`	*standard ligatures*	*Ersetzt aus typografischen Gründen eine Reihe von Glyphen durch eine einzelne Glyphe.*
`lnum`	*lining figures*	*Wandelt Mediävalziffern in die standardmäßigen Versalziffern um.*
`locl`	*localized forms*	*Lokalisierte Glyphenformen können durch Standardformen ersetzt werden. Diese Funktion benötigt die Optionen* `script` *und* `language`.

1. See www.microsoft.com/typography/TrueTypeProperty21.mspx

Tabelle 6.1 Unterstützte OpenType-Funktionen für westliche Typografie (Funktionen für CJK-Text siehe Tabelle 6.6)

Schlüsselwort	Name	Beschreibung
mgrk	*mathematical Greek*	*Ersetzt typografische Standardformen für griechische Glyphen durch die entsprechenden in der Mathematik üblichen Formen.*
numr	*numerators*	*Ersetzt Ziffern vor einem Schrägstrich durch Zählerziffern und ersetzt den typografischen Schrägstrich durch den Bruchstrich.*
onum	*oldstyle figures*	*Wandelt die standardmäßigen Versalziffern in Mediävalziffern um.*
ordn	*ordinals*	*Ersetzt die standardmäßigen alphabetischen Glyphen durch die entsprechenden Ordinalformen, die hinter Ziffern verwendet werden; erzeugt normalerweise auch das Zeichen Numero (U+2116).*
ornm	*ornaments*	*Ersetzt das Aufzählungszeichen (•) und ASCII-Zeichen durch Ornamente.*
pcap	*petite capitals*	*Wandelt Kleinbuchstaben in Kapitälchen (petite caps) um, die kleiner sind als normale Kapitälchen.*
pnum	*proportional figures*	*Ersetzt äquidistante (nichtproportionale) Ziffern durch Ziffern mit proportionalen Breiten.*
salt	*stylistic alternates*	*Ersetzt die Standardformen durch alternative Stilvarianten. Diese lassen sich nicht immer einer bestimmten Kategorie wie Swash-Zeichen oder historischen Formen zuordnen.*
sinf	*scientific inferiors*	*Ersetzt Versal- oder Mediävalziiffern durch tiefgestellte Ziffern (kleinere Glyphen), hauptsächlich für chemische und mathematische Formeln).*
smcp	*small capitals*	*Wandelt Kleinbuchstaben in Kapitälchen um.*
ss01 ... ss20	*stylistic set 1-20*	*Neben oder anstelle der stilistischen Alternativen einzelner Glyphen (siehe Funktion* salt*), können einige Fonts Sätze von Stilvarianten für Glyphen enthalten, die Teilen des Zeichensatzes entsprechen, z.B. mehrere Varianten für Kleinbuchstaben in einem lateinischen Font.*
subs	*subscript*	*Ersetzt eine Standard-Glyphe durch eine tiefgestellte Glyphe.*
sups	*superscript*	*Ersetzt Versal- und Mediävalziffern durch hochgestellte Ziffern (meist für Fußnotenindikatoren) und ersetzt Kleinbuchstaben durch hochgestellte Buchstaben (meist für Abkürzungen französischer Titel).*
swsh	*swash*	*Ersetzt Standard-Glyphen durch die entsprechenden Swash-Zeichen.*
titl	*titling*	*Ersetzt Standard-Glyphen durch bestimmten Formen für Überschriften.*
tnum	*tabular figures*	*Ersetzt proportionale Ziffern durch äquidistante (nichtproportionale) Ziffern.*
unic	*unicase*	*Ordnet Groß- und Kleinbuchstaben einer Mischung aus Kleinbuchstaben und Kapitälchen zu und vereinheitlicht sie damit zu einem Alphabet aus gleich großen Buchstaben.*
zero	*slashed zero*	*Ersetzt die Glyphe für die Ziffer Null durch eine Null mit Schrägstrich.*

6.3.2 OpenType-Layoutfunktionen mit Textlines und Textflows

PDFlib unterstützt OpenType-Layoutfunktionen in den Funktionen Textline und Textflow, jedoch nicht in einfachen Textausgabefunktionen wie *PDF_show( )*.

Voraussetzungen für OpenType-Layoutfunktionen. Um OpenType-Features mit einem Font zu verwenden, muss er folgende Voraussetzungen erfüllen:

- Der Font muss vom Typ TrueType *(*.ttf)*, OpenType *(*.otf)* oder aus einer TrueType-Collection *(*.ttc)* sein.
- Die Fontdatei muss eine GSUB-Tabelle mit den verwendeten OpenType-Funktionen enthalten.
- Der Font muss mit *encoding=unicode* oder *glyphid* oder einer Unicode-CMap geladen werden.
- Die Option *readfeatures* von *PDF_load_font( )* darf nicht auf *false* gesetzt sein.

PDFlib unterstützt OpenType-Funktionen mit Lookups in den GSUB-Tabellen. Außer beim Kerning unterstützt PDFlib keine auf der GPOS-Tabelle basierenden OpenType-Funktionen.

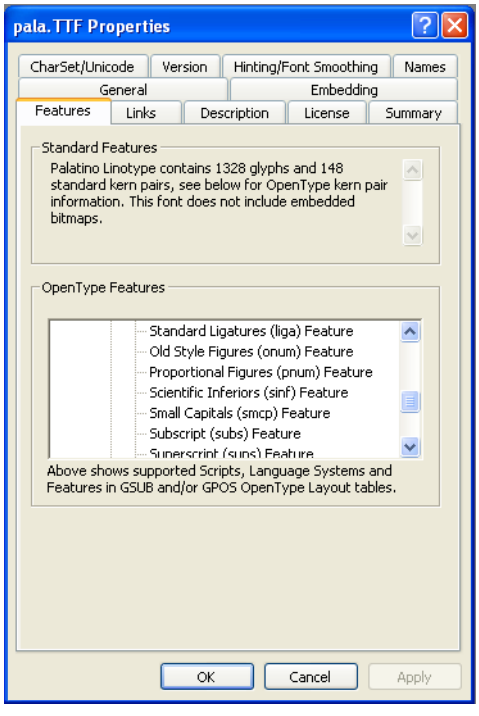


Abb. 6.3 Die Font Property Extension von Microsoft listet die OpenType-Features in einem Font auf

Einschränkungen. Beachten Sie beim Umgang mit OpenType-Funktionen Folgendes:

- OpenType-Funktionen (Optionen *features, script, language)* werden nur auf Glyphen innerhalb eines Fonts und nicht über Fontgrenzen hinweg angewendet, also nicht

auf Glyphen aus dem Basisfont und einem oder mehreren Fallback-Fonts, falls diese festgelegt wurden.

- Achten Sie darauf, Funktionen nach Bedarf zu aktiveren und zu deaktivieren. Versehentlich für den gesamten Text aktivierte OpenType-Funktionen können zu unerwarteten Ergebnissen führen.

Aktivieren und Deaktivieren von OpenType-Funktionen. Sie können OpenType-Funktionen nach ihren Wünschen für bestimmte Textstellen aktivieren oder deaktivieren. Mit der Textoption *features* können Sie Funktionen über die Angabe des Namens aktivieren und durch ein vorangestelltes *no* vor dem Funktionsnamen deaktivieren. Mit folgender Inline-Optionsliste für Textflow werden OpenType-Features wie folgt gesteuert:

```
<features={liga}>ffi<features={noliga}
```

Für Textlines können Sie OpenType-Funktionen folgendermaßen aktivieren:

```
p.fit_textline("ffi", x, y, "features={liga}");
```

OpenType-Features lassen sich auch als Blockeigenschaften für die Verwendung mit dem PDFlib Personalization Server (PPS) aktivieren.

Auf einen Text lässt sich mehr als eine Funktion anwenden, jedoch müssen die Funktionstabellen im Font entsprechend vorbereitet sein und die entsprechenden Feature-Lookups in der korrekten Reihenfolge enthalten sein. Betrachten wir zum Beispiel das Wort *office*, die Funktionen *ligature (liga)* und *small cap (smcp, Kapitäl-chen)*. Sind beide Funktionen aktiviert (unter der Annahme, dass der Font die entsprechenden Funktionseinträge enthält), würde man erwarten, dass die Funktion *small cap* (Kapitälchen) angewendet wird, nicht jedoch die Funktion *ligature*. Bei korrekter Implementierung in den Fonttabellen erzeugt PDFlib die erwartete Ausgabe, also Kapitälchen ohne Ligatur.

Deaktivieren von Ligaturen mithilfe von Steuerzeichen. Einige Sprachen verbieten den Gebrauch von Ligaturen in bestimmten Situationen. Typografische Regeln für Deutsch und andere Sprachen verbieten den Einsatz von Ligaturen bei Zusammensetzungen über die einzelnen Wortbestandteile hinweg. Zum Beispiel darf die Kombination *f+i* in dem Wort *Schilfinsel* nicht durch eine Ligatur ersetzt werden, da sie die Grenzen zwischen zwei kombinierten Wörtern überspannt.

Wie oben beschrieben, können Sie Ligaturen und andere OpenType-Features mit der Option *features* aktivieren und deaktivieren. In Ausnahmefällen wie dem obigen kann das Deaktivieren von Ligaturen über Optionen umständlich sein. Um Ligaturen leichter steuern zu können, können Sie sie über Steuerzeichen im Text deaktivieren und damit das Aktivieren/Deaktivieren von Funktionen mit mehreren Optionen vermeiden. Durch Einsetzen des Zeichens *Zero-width non-joiner* (U+200C, *‌* siehe auch Tabelle 6.4) zwischen den konstituierenden Zeichen wird das Ersetzen durch eine Ligatur verhindert, selbst wenn diese in der Option *features* aktiviert sind. Zum Beispiel erzeugt die folgende Sequenz keine *f+i*-Ligatur:

```
<features={liga charref=true}>Schilf&zwnj;insel
```

Schrift- und sprachspezifische OpenType-Layoutfunktionen. OpenType-Features kann man in allen Situationen oder für eine bestimmte Schrift anwenden oder sogar für eine bestimmte Kombination aus Schrift und Sprache umsetzen. Neben der Option *features*

kann man deshalb optional die Textoptionen *script* und *language* mit angeben. Sie haben nur einen spürbaren Effekt, wenn die Funktion in einer schrift- oder sprachspezifischen Weise im Font implementiert ist.

Beispielsweise ist bei der Auswahl von Türkisch als Sprache die Ligatur für die Glyphen *f* und *i* nicht in allen Fonts vorhanden, da die ligierte Form *i* mit dem im Türkischen üblichen punktlosen *i* verwechselt werden könnte. Ohne die Angabe einer Schrift/Sprache wird bei einem solchen Font durch folgende Textflow-Option dann eine Ligatur erzeugt:

```
<features={liga}>fi
```

Mit Angabe der Option für Türkisch wird durch folgende Textflow-Option jedoch keine Ligatur erzeugt:

```
<script=latn language=TRK features={liga}>fi
```

Die Funktion *locl* selektiert explizit die sprachspezifischen Zeichenformen. Wie aus folgenden Beispielen ersichtlich, enthält die Funktion *liga* sprachspezifische Ligaturen:

```
Variantenzeichen für Serbisch:
<features={locl} script=cyrl language=SRB charref>&#x0431;

Variantenziffern für Urdu:
<features={locl} script=arab language=URD charref>&#x0662;&#x0663;&#x0664;&#x0665;
```

Für Informationen zu unterstützten Schlüsselwörtern für Schrift und Sprache siehe Abschnitt 6.4.2, »Schrift und Sprache«, Seite 169.

Kombination von OpenType-Funktionen und Shaping. Shaping für komplexe Schriftsysteme (siehe Abschnitt 6.4, »Ausgabe komplexer Schriftsysteme«, Seite 167) stützt sich stark auf OpenType-Features, die automatisch ausgewählt werden. Bei einigen Fonts ist es jedoch sinnvoll, automatisch ausgewählte OpenType-Funktionen für das Shaping mit solchen OpenType-Funktionen zu kombinieren, die von der Client-Anwendung ausgewählt wurden. PDFlib wendet vor den automatisch gewählten OpenType-Funktionen für Shaping (Optionen *shaping*, *script* und *language)* zuerst die benutzerdefinierten OpenType-Funktionen an (Option *features)* an.

Abfrage von OpenType-Funktionen per Programm. Sie können OpenType-Funktionen in einem Font mit *PDF_info_font( )* programmgesteuert abfragen. Die folgende Anweisung ruft eine durch Leerzeichen getrennte Liste mit allen OpenType-Funktionen ab, die im Font vorhanden sind und von PDFlib unterstützt werden:

```
result = (int) p.info_font(font, "featurelist", "");
if (result != -1)
{
        /* Abfrage des Strings, der die durch Leerzeichen
            getrennte Funktionsliste enthält*/
        featurelist = p.get_string(result, "");
}
else
{
        /* keine unterstützten Features gefunden */
}
```

Mit der folgenden Abfrage können Sie prüfen, ob PDFlib und der Testfont eine bestimmte Funktion, wie zum Beispiel Ligaturen *(liga)* unterstützen:

```
result = (int) p.info_font(font, "feature", "name=liga");
if (result == 1)
{
        /* Feature wird von Font und PDFlib unterstützt */
}
```

6.4 Ausgabe komplexer Schriftsysteme

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `complex_scripts/starter_shaping`.

6.4.1 Komplexe Schriftsysteme

Die lateinische Schrift setzt grundsätzlich von links nach rechts ein Zeichen neben das andere. Andere Schriftsysteme haben zusätzliche Anforderungen an die korrekte Textausgabe. Wir bezeichnen solche Schriftsysteme als komplexe Schriftsysteme. PDFlib kann Text in vielen komplexen Schriftsystemen verarbeiten, einschließlich der in Tabelle 6.2 aufgeführten.

Dieser Abschnitt behandelt ausführlich das Thema Shaping für komplexe Schriften. Einige Schriftsysteme (Scripts) erfordern zusätzliche Verarbeitung:

- Arabische und hebräische Schrift setzt Text von rechts nach links. Gemischter Text (z.B. Arabisch mit lateinischen Einschüben) besteht aus linksläufigen und rechtsläufigen Segmenten (bidirektional, Bidi). Diese Segmente müssen neu angeordnet werden, was als Bidi-Problem bezeichnet wird.
- Manche Schriften, besonders Arabisch, verwenden je nach Position des Zeichens (isoliert, am Anfang, in der Mitte oder am Ende eines Worts) verschiedene Zeichenformen.
- Zwingend erforderliche Ligaturen ersetzen Zeichensequenzen.
- Die horizontale und vertikale Position der Glyphen muss angepasst werden.
- Bei indischen Schriften müssen manche Zeichen neu angeordnet werden, können also ihre Position im Text verändern.
- Für einige Schriften gelten bestimmte Regeln beim Umbrechen von Wörtern und beim Blocksatz.

Shaping. Das Verfahren zur Vorbereitung des Eingabetextes für die korrekte Darstellung heißt Shaping (dieser Begriff umfasst auch die Neuanordnung und Bidi-Verarbeitung von Text). Der Benutzer liefert den Text immer ungeformt und in der logischen Reihenfolge, wobei PDFlib das notwendige Shaping vor Erzeugung der PDF-Ausgabe übernimmt.

Shaping für komplexe Schriftsysteme lässt sich mit der Textoption *shaping* aktivieren, die wiederum die Option *script* benötigt. Die Option *language* kann optional angegeben werden. Die folgende Optionsliste aktiviert Shaping (und Bidi-Verarbeitung) für Arabisch:

```
shaping script=arab
```

Tabelle 6.2 Komplexe Schriftsysteme und Schlüsselwörter für die Option `script`

Schriftsystem	***Schriftname***	***Sprache/Region (unvollständige Liste)***	***Schlüsselwort***
unbestimmt	–		`_none`
automatische Schrifterkennung	–	*Dieses Schlüsselwort wählt die Schrift aus, zu der die meisten Zeichen im Text gehören, wobei* `_latn` *und* `_none` *ignoriert werden.*	`_auto`
Europäisch	*Lateinisch*	*viele europäische und andere Sprachen*	`latn`
	Griechisch	*Griechisch*	`grek`
	Kyrillisch	*Russisch und viele andere slawische Sprachen*	`cyrl`
Nahöstlich	*Arabisch*	*Arabisch, Persisch (Farsi), Urdu, Paschto und andere*	`arab`
	Hebräisch	*Hebräisch, Jiddisch und andere*	`hebr`
	Syrisch	*Orthodoxes Syrisch, Maronitisch, Assyrisch*	`syrc`
	Thaana	*Dhivehi/Maledivisch*	`thaa`
Südasiatisch (Indien)	*Devanagari*	*Hindi und klassisches Sanskrit*	`deva`
	Bengalisch	*Bengalisch, Assamesisch*	`beng`
	Gurmukhi	*Punjabi*	`guru`
	Gujarati	*Gujarati*	`gujr`
	Oriya	*Oriya/Orissa*	`orya`
	Tamil	*Tamil/Tamil Nadu, Sri Lanka*	`taml`
	Telugu	*Telugu/Andrha Pradesh*	`telu`
	Kannada	*Kannada/Karnataka*	`knda`
	Malayalam	*Malayalam/Kerala*	`mlym`
Südostasiatisch	*Thai*	*Thailändisch*	`thai`
	Lao	*Laotisch*	`»lao «`[1]
	Khmer	*Khmer (Kambodschanisch)*	`khmr`
Ostasiatisch	*Han*	*Chinesisch, Japanisch, Koreanisch*	`hani`
	Hiragana	*Japanisch*	`hira`
	Katakana	*Japanisch*	`kana`
	Hangul	*Koreanisch*	`hang`
Andere	*Weitere vierbuchstabige Codes können entsprechend der OpenType-Spezifikation auch verwendet werden, werden aber nicht unterstützt. Für eine vollständige Liste siehe:* `www.microsoft.com/typography/developers/OpenType/scripttags.aspx`		

1. Beachten Sie die Leerzeichen am Ende.

Einschränkungen. Beachten Sie beim Umgang mit Shaping für komplexe Schriftsysteme Folgendes:

- Sie müssen die Optionen *shaping* und *script* explizit angegeben, da PDFlib sie nicht automatisch setzt.
- Schriftspezifisches Shaping (Optionen *shaping, script, language)* wird nur auf Glyphen innerhalb desselben Fonts angewendet, aber nicht auf Glyphen aus anderen Fonts und über Fontgrenzen hinweg. Bei der Verwendung von Fallback-Fonts wird Shaping nur innerhalb von Textfragmenten mit demselben (Haupt- oder Fallback-) Font angewendet.
- Da beim Shaping Zeichen im Text neu angeordnet werden können, muss besonders auf Attribut-Änderungen innerhalb eines Wortes geachtet werden. Wenn Sie zum Beispiel Inline-Optionen im Textflow verwenden, um das zweite Zeichen in einem Wort einzufärben – was soll passieren, wenn durch das Shaping das erste und zweite Zeichen vertauscht werden? Aus diesem Grund sollten alle Textauszeichnungen in umgeformtem Text nur an Wortgrenzen, aber nicht innerhalb eines Wortes geändert werden.

Voraussetzungen für das Shaping. Ein Font muss für die Verwendung mit Shaping die Glyphen für die Zielschrift enthalten und außerdem folgende Anforderungen erfüllen:

- Es muss sich um einen TrueType- oder OpenType-Font mit GDEF-, GSUB- und GPOS-Funktionstabellen handeln und er muss die für die Zielsprache passenden Unicode-Zuordnungen aufweisen. Als Alternative zu den OpenType-Tabellen für die arabischen und hebräischen Schriften kann der Font auch Glyphen für die Unicode-Präsentationsformen enthalten (arabische Fonts von Apple sind auf diese Art strukturiert). In diesem Fall werden für den Shaping-Prozess interne Tabellen genutzt. Für thailändischen Text muss der Font den Konventionen von Microsoft, Apple oder Monotype WorldType (wie zum Beispiel in einigen IBM-Produkten verwendet) entsprechende kontextuelle Formen für Thailändisch enthalten.
- Der Font muss mit *encoding=unicode* oder *glyphid* geladen werden.
- Die Option *vertical* von *PDF_load_font( )* darf nicht verwendet werden und die Option *readshaping* darf nicht auf *false* gesetzt sein.

6.4.2 Schrift und Sprache

Einstellungen für Schrift und Sprache sind ausschlaggebend bei den unten aufgeführten funktionalen Aspekten. Sie können über folgende Optionen gesteuert werden:

- Mit der Textoption *script* wird das Schriftsystem bestimmt. Die in Tabelle 6.2 aufgeführten vierbuchstabigen Schlüsselwörter werden unterstützt, zum Beispiel:

```
script=latn
script=cyrl
script=arab
script=hebr
script=deva
script={lao }
```

  Mit *script=_auto* weist PDFlib automatisch die Schrift mit den meisten Zeichen im Text zu. Da lateinischer Text kein Shaping benötigt, wird er bei der automatischen Schrifterkennung nicht berücksichtigt. Sie können die im Text verwendeten Schriften mit dem Schlüsselwort *scriptlist* von *PDF_info_textline( )* abfragen.

- Die Option *language* bestimmt die natürliche Sprache, in der der Text geschrieben wurde. Die in Tabelle 6.3 aufgeführten dreibuchstabigen Schlüsselwörter werden unterstützt, zum Beispiel:

  ```
  language=ARA
  language=URD
  language=ZHS
  language=HIN
  ```

Verarbeitung komplexer Schriften. Komplexe Schriftverarbeitung (Option *shaping)* benötigt die Option *script*. Zusätzlich kann die Option *language* angegeben werden. Sie steuert die sprachspezifischen Aspekte des Shapings, wie unterschiedliche Ziffern bei Arabisch und Urdu. Nur wenige Fonts verfügen jedoch über sprachspezifische Tabellen für Schrift-Shaping. Deshalb reicht in den meisten Fällen die Angabe der Option *script* aus und das Shaping kann durch Angabe der Option *language* nicht weiter verbessert werden.

OpenType-Layoutfunktionen. Fonts können OpenType-Layoutfunktionen auf eine sprachspezifische Weise implementieren (siehe Abschnitt »Schrift- und sprachspezifische OpenType-Layoutfunktionen«, Seite 164). Da das Verhalten einiger Funktionen sich durch Angabe der Optionen *script* und *language* ändert, die Funktionen aber auch ohne diese Optionen verwendet werden können (z.B. *liga)*, ist die Funktion *locl* nur zusammen mit der Optionen *script* und *language* sinnvoll.

Hinweis *Obwohl der fortgeschrittene Zeilenumbruch für Textflow (siehe Abschnitt 8.2.9, »Erweiterter Zeilenumbruch für spezielle Schriftsysteme«, Seite 240) auch sprachspezifische Schritte umfasst, wird er nicht durch die Option* `language` *gesteuert, sondern durch die Option* `locale`*, die nicht nur Sprachen, sondern auch Länder und Regionen bestimmt.*

Tabelle 6.3 Schlüsselwörter für die Option language

Schlüsselwort	Sprache	Schlüsselwort	Sprache	Schlüsselwort	Sprache
_none	*unbestimmt*	FIN	*Finnisch*	NEP	*Nepalesisch*
AFK	*Afrikaans*	FRA	*Französisch*	ORI	*Oriya*
SQI	*Albanisch*	GAE	*Gälisch*	PAS	*Paschto*
ARA	*Arabisch*	DEU	*Deutsch*	PLK	*Polnisch*
HYE	*Armenisch*	ELL	*Griechisch*	PTG	*Portugiesisch*
ASM	*Assamesisch*	GUJ	*Gujarati*	ROM	*Rumänisch*
EUQ	*Baskisch*	HAU	*Hausa*	RUS	*Russisch*
BEL	*Weißrussisch*	IWR	*Hebräisch*	SAN	*Sanskrit*
BEN	*Bengali*	HIN	*Hindi*	SRB	*Serbisch*
BGR	*Bulgarisch*	HUN	*Ungarisch*	SND	*Sindhi*
CAT	*Katalanisch*	IND	*Indonesisch*	SNH	*Singhalesisch*
CHE	*Tschetschenisch*	ITA	*Italienisch*	SKY	*Slowakisch*
ZHP	*phonetisches Chinesisch*	JAN	*Japanisch*	SLV	*Slowenisch*
ZHS	*vereinfachtes Chinesisch*	KAN	*Kannada*	ESP	*Spanisch*
ZHT	*traditionelles Chinesisch*	KSH	*Kashmiri*	SVE	*Schwedisch*
COP	*Koptisch*	KHM	*Khmer*	SYR	*Syrisch*
HRV	*Kroatisch*	KOK	*Konkani*	TAM	*Tamil*
CSY	*Tschechisch*	KOR	*Koreanisch*	TEL	*Telugu*
DAN	*Dänisch*	MLR	*reformiertes Malayalam*	THA	*Thailändisch*
NLD	*Niederländisch*	MAL	*traditionelles Malayalam*	TIB	*Tibetisch*
DZN	*Dzongkha*	MTS	*Maltesisch*	TRK	*Türkisch*[1]
ENG	*Englisch*	MNI	*Manipuri*	URD	*Urdu*
ETI	*Estnisch*	MAR	*Marathi*	WEL	*Walisisch*
FAR	*Farsi*	MNG	*Mongolisch*	JII	*Jiddisch*

1. Manche Fonts verwenden fälschlicherweise TUR für Türkisch; PDFlib behandelt TUR wie TRK.

6.4.3 Shaping komplexer Schriftsysteme

Beim Shaping werden geeignete Glyphformen ausgewählt, je nachdem, ob ein Zeichen am Anfang, in der Mitte oder am Ende eines Wortes oder isoliert steht. Shaping ist eine entscheidende Komponente bei der Textformatierung für Arabisch und Hindi. Beim Shaping kann auch eine Abfolge von zwei oder mehr Zeichen durch eine geeignete Ligatur ersetzt werden. Da der Shaping-Prozess die richtigen Zeichenformen automatisch bestimmt, dürfen explizite Ligaturen und Unicode-Darstellungsformen (z.B. arabische Präsentationsformen U+FB50) nicht als Eingabezeichen verwendet werden.

Da komplexe Schriftsysteme mehrere verschiedene Glyphformen pro Zeichen sowie zusätzliche Regeln für die Auswahl und Platzierung dieser Glyphen erfordern, kann man das Shaping für komplexe Schriften nicht mit allen Fontarten verwenden, sondern nur mit solchen Fonts, die die notwendigen Informationen enthalten. Shaping wird für TrueType- und OpenType-Fonts unterstützt, da sie die gewünschten Funktionstabellen enthalten (für weitere Informationen siehe Abschnitt »Voraussetzungen für das Shaping«, Seite 169).

Shaping kann nur für Zeichen im selben Font durchgeführt werden, weil die Shaping-Information fontspezifisch ist. Da es zum Beispiel nicht sinnvoll ist, Ligaturen über verschiedene Fonts hinweg zu bilden, kann Shaping bei einem Wort, das aus Zeichen verschiedener Fonts besteht, nicht angewendet werden.

Überschreiben des Shaping-Verhaltens. Für den Fall, dass Sie das Standardverhalten beim Shaping überschreiben möchten, unterstützt PDFlib verschiedene Unicode-Formatierungszeichen. Diese können Sie auch mit den jeweiligen Entity-Namen angeben (siehe Tabelle 6.4).

Tabelle 6.4 Unicode-Steuerzeichen für das Überschreiben des Standardverhaltens beim Shaping

Format-zeichen	Entity-Name	Unicode-Name	Funktion
U+200C	ZWNJ	ZERO WIDTH NON-JOINER	zwischen zwei benachbarten Zeichen wird keine kursive Verbindung gebildet
U+200D	ZWJ	ZERO WIDTH JOINER	zwischen zwei benachbarten Zeichen wird eine kursive Verbindung gebildet

6.4.4 Bidirektionale Formatierung

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `complex_scripts/bidi_formatting`.

Bei linksläufigem Text (besonders bei arabischen und hebräischen, aber auch einigen anderen Schriften) kommen eingeschobene Sequenzen von rechtsläufigem lateinischen Text, wie Adressen oder Zitaten in anderen Sprachen, häufig vor. Diese gemischten Textsequenzen benötigen bidirektionale Formatierung (Bidi). Da Zahlen immer von links nach rechts geschrieben werden, betrifft das Bidi-Problem selbst vollständig auf arabisch oder hebräisch geschriebenen Text. PDFlib implementiert bidirektionale Neuanordnung von Text gemäß dem im Unicode Standard Annex #9[1] spezifizierten Bidi-Algorithmus. Bidi-Verarbeitung muss nicht über eine Option aktiviert werden, sondern

1. Siehe www.unicode.org/unicode/reports/tr9/

wird automatisch als Teil des Shaping-Prozesses angewendet, sobald linksläufiger Text mit der entsprechenden Option *script* vorliegt.

Hinweis *Die Bidi-Verarbeitung wird derzeit nur für Textlines (also einzeilige Textausgabe) unterstützt, nicht jedoch für mehrzeilige Textflows.*

Überschreiben des Bidi-Algorithmus. Während die automatische Bidi-Verarbeitung im Allgemeinen zu guten Resultaten führt, ist in manchen Fällen eine explizite Benutzersteuerung erforderlich. PDFlib unterstützt für diesen Zweck verschiedene Formatierungscodes für Schreibrichtungen. Der Einfachheit halber können diese Formatierungszeichen auch mit dem jeweiligen Entity-Namen angegeben werden (siehe Tabelle 6.5). Mit den bidirektionalen Formatierungscodes können sie den Standard-Bidi-Algorithmus in den folgenden Situationen überschreiben:

- ein linksläufiger Absatz beginnt mit einem rechtsläufigen Zeichen;
- bei verschachtelten Segmenten mit gemischtem Text;
- bei schwachen Zeichen, z.B. Satzzeichen zwischen rechts- und linksläufigem Text;
- bei Bestellnummern und ähnlichen Sequenzen mit gemischtem Text

Tabelle 6.5 Formatierungscodes für die Schreibrichtung zum Überschreiben des bidirektionalen Algorithmus

Formatzeichen	Entity-Name	Unicode-Name	Funktion
U+202A	LRE	LEFT-TO-RIGHT EMBEDDING (LRE)	Anfang einer eingebetteten rechtsläufigen Sequenz
U+202B	RLE	RIGHT-TO-LEFT EMBEDDING (RLE)	Anfang einer eingebetteten linksläufigen Sequenz
U+200E	LRM	LEFT-TO-RIGHT MARK (LRM)	rechtsläufiges Nullbreitenzeichen
U+200F	RLM	RIGHT-TO-LEFT MARK (RLM)	linksläufiges Nullbreitenzeichen
U+202D	LRO	LEFT-TO-RIGHT OVERRIDE (LRO)	Zeichen werden wie Zeichen einer rechtsläufigen Schrift behandelt
U+202E	RLO	RIGHT-TO-LEFT OVERRIDE (RLO)	Zeichen werden wie Zeichen einer linksläufigen Schrift behandelt
U+202C	PDF	POP DIRECTIONAL FORMATTING (PDF)	stellt den bidirektionalen Zustand vor dem letzten LRE, RLE, RLO oder LRO wieder her

Optionen für verbesserte Verarbeitung linksläufiger Dokumente. Die Standardeinstellungen der verschiedenen Formatierungsoptionen und das Verhalten von Acrobat sind auf rechtsläufige Textausgabe ausgerichtet. Verwenden Sie für linksläufige Textformatierung und Dokumentanzeige die folgenden Optionen:

- Platzieren einer rechtsbündigen Textline:

```
position={right center}
```

- Erstellen eines Führungszeichens zwischen Text und linkem Rand:

```
leader={alignment=left text=.}
```

- Verbessern der linksläufigen Dokument- und Seitenanzeige in Acrobat mit der folgenden Option von *PDF_begin/end_document( )*:

```
viewerpreferences={direction=r2l}
```

Behandlung von Bidi-Text in Ihrem Code. Beachten Sie bitte beim Umgang mit bidirektionalem Text folgende Punkte:

- Mit den Schlüsselwörtern *startx/starty* und *endx/endy* von *PDF_info_textline( )* können Sie die jeweiligen Koordinaten der logischen Start- und Endzeichen ermitteln.
- Mit dem Schlüsselwort *writingdirx* von *PDF_info_textline( )* können Sie die Haupt-Schreibrichtung des Textes ermitteln. Diese Richtung wird von den ersten Zeichen des Textes oder von direktionalen Formatierungscodes entsprechend Tabelle 6.5 abgeleitet (falls im Text vorhanden).
- Mit dem Schlüsselwort *auto* für die Option *position* von *PDF_info_textline( )* können Sie arabischen oder hebräischen Text automatisch am rechten Rand und lateinischen Text automatisch am linken Rand ausrichten lassen. Zum Beispiel richtet die folgende Optionsliste für Textline den Text rechts oder links an der Grundlinie aus:

```
boxsize={width 0} position={auto bottom}
```

6.4.5 Arabische Textformatierung

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `complex_scripts/arabic_formatting`.

Neben der oben besprochenen bidirektionalen Formatierung und dem Shaping sind bei der Erzeugung arabischer Textausgabe noch weitere Formataspekte zu beachten.

Arabische Ligaturen. Die arabische Schrift macht umfangreichen Gebrauch von Ligaturen. Viele arabische Fonts enthalten zwei Arten von Ligaturen, die in PDFlib unterschiedlich behandelt werden:

- Zwingend erforderliche Ligaturen (Funktion *rlig*) müssen immer angewendet werden, zum Beispiel die Ligatur Lam-Alef und ihre Varianten. Vorgeschriebene Ligaturen werden verwendet, wenn die Option *shaping* über *script=arab* aktiviert ist.
- Optionale arabischen Ligaturen (Funktionen *liga* und *dlig*) werden nicht automatisch verwendet, können aber wie andere benutzergesteuerte OpenType-Funktionen aktiviert werden, das heißt über *features={liga}*. Optionale arabische Ligaturen werden nach komplexer Schriftverarbeitung und Shaping angewendet.

Lateinische Ligaturen in arabischem Text. In Textlines kann die schriftspezifische Verarbeitung von OpenType-Funktionen zu unerwarteten Ergebnissen führen. Beispielsweise funktionieren lateinische Ligaturen in Kombination mit arabischem Text innerhalb derselben Textline nicht, da die Option *script* nur einmal für den Inhalt einer Textline übergeben werden kann und sich sowohl auf die Optionen *shaping* als auch auf *feature* auswirkt:

```
shaping script=arab features={liga}     FALSCH, funktioniert nicht bei gängigen Fonts!
```

Arabische Fonts enthalten jedoch normalerweise keine lateinischen Ligaturen für das arabische Schriftsystem, sondern nur für die Standardschrift oder lateinische Schrift – aber die Option *script* kann innerhalb einer einzelnen Textline nicht geändert werden. Deshalb findet PDFlib hier keine lateinischen Ligaturen und gibt stattdessen einfache Zeichen aus.

Vermeidung von Ligaturen. In einigen Fällen, zum Beispiel bei bestimmten Abkürzungen, ist die Verbindung benachbarter Zeichen nicht erwünscht. In solchen Fällen kön-

nen Sie die in Tabelle 6.4 aufgeführten Formatierungszeichen verwenden, um die Verbindung von Zeichen zu erzwingen oder zu verhindern. Das Nullbreitenzeichen im folgenden Beispiel verhindert zur korrekten Bildung einer Abkürzung die Verbindung der einzelnen Zeichen:

```
&#x0623;&#x064A;&ZWNJ;&#x0628;&#x064A;&ZWNJ;&#x0625;&#x0645;
```

Tatweel-Formatierung für arabische Texte. Sie können arabische Wörter strecken, indem Sie eine oder mehrere Instanzen des Tatweel-Zeichens U+0640 (auch Kashida genannt) einfügen. Obwohl PDFlib dadurch Text nicht automatisch im Blocksatz formatiert, können Sie dieses Zeichen in den Text einfügen, um Wörter zu strecken.

Hinzufügen lateinischer Zeichen zu einem arabischen Font. Manche arabischen Fonts enthalten keine Glyphen für lateinische Zeichen, zum Beispiel die arabischen Fonts von OS X. In diesem Fall können Sie mit der Option *fallbackfonts* lateinische Zeichen in einen arabischen Font einfügen. PDFlib schaltet je nach arabischer oder lateinischer Texteingabe automatisch zwischen beiden Fonts um. Sie müssen also die Fonts in ihrer Anwendung nicht austauschen, sondern können den gemischten lateinisch-arabischen Text mit einer einzigen Fontspezifikation übergeben.

Mit der folgenden Optionsliste für die Option *fallbackfonts* zum Laden von Fonts können Sie lateinische Zeichen des Fonts Helvetica zum geladenen arabischen Font hinzufügen:

```
fallbackfonts={
 {fontname=Helvetica encoding=unicode forcechars={U+0021-U+00FF}}
}
```

6.5 Chinesische, japanische und koreanische Textausgabe

6.5.1 Verwendung von CJK-Fonts vom Typ TrueType und OpenType

Hinweis *PDFlib GmbH bietet die Fonts MS Gothic und MS Mincho zum kostenlosen Download auf www.pdflib.com an. PDFlib-Lizenznehmer sind berechtigt, diese Fonts ohne den Erwerb einer separaten Fontlizenz zu verwenden.*

PDFlib unterstützt CJK-Fonts in den Formaten TrueType, TrueType-Collections (TTC) und OpenType. CJK-Fonts werden folgendermaßen verarbeitet:

- Wenn die Option *embedding* auf *true* gesetzt ist, wird der Font in einen CID-Font umgewandelt und in die PDF-Ausgabe eingebettet.
- Unter Windows können Namen von CJK-Host-Fonts an *PDF_load_font()* als UTF-8 mit BOM oder UTF-16 übergeben werden. Nicht lateinische Namen von Host-Fonts werden unter OS X nicht unterstützt.

Im folgenden Beispiel wird der Font *ArialUnicodeMS* verwendet, um chinesischen Text anzuzeigen. Der Font muss entweder auf dem System installiert sein oder so konfiguriert sein, wie in Abschnitt 5.4.4, »Suche nach Fonts«, Seite 136 beschrieben:

```
font = p.load_font("Arial Unicode MS", "unicode", "");
if (font == -1) { ... }
p.setfont(font, 24);
p.fit_textline("\u4e00\u500b\u4eba", x, y, "");
```

Zugriff auf einzelne Fonts einer TrueType Collection (TTC). TTC-Dateien enthalten mehrere einzelne Fonts. Auf jeden Font können Sie durch Angabe des korrekten Namens zugreifen. Wenn Sie jedoch nicht wissen, welche Fonts eine TTC-Datei enthält, können Sie die Fonts numerisch adressieren, indem Sie einen Doppelpunkt sowie den Index des Fonts innerhalb der TTC-Datei anhängen (beginnend bei 0). Die TTC-Datei *msgothic.ttc* enthält beispielsweise mehrere Fonts, die sich in *PDF_load_font()* wie folgt adressieren lassen (alle Einträge in einer Zeile können äquivalent sind):

```
msgothic:0          MS Gothic           msgothic:
msgothic:1          MS PGothic
msgothic:2          MS UI Gothic
```

Beachten Sie, dass *msgothic* (ohne Suffix) keinen Fontnamen darstellt, da es den Font nicht eindeutig bezeichnet. In Kombination mit TTC-Indizierung können Alias-Namen für Fonts verwendet werden (siehe Abschnitt »Quellen für Fontdaten«, Seite 136). Ist ein Font mit dem angegebenen Index nicht auffindbar, so scheitert der Funktionsaufruf.

Die TTC-Fontdatei muss nur einmal konfiguriert werden; alle in der TTC-Datei indizierten Fonts werden automatisch gefunden. Mit folgendem Code lassen sich alle in *msgothic.ttc* indizierten Fonts konfigurieren (siehe Abschnitt 5.4.4, »Suche nach Fonts«, Seite 136):

```
p.set_option("FontOutline={msgothic=msgothic.ttc}");
```

Horizontale und vertikale Schreibrichtung. PDFlib unterstützt sowohl horizontale als auch vertikale Schreibrichtung. Vertikale Schreibrichtung kann auf verschiedene Arten

aktiviert werden (beachten Sie, dass vertikale Schreibrichtung nicht für Type-1-Fonts unterstützt wird):

- TrueType- und OpenType-Fonts mit Encodings, die keine CMaps sind, können mit der Fontoption *vertical* für vertikale Schreibrichtung verwendet werden.
- Fonts, deren Name mit dem Zeichen @ beginnt, werden immer im vertikalen Modus verarbeitet.
- Für Standard-CJK-Fonts und CMaps wird der entsprechende Modus gemeinsam mit dem Encoding über einen geeigneten CMap-Namen ausgewählt. CMaps mit einem auf *-H* endenden Namen bewirken horizontale Schreibrichtung, während die Endung *-V* vertikale Schreibrichtung bewirkt.

Hinweis *Der Zeichenabstand muss negativ sein, um die Zeichen bei vertikaler Schreibrichtung weiter voneinander zu trennen.*

6.5.2 EUDC- und SING-Fonts für Gaiji-Zeichen

PDFlib unterstützt EUDC-Fonts von Windows (*end-user defined characters, *.tte)* und SING-Fonts *(*.gai),* die für den Zugriff auf benutzerdefinierte Gaiji-Zeichen für CJK-Text verwendet werden können. Fonts mit benutzerdefinierten Zeichen lassen sich mit dem Mechanismus der Fallback-Fonts leicht in andere Fonts integrieren. Gaiji-Zeichen werden häufig in EUDC- oder SING-Fonts zur Verfügung gestellt.

Verwendung von Fallback-Fonts für Gaiji-Zeichen. Typischerweise werden Gaiji- Zeichen aus Windows EUDC-Fonts oder SING-Glyphlets entnommen, jedoch akzeptiert die Option *fallbackfonts* jegliche Art von Font. Deshalb ist dieser Ansatz nicht auf Gaiji-Zeichen beschränkt, sondern kann für jede Art von Symbol (z.B. ein Firmenlogo in einem separaten Font) verwendet werden. Sie können die folgende Optionsliste zum Laden von Fonts für die Option *fallbackfonts* verwenden, um eine benutzerdefiniertes (Gaiji-) Zeichen aus einem EUDC-Font zu dem geladenen Font hinzuzufügen:

```
fallbackfonts={
 {fontname=EUDC encoding=unicode forcechars=U+E000 fontsize=140% textrise=-20%}
}
```

Sobald ein Basisfont mit dieser Konfiguration des Fallback-Fonts geladen wurde, kann das EUDC-Zeichen im Text ohne jegliche Fontänderung verwendet werden.

Bei SING-Fonts muss der Unicode-Wert nicht übergeben werden, da er automatisch von PDFlib ermittelt wird:

```
fallbackfonts={
 {fontname=PDFlibWing encoding=unicode forcechars=gaiji}
}
```

Vorbereiten von EUDC-Fonts. Sie können den in Windows verfügbaren EUDC-Editor *(End-user defined characters)* nutzen, um eigene Zeichen zu erstellen, die in PDFlib einsetzbar sind. Dazu gehen Sie wie folgt vor:

- Erstellen Sie mit *eudcedit.exe* ein oder mehrere eigene Zeichen an den gewünschten Unicode-Positionen.
- Finden Sie die Datei *EUDC.TTE* im Verzeichnis *\Windows\fonts* und kopieren Sie sie in ein anderes Verzeichnis. Da diese Datei im Windows Explorer nicht sichtbar ist, müssen Sie die Befehle *dir* und *copy* in der Windows-Eingabeaufforderung (MS-DOS-Fenster) verwenden. Dann konfigurieren Sie den Font zum Einsatz mit PDFlib, wobei Sie

eines der Verfahren in Abschnitt 5.4.4, »Suche nach Fonts«, Seite 136, verwenden:

```
p.set_option("FontOutline={EUDC=EUDC.TTE}");
p.set_option("SearchPath={{...directory name...}}");
```

Sie können *EUDC.TTE* auch im aktuellen Verzeichnis ablegen.

- Alternativ zum vorherigen Schritt können Sie folgende Funktion aufrufen, um die Fontdatei direkt aus dem Windows-Verzeichnis heraus zu konfigurieren. Auf diese Weise greifen Sie immer auf den in Windows aktuell verwendeten EUDC-Font zu:

  ```
  p.set_option("FontOutline={EUDC=C:\Windows\fonts\EUDC.TTE}");
  ```

- Integrieren Sie wie oben beschrieben den EUDC-Font mit der Option *fallbackfonts* in einen Basisfont. Wenn Sie direkt auf den Font zugreifen möchten, laden Sie ihn wie üblich mit dem folgenden Aufruf in PDFlib:

  ```
  font = p.load_font("EUDC", "unicode", "");
  ```

  Verwenden Sie die im ersten Schritt gewählten Unicode-Werte zur Zeichenausgabe.

6.5.3 OpenType-Layoutfunktionen für erweiterte CJK-Textausgabe

Wie in Abschnitt 6.3, »OpenType-Layoutfunktionen«, Seite 160 beschrieben, unterstützt PDFlib erweiterte typografische Layout-Tabellen in OpenType- und TrueType-Fonts. Mit OpenType-Funktionen lassen sich zum Beispiel alternative Formen lateinischer Glyphen halber oder proportionaler Breiten oder alternative Zeichenformen auswählen. Tabelle 6.6 gibt eine Übersicht über OpenType-Funktionen für CJK-Textausgabe.

Die Funktion *vert* (vertikale Schreibrichtung) wird bei Fonts mit vertikaler Schreibrichtung automatisch aktiviert (durch Übergabe der Option *vertical* an *PDF_load_font( )*) und wird bei Fonts mit horizontaler Schreibrichtung automatisch deaktiviert.

Tabelle 6.6 Unterstützte OpenType-Layoutfunktionen für Chinesisch, Japanisch und Koreanisch (Tabelle 6.1 listet zusätzlich unterstützte OpenType-Layoutfunktionen für den allgemeinen Gebrauch auf)

Schlüsselwort	Name	Beschreibung
expt	*expert forms*	*Wie bei den JIS78-Formen ersetzt diese Funktion japanische Standardformen durch die entsprechenden benutzerdefinierten Formen.*
fwid	*full widths*	*Ersetzt Glyphen anderer Breiten durch Glyphen voller Breite (normalerweise em). Dies umfasst auch lateinische Zeichen und verschiedenste Symbole.*
hkna	*horizontal Kana alternates*	*Ersetzt Standard-Kana durch Formen, die speziell nur auf horizontale Schreibrichtung ausgerichtet sind.*
hngl	*Hangul*	*Ersetzt koreanische Hanja-Zeichen (in chinesischem Stil) durch die entsprechenden Hangul-Zeichen (in Silbenschrift).*
hojo	*Hojo Kanji forms (JIS X 0212-1990)*	*Zugriff auf die Glyphen JIS X 0212-1990 (auch »Hojo Kanji« genannt), wenn die Form JIS X 0213:2004 vordefiniert ist.*
hwid	*half widths*	*Ersetzt Glyphen proportionaler oder fester Breiten, außer einem halben em, durch Glyphen mit Breiten von einem halben em (en).*
ital	*italics*	*Ersetzt gerade Glyphen durch die entsprechenden kursiven Glyphen.*
jp04	*JIS2004 forms*	*(Untermenge der Funktion* nlck*) Zugriff auf die Glyphen gemäß JIS X 0213:2004.*
jp78	*JIS78 forms*	*Ersetzt japanische Standard-Glyphen (JIS90) durch die entsprechenden Formen aus JIS C 6226-1978 (JIS78).*

Tabelle 6.6 Unterstützte OpenType-Layoutfunktionen für Chinesisch, Japanisch und Koreanisch (Tabelle 6.1 listet zusätzlich unterstützte OpenType-Layoutfunktionen für den allgemeinen Gebrauch auf)

Schlüsselwort	Name	Beschreibung
jp83	*JIS83 forms*	*Ersetzt japanische Standard-Glyphen (JIS90) durch die entsprechenden Formen aus JIS X 0208-1983 (JIS83).*
jp90	*JIS90 forms*	*Ersetzt japanische Standard-Glyphen aus JIS78 oder JIS83 durch die entsprechenden Formen aus JIS X 0208-1990 (JIS90).*
locl	*localized forms*	*Lokalisierte Glyphenformen können durch Standardformen ersetzt werden. Diese Funktion benötigt die Optionen* script *und* language.
nalt	*alternate annotation forms*	*Ersetzt Standard-Glyphen durch verschiedene Notationsformen (z.B. Glyphen in offenen oder geschlossenen Kreisen, Quadraten, Klammern, Rauten oder abgerundeten Rechtecken).*
nlck	*NLC Kanji forms*	*Zugriff auf neue Glyphformen, die im Jahr 2000 vom National Language Council (NLC) in Japan für eine Reihe von JIS-Zeichen definiert wurden.*
pkna	*proportional Kana*	*Ersetzt Glyphen vom Typ Kana und Kana-bezogene Glyphen mit gleichmäßigen Breiten (halbe oder volle Breite) durch proportionale Glyphen.*
pwid	*proportional widths*	*Ersetzt Glyphen mit gleichmäßigen Breiten (normalerweise Breite eines ganzen oder halben em) durch Glyphen mit proportionalem Abstand.*
qwid	*quarter widths*	*Ersetzt Glyphen anderer Breiten durch Glyphen mit Breiten von einem Viertel em (ein halbes en).*
ruby	*Ruby notation forms*	*Ersetzt Standard-Glyphen vom Typ Kana durch kleinere Glyphen, die für (normalerweise hochgestelltes) Ruby bestimmt sind.*
smpl	*simplified forms*	*Ersetzt traditionelle chinesische oder japanische Formen durch die entsprechenden vereinfachten Formen.*
tnam	*traditional name forms*	*Ersetzt vereinfachte japanische Kanji-Formen durch die entsprechenden traditionellen Formen. Dies ist gleichbedeutend mit der Funktion* trad*, jedoch auf die für Personennamen geeigneten traditionellen Formen beschränkt.*
trad	*traditional forms*	*Ersetzt vereinfachte chinesische Hanzi-Formen oder japanische Kanji-Formen durch die entsprechenden traditionellen Formen.*
twid	*third widths*	*Ersetzt Glyphen anderer Breiten durch Glyphen mit Breiten von einem Drittel em.*
vert	*vertical writing*	*Ersetzt Standardformen durch Varianten, die auf vertikale Schreibrichtung ausgerichtet sind.*
vkna	*vertical Kana alternates*	*Ersetzt Standard-Kana durch Formen, die speziell nur auf vertikale Schreibrichtung ausgerichtet sind.*
vrt2	*vertical alternates and rotation*	*(Überschreibt die Funktion* vert*, eine Untermenge der Funktion* vrt2*) Ersetzt Glyphen mit fester Breite (halber, Drittel- oder Viertelbreite) oder proportionaler Breite (zumeist Latin oder Katakana) durch Formen, die für vertikale Schreibrichtung geeignet sind (d.h., um 90° im Uhrzeigersinn gedreht).*

6.5.4 Variantenselektoren und Variantensequenzen von Unicode

Unicode-Zeichen können durch eine große Vielzahl von Glyphen dargestellt werden. Im Allgemeinen werden solche visuellen Unterschiede durch die Verwendung geeigneter Fonts realisiert (z.B. regulär im Vergleich zu kursiv). In einigen Situationen ist die Wahl der Glyphe semantisch relevant und muss deutlich gemacht werden, auch bei Klartext ohne fontbezogene Formatierungsinformationen. Für diesen Zweck bietet Unicode den Mechanismus der Variantenselektoren.

Variantensequenzen. Eine Variantensequenz besteht aus einem Basis-Unicode-Zeichen gefolgt von einem Variantenselektor. Die Sequenz wird als *Variante* des Standardzeichens bezeichnet. Der Unicode-Standard kennt zwei Arten solcher Sequenzen:

- Standardisierte Variantensequenzen sind in der Datei *StandardizedVariants.txt*[1] in der Unicode-Zeichendatenbank definiert. Sie verwenden einen von 16 Variantenselektoren im Bereich *U+FE00 - U+FE0F*. Standardisierte Variantensequenzen werden für die Auswahl von alternativen mathematischen Glyphen, Emoji-Varianten und mongolischem Text verwendet.
- Ideografische Variantensequenzen (IVS) werden vom Registrierungsprozess gemäß dem Unicode Technical Standard #37, »Unicode Ideographic Variation Database« definiert und in der *Ideographic Variation Database*[2] aufgeführt. Ein IVS besteht aus einem vereinheitlichten ideografischen Zeichen als Basiszeichen und einem von 240 Variantenselektoren im Bereich *U+E0100 - U+E01EF*. IVS werden vor allem zur Auswahl geeigneter Glyphen für Orts- und Personennamen verwendet.

Wenn der Variantenselektor für das Basiszeichen nicht eingesetzt werden kann, weil der Font zum Beispiel nicht die erforderliche Glyphe enthält, wird er ignoriert.

Erzeugung von Glyphvarianten mit PDFlib. Geeignete Zeichen für Variantensequenzen von Unicode (UVS) müssen vom Font bereitgestellt werden. Derzeit ist OpenType das einzige Fontformat, dass UVS speichern kann (über eine CMap-Tabelle mit Format 14). PDFlib verarbeitet die UVS-Tabelle in einem OpenType-Font, wenn dies nicht über die Fontoption *readselectors* deaktiviert wurde. Da ein Font nur für Content-Strings, nicht aber für Hypertext- und Name-Strings verfügbar ist, werden Variantenselektoren für diese Stringtypen immer ignoriert.

Wenn Sie wissen, dass ein Font die erforderlichen Glyphen enthält, müssen Sie für den Einsatz von Variantensequenzen die Sequenz nur an die Textausgabe-Funktionen von PDFlib übergeben. Mit den folgenden Codefragmenten lässt sich die Standardglyphe eines Basiszeichens von Unicode sowie eine durch einen Selektor gewählte Variante ausdrucken:

```
p.fit_textline("&#x2268; &#x2268;&#xFE00;", 50, 700,
        "fontname={Cambria Math} encoding=unicode fontsize=24 charref=true");

p.fit_textline("&#x3402;&#xE0100; &#x3402;&#xE0101;", 50, 650,
        "fontname={KozMinPr6N-Regular} encoding=unicode fontsize=24 charref=true");
```

Die jeweilige Ausgabe sehen Sie in Abbildung 6.4; beachten Sie den Unterschied innerhalb der Glyphenpaare.

1. Siehe www.unicode.org/Public/UNIDATA/StandardizedVariants.html
2. Siehe www.unicode.org/ivd

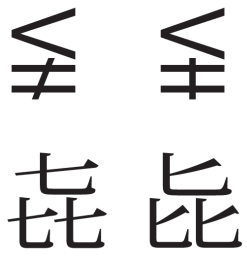


Abb. 6.4 Vordefinierte Glyphen und Glyphvarianten

Hinweis Variantensequenzen werden nicht in der Unteroption `forcechars` *der Option* `fallbackfonts` *unterstützt.*

Abfrage von Variantenselektoren in einem Font. Mit *PDF_info_font()* können Sie prüfen, ob ein Font überhaupt Variantenselektoren enthält. Mit dem Schlüsselwort *selector* und der Option *index* lässt sich eine Liste aller im Font verfügbaren Variantenselektoren abfragen:

```
for (i = 0; i < 256; i++)
{
        selectors[i] = (int) p.info_font(font, "selector", "index=" + i);
        if (selectors[i] == -1)
        {
            selectorcount = i;
            break;
        }
}
```

Mit dem folgenden Codefragment lässt sich prüfen, ob der Font eine Glyphvariante für eine bestimmte Sequenz enthält:

```
if (p.info_font(font, "unicode", "unicode=" + uv + "selector=" + s) == -1)
{
        /* keine Glyphvariante für diese Sequenz im Font verfügbar */
}
```

Mit dieser Abfrage soll nur überprüft werden, ob eine Variante zur Verfügung steht. Der daraus resultierende Unicode-Wert (falls eine Variante verfügbar ist) ist wahrscheinlich wenig nützlich, weil PDFlib den Varianten PUA-Unicode-Werte zuweist.

6.5.5 Standard-CJK-Fonts

Hinweis Das Konzept der Standard-CJK-Fonts ist veraltet; benutzen Sie stattdessen extern konfigurierte Fontdateien mit oder ohne Einbettung.

Acrobat unterstützt einen Satz von Standardfonts für CJK-Text. Diese Fonts werden mit der Acrobat-Installation ausgeliefert und müssen somit nicht in der PDF-Datei eingebettet sein. Die Standard-CJK-Fonts unterstützen sowohl horizontale als auch vertikale Schreibrichtung. Weitere Information zu den Namen dieser Fonts und anwendbaren CMaps finden Sie in Tabelle 6.7. Weitere Informationen zu CJK-CMaps finden Sie in Abschnitt 4.5, »Encodings für Chinesisch, Japanisch und Koreanisch«, Seite 110.

Tabelle 6.7 Standardfonts und CMaps (Encodings) von Acrobat für Japanisch, Chinesisch und Koreanisch

Sprache	Fontname	Unterstützte CMaps (Encodings)
Vereinfachtes Chinesisch	AdobeSongStd-Light[2]	GB-EUC-H, GB-EUC-V, GBpc-EUC-H, GBpc-EUC-V, GBK-EUC-H, GBK-EUC-V, GBKp-EUC-H, GBKp-EUC-V, GBK2K-H, GBK2K-V, UniGB-UCS2-H, UniGB-UCS2-V, UniGB-UTF16-H[1], UniGB-UTF16-V[1]
Traditionelles Chinesisch	AdobeMingStd-Light[2]	B5pc-H, B5pc-V, HKscs-B5-H, HKscs-B5-V, ETen-B5-H, ETen-B5-V, ETenms-B5-H, ETenms-B5-V, CNS-EUC-H, CNS-EUC-V, UniCNS-UCS2-H, UniCNS-UCS2-V, UniCNS-UTF16-H[1], UniCNS-UTF16-V[1]
Japanisch	KozMinPro-Regular-Acro[3] KozGoPro-Medium[2] KozMinProVI-Regular[2]	83pv-RKSJ-H, 90ms-RKSJ-H, 90ms-RKSJ-V, 90msp-RKSJ-H, 90msp-RKSJ-V, 90pv-RKSJ-H, Add-RKSJ-H, Add-RKSJ-V, EUC-H, EUC-V, Ext-RKSJ-H, Ext-RKSJ-V, H, V, UniJIS-UCS2-H, UniJIS-UCS2-V, UniJIS-UCS2-HW-H[3], UniJIS-UCS2-HW-V[3], UniJIS-UTF16-H[1], UniJIS-UTF16-V[1]
Koreanisch	AdobeMyungjoStd-Medium[2]	KSC-EUC-H, KSC-EUC-V, KSCms-UHC-H, KSCms-UHC-V, KSCms-UHC-HW-H, KSCms-UHC-HW-V, KSCpc-EUC-H, UniKS-UCS2-H, UniKS-UCS2-V, UniKS-UTF16-H[1], UniKS-UTF16-V[1]

1. Nur bei der Generierung von PDF ab Version 1.5 verfügbar
2. Nur bei der Generierung von PDF ab Version 1.6 verfügbar
3. Die HW-CMaps dürfen nicht mit den Fonts KozMinPro-Regular-Acro und KozGoPro-Medium-Acro verwendet werden, da diese nur proportionale ASCII-Zeichen, aber keine Zeichen halber Breite enthalten.

Beibehalten älterer nativer CJK-Codes. Bei *keepnative=true* werden ältere native Zeichencodes (z.B. Shift-JIS) entsprechend der selektierten CMap in die PDF-Ausgabe geschrieben; ansonsten wird der Text in Unicode konvertiert. Vorteilhaft bei *keepnative=true* ist, dass solche Fonts für Formularfelder ohne Einbettung verwendet werden können (siehe die Beschreibung der Fontoption von *keepnative* in der *PDFlib-Referenz*). Bei *keepnative=false* werden in CID-Werte konvertiert, die in die PDF-Ausgabe geschrieben werden. Dadurch lassen sich OpenType-Funktionen und der Textflow-Formatierer verwenden.

Um kleinere Probleme in Acrobat zu vermeiden, empfehlen wir, *keepnative=false* zu setzen, wenn keine Fonteinbettung gewünscht ist und CJK-Ladesequenzen auf *embedding=true* zu setzen, wenn *keepnative=true* gewünscht ist.

7 Import von Rasterbildern, SVG-Grafik und PDF-Seiten

7.1 Rasterbilder

7.1.1 Einbetten von Rasterbildern

Die Einbettung von Rasterbildern mit PDFlib ist einfach zu bewerkstelligen. Die Bilddatei muss zunächst mit einer PDFlib-Funktion geöffnet werden, die die Bildparameter analysiert und die Pixeldaten in die PDF-Ausgabe kopiert. Die Funktion *PDF_load_image( )* gibt ein Handle zurück, das als Bilddeskriptor dient. Es kann zusammen mit Positionierungs- und Skalierungsparametern im Aufruf von *PDF_fit_image( )* verwendet werden:

```
image = p.load_image("auto", "image.jpg", "");
if (image == -1)
        throw new Exception("Error: " + p.get_errmsg());

p.fit_image(image, 0.0, 0.0, "");
p.close_image(image);
```

PDF_fit_image( ) erhält als letztes Argument eine Optionsliste mit verschiedenen Optionen zur Positionierung, Skalierung und Rotation des Bildes. Weitere Informationen hierzu finden Sie in Abschnitt 7.4, »Platzieren von Bildern, Grafiken und importierten PDF-Seiten«, Seite 207.

Cookbook *Codebeispiele zur Bildbearbeitung finden Sie in der Kategorie* `images` *des PDFlib Cookbook.*

Wiederverwendung von Bilddaten. PDFlib unterstützt ein wichtiges PDF-Verfahren zur Reduzierung der Dateigröße bei mehrfach vorkommenden Rasterbildern. Betrachten wir zum Beispiel ein Layout, bei dem sich ein bestimmtes Logo oder ein Hintergrund auf mehreren Seiten befindet. In solchen Fällen ist es möglich, die Bilddaten in der PDF-Datei nur einmal zu speichern und auf jeder Seite nur noch eine Referenz darauf anzulegen. Dazu öffnen Sie die Bilddatei und rufen einfach *PDF_fit_image( )* auf jeder Seite auf, auf der Sie das Logo oder den Hintergrund platzieren möchten. Sie können das geöffnete Bild auf mehreren Seiten platzieren und dabei unterschiedliche Skalierungsfaktoren verwenden (solange es nicht wieder geschlossen wird). Abhängig von der Bildgröße und der Häufigkeit des Auftretens kann dieses Verfahren enorm viel Speicherplatz sparen.

Skalierung und dpi-Berechnung. PDFlib ändert die Pixelanzahl eines importierten Bildes nie. Beim Skalieren werden die Bildpunkte entweder vergrößert oder verkleinert, es findet jedoch keinerlei Downsampling statt. Der Skalierungsfaktor 1 bewirkt eine Pixelgröße von einer Einheit in Benutzerkoordinaten. Anders ausgedrückt: das Bild wird mit seiner Originalauflösung (oder 72 dpi, falls das Bild keine Auflösungsangabe enthält) importiert, wenn das Benutzerkoordinatensystem nicht skaliert wurde (da ein Zoll 72 Punkt entspricht).

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `images/image_dimensions`. *Es zeigt, wie Sie die Maße eines Bildes abfragen und das Bild in verschiedenen Größen platzieren.*

Farbraum importierter Bilder. Außer beim Hinzufügen oder Entfernen von ICC-Profilen oder Anwenden einer Schmuckfarbe entsprechend der Optionen für *PDF_load_image()* versucht PDFlib in der Regel, den Farbraum importierter Bilder zu erhalten. Dies ist jedoch bei einigen seltenen Kombinationen nicht möglich, zum Beispiel bei YCbCr in TIFF, was nach RGB konvertiert wird.

PDFlib führt keine Konvertierung zwischen RGB und CMYK durch. Wo dies erforderlich ist, muss die Konvertierung vor dem Laden des Bildes in PDFlib erfolgen.

Mehrseitige Bilddateien. PDFlib unterstützt Rasterbilder vom Typ GIF, TIFF und JBIG2, die aus mehr als einem Bild bestehen, sogenannte mehrseitige Bilddateien. Verwenden Sie für mehrseitige Bilddateien die Option *page* von *PDF_load_image():*

```
image = p.load_image("tiff", filename, "page=2");
```

Die Option *page* gibt an, dass eine Datei mit mehreren Bildern verwendet werden soll und wählt das zu importierende Bild aus, beginnend mit 1. Diese Option kann solange hochgesetzt werden, bis *PDF_load_image()* mit dem Wert -1 signalisiert, dass keine weiteren Bilder in der Datei vorhanden sind.

Cookbook *Ein vollständiges Codebeispiel zur Konvertierung aller Bilder aus einer TIFF-Datei mit mehreren Bildern in eine mehrseitige PDF-Datei finden Sie im Cookbook-Topic* `images/multi_page_tiff`.

Inline-Bilder. Im Gegensatz zu wiederverwendbaren Bildern, die als XObjects vom Typ Image in die PDF-Ausgabe geschrieben werden, werden Inline-Bilder direkt in den entsprechenden Content-Stream (Beschreibung von Seite, Pattern, Template oder Glyphe) aufgenommen. Dies spart zwar Speicherplatz, sollte aber nur bei geringer Bildgröße (maximal 4 KB) verwendet werden. Inline-Bilder werden hauptsächlich für Bitmaps in Typ-3-Fonts benutzt und sind sonst nicht empfehlenswert.

Inline-Bilder können mit *PDF_load_image()* und der Option *inline* generiert werden. Sie sind nicht wiederverwendbar, das heißt, das zugehörige Handle darf nicht an Funktionen übergeben werden, die Image-Handles akzeptieren. Deshalb führt die Funktion *PDF_load_image(),* wenn sie mit der Option *inline* aufgerufen wird, intern Operationen durch, die folgenden Anweisungen entsprechen:

```
p.fit_image(image, 0, 0, "");
p.close_image(image);
```

Inline-Bilder werden nur für *imagetype=ccitt, jpeg* und *raw* unterstützt. Bei anderen Bildtypen wird die Option *inline* ignoriert.

OPI-Unterstützung. Beim Laden eines Bildes können im Aufruf von *PDF_load_image()* weitere Informationen für OPI (Open Prepress Interface) Version 1.3 oder 2.0 übergeben werden. PDFlib akzeptiert alle Standard-PostScript-Kommentare für OPI 1.3 und 2.0 als Optionen (nicht die entsprechenden PDF-Schlüsselwörter!) und bettet die übergebenen OPI-Informationen unverändert an die generierte PDF-Ausgabe ein. Das folgende Beispiel ergänzt ein Bild um OPI-Informationen:

```
String  optlist13 =
        "OPI-1.3 { ALDImageFilename bigfile.tif " +
        "ALDImageDimensions {400 561} " +
        "ALDImageCropRect {10 10 390 550} " +
        "ALDImagePosition {10 10  10 540  390 540  390 10} }";

image = p.load_image("tiff", filename, optlist13);
```

XMP-Metadaten in Bildern. Bilddateien können XMP-Metadaten enthalten. Um die Größe der Ausgabedatei zu reduzieren, ignoriert PDFlib standardmäßig Bild-Metadaten von Bildern im Format TIFF, JPEG und JPEG 2000. Mit der folgenden Option von *PDF_load_image( )* lassen sich jedoch die XMP-Metadaten im PDF-Ausgabedokument an das generierte Bild anhängen:

```
metadata={keepxmp=true}
```

7.1.2 Unterstützte Rasterbildformate

PDFlib verarbeitet alle Rasterbildformate, die unten beschrieben werden. Standardmäßig reicht PDFlib die komprimierten Bilddaten, sofern dies möglich ist, unverändert an die PDF-Ausgabe weiter, da PDF intern die meisten in Rasterbildformaten verwendeten Kompressionsverfahren unterstützt. Diese Technik (in den folgenden Beschreibungen *Pass-Through-Modus* genannt) bewirkt einen äußerst schnellen Bildimport, da die Dekompression und die darauf folgende erneute Kompression der Bilddaten entfallen. In diesem Modus kann PDFlib die Integrität der komprimierten Bilddaten nicht überprüfen. Unvollständige oder beschädigte Daten können zu Fehler- oder Warnmeldungen führen, wenn das PDF-Dokument in Acrobat angezeigt wird (zum Beispiel *Nicht genügend Daten für ein Bild)*. Der Pass-Through-Modus lässt sich in gewissem Maße mit der Option *passthrough* von *PDF_load_image( )* steuern.

Wenn eine Rasterbilddatei nicht erfolgreich importiert werden konnte, gibt *PDF_load_image( )* einen Fehlercode zurück. Um Einzelheiten über die Ursachen zu erfahren, können Sie mit *PDF_get_errmsg( )* eine detaillierte Fehlermeldung abrufen.

PNG-Bilder. PDFlib unterstützt alle Varianten von PNG-Bildern (ISO 15948). PNG-Bilder werden meistens im Pass-Through-Modus verarbeitet. Enthält ein PNG-Bild Transparenzinformation, so bleibt diese im generierten PDF erhalten (siehe Abschnitt 7.1.4, »Bildmasken und Transparenz«, Seite 190).

Wenn ein PNG-Bild einen sRGB-Chunk enthält, wird das ICC-Profil für sRGB an das Bild angehängt, sofern die Option *honoriccprofile* nicht auf *false* gesetzt ist oder mit der Op-tion *iccprofile* nicht bereits ein anderes ICC-Profil an das Bild angehängt wurde. Wenn die Option *renderingintent* nicht übergeben wurde, wird der Rendering-Intent im sRGB-Teil verwendet.

JPEG-Bilder. PDFlib unterstützt folgende Varianten von JPEG-Bildern (ISO 10918-1):

- Graustufen, RGB-Farbe (gewöhnlich als YCbCr kodiert) und CMYK-Farbe
- *Baseline*-Kompressionsmodus von JPEG, der in den meisten JPEG-Bildern verwendet wird
- *Progressive*-Kompressionsmodus von JPEG

JPEG-Bilder können in unterschiedlichen Dateiformaten verpackt sein. PDFlib unterstützt alle gängigen JPEG-Dateiformate und -Funktionen:

- JFIF, das von einer Vielzahl von Anwendungen unterstützt wird.
- JPEG-Dateien, die von Adobe Photoshop und anderen Adobe-Anwendungen erzeugt wurden.
- PDFlib liest Beschneidungspfade aus JPEG-Bildern ein, die mit Adobe Photoshop erstellt wurden.
- PDFlib interpretiert eingebettete ICC-Profile in JPEG-Bildern, sofern die Option *honoriccprofile* nicht auf *false* gesetzt ist.
- Wenn das JPEG-Bild einen Exif-Marker enthält, wird die Farbraum-Information im Exif-Marker interpretiert. Wird der sRGB-Farbraum gefunden, wird das ICC-Profil für sRGB an das Bild angehängt (vorausgesetzt, das Bild enthält kein explizit eingebettetes ICC-Profil, die Option *honoriccprofile* ist nicht auf *false* gesetzt und dem Bild wurde kein anderes ICC-Profil mit der Option *iccprofile* zugewiesen).
- Der Eintrag zur Bildorientierung in einem Exif-Marker, der die gewünschte Ausrichtung des Bildes angibt, wird ausgewertet. Mit der Option *ignoreorientation* kann er ignoriert werden (was bei vielen Anwendungen der Fall ist).

JPEG-Bilder sind immer komprimiert, aber manche Varianten müssen für die korrekte Anzeige in Acrobat transkodiert werden. Bei der Transkodierung bleiben Pixelzahl oder Farbe eines Bildes unverändert, da keine vollständige Dekompression mit anschließender erneuter Kompression stattfindet, entstehen keine sichtbaren Artefakte. Bei bestimmten Arten von JPEG-Bildern erfolgt die Transkodierung in PDFlib automatisch, sie kann aber auch über die Option *passthrough* für *PDF_load_image( )* gesteuert werden.

JPEG-2000-Bilder. JPEG-2000-Bilder (ISO 15444-2) erfordern PDF ab Version 1.5 und werden immer im Pass-Through-Modus verarbeitet. PDFlib unterstützt JPEG-2000-Bilder wie folgt:

- Es werden JP2- und JPX-Baseline-Bilder (üblicherweise **.jp2* oder **.jpf)* unterstützt, wobei die Bedingungen bezüglich des Farbraums berücksichtigt werden (siehe unten). Alle Werte für die Farbtiefe im Bereich 1-38 werden unterstützt.
  Es werden folgende Farbräume unterstützt: sRGB, sRGB-grey, ROMM-RGB, sYCC, e-sRGB, e-sYCC, CIELab, ICC-basierte Farbräume und CMYK. PDFlib verändert den ursprünglichen Farbraum der JPEG-2000-Bilddatei nicht.
- Externe ICC-Profile können auf ein JPEG-2000-Bild nicht angewendet werden, d.h. die Option *iccprofile* darf nicht benutzt werden. In der JPEG-2000-Bilddatei vorhandene eingeschränkte oder volle ICC-Profile bleiben immer erhalten, d.h. die Option *honoriccprofile* ist immer auf *true* gesetzt.

*Hinweis JPM-Bilddateien gemäß ISO 15444-6 (meist *.jpm) werden nicht unterstützt.*

Zusätzliche Einschränkungen bei JPEG 2000 für PDF/X-4/5 (JPEG 2000 ist für PDF/X-3, das auf PDF 1.4 basiert, nicht erlaubt):

- Die Anzahl der Farbkanäle muss 1, 3 oder 4 betragen.
- Die Bit-Tiefe jedes Farbkanals muss bei 1, 8 oder 16 liegen.
- Alle Farbkanäle müssen die gleiche Bit-Tiefe haben.
- Genau eine Farbraum-Definition muss in der JPEG-2000-Bilddatei vorhanden sein.
- CMYK-Bilder können nur verwendet werden, wenn die Ausgabebedingung ein CMYK-Gerät ist oder die Option *defaultcmyk* von *PDF_begin_page_ext( )* übergeben wurde.

Zusätzliche Einschränkungen bei JPEG 2000 für PDF/A-2 (JPEG 2000 ist für PDF/A-1, das auf PDF 1.4 basiert, nicht erlaubt):

- Die Anzahl der Farbkanäle muss 1, 3 oder 4 betragen.
- Alle Farbkanäle müssen die gleiche Bit-Tiefe haben.
- Wenn mehr als eine Farbraum-Definition in der JPEG-2000-Bilddatei vorhanden ist, muss genau eine Farbraum-Definition im Feld *APPROX* den Wert 0x01 haben.

JBIG2-Bilder. PDFlib unterstützt ein- und mehrseitige JBIG2-Bilder (ISO 14492). JBIG2-Bilder enthalten immer Pixeldaten in schwarzweiß.

Aufgrund der Beschaffenheit der JBIG2-Komprimierung können sich mehrere Seiten in einem mehrseitigen JBIG2-Stream auf dieselben globalen Segmente beziehen. Wenn mehr als eine Seite eines mehrseitigen JBIG2-Streams konvertiert wird, können die globalen Segmente von den generierten PDF-Bildern gemeinsam genutzt werden. Da die Aufrufe über *PDF_load_image()* unabhängig voneinander sind, müssen Sie PDFlib folgendermaßen im Voraus darüber informieren, dass mehrere Seiten aus demselben JBIG2-Stream konvertiert werden:

- Beim Laden der ersten Seite werden alle globalen Segmente in das PDF kopiert. Verwenden Sie die folgende Optionsliste für *PDF_load_image():*

  ```
  page=1 copyglobals=all
  ```

- Beim Laden nachfolgender Seiten aus demselben JBIG2-Stream müssen Sie das Image-Handle <*N*> für die Seite 1 übergeben, damit PDFlib Verweise auf die globalen Segmente erzeugen kann, die bereits mit Seite 1 kopiert wurden. Verwenden Sie die folgende Optionsliste für *PDF_load_image():*

  ```
  page=2 imagehandle=<N>
  ```

Die Client-Anwendung muss sicherstellen, dass *copyglobals/imagehandle* nur auf solche Seiten angewendet wird, die aus demselben JBIG2-Stream extrahiert werden. Ohne die Optionen *copyglobals* kopiert PDFlib automatisch alle erforderlichen Daten für die aktuelle Seite.

GIF-Bilder. PDFlib unterstützt alle GIF-Varianten (insbesondere GIF 87a und 89a) mit *interlaced* und *non-interlaced* Pixeldaten sowie jede Palettengröße. GIF-Bilder werden immer mit Flate-Kompression komprimiert.

TIFF-Bilder. PDFlib verarbeitet TIFF-Bilder aller wichtigen Varianten:

- Kompressionsverfahren: nicht komprimiert, CCITT (Gruppe 3, Gruppe 4 und RLE), ZIP (=Flate) und PackBits (=RunLength), LZW und JPEG (alte und neue Variante), sowie andere seltene Kompressionsverfahren;
- Farbraum: schwarzweiß, Graustufen, RGB, CMYK, CIELab und YCbCr; der Farbraum in importierten TIFF-Bildern bleibt mit der folgenden Ausnahme unverändert erhalten: LZW-komprimierte TIFF-Bilder mit CIELab-Farbe werden zu RGB konvertiert, der CIELab-Farbraum bleibt dabei nicht erhalten.
- Die Farbtiefe muss bei 1, 2, 4, 8 oder 16 Bit pro Farbkomponente liegen. 16-Bit-Bilder benötigen PDF 1.5.
- Das Format BigTIFF, das das ursprüngliche TIFF-Format über die 4-GB-Grenze hinaus erweitert.

Die folgenden TIFF-Eigenschaften werden beim Import eines Bildes berücksichtigt:

- TIFF-Dateien, die mehrere Bilder enthalten (siehe Abschnitt »Mehrseitige Bilddateien«, Seite 184); verwenden Sie die Option *page*, um ein bestimmtes Bild in einer TIFF-Datei auszuwählen.
- Alphakanäle oder Masken werden interpretiert, sofern die Option *ignoremask* nicht gesetzt ist. Mit der Option *alphachannelname* können Sie eine Maske explizit auswählen. Für weitere Informationen zu Masken siehe Abschnitt 7.1.4, »Bildmasken und Transparenz«, Seite 190.
- PDFlib liest Beschneidungspfade aus TIFF-Bildern ein, die mit Adobe Photoshop oder anderen kompatiblen Anwendungen erstellt wurden, sofern die Option *ignoreclippingpath* nicht gesetzt ist.
- PDFlib interpretiert eingebettete ICC-Profile in TIFF-Bildern, sofern die Option *honoriccprofile* nicht auf *false* gesetzt ist.
- Wenn das TIFF-Bild einen Exif-Marker enthält, wird die Farbraum-Information im Exif-Marker interpretiert. Wird der sRGB-Farbraum gefunden, wird das ICC-Profil für sRGB an das Bild angehängt (vorausgesetzt, das Bild enthält kein explizit eingebettetes ICC-Profil, die Option *honoriccprofile* ist nicht auf *false* gesetzt und kein anderes ICC-Profil wurde dem Bild mit der Option *iccprofile* zugewiesen).
- PDFlib interpretiert das *orientation*-Tag, mit dem die gewünschte Bildorientierung festgelegt wird. Durch Setzen der Option *ignoreorientation* auf *true* kann PDFlib angewiesen werden, dieses Tag zu ignorieren (was in vielen Anwendungen der Fall ist).

Manche TIFF-Eigenschaften (zum Beispiel Schmuckfarbe) und bestimmte Merkmalskombinationen werden nicht unterstützt.

BMP-Bilder. PDFlib unterstützt folgende Varianten von BMP-Bildern:
- BMP-Version 2 und 3
- Farbtiefe von 1, 4 und 8 Bit pro Komponente, einschließlich 3 x 8 = 24 Bit TrueColor. 16-Bit-Bilder werden als 5+5+5 behandelt (1 Bit wird nicht genutzt). 32-Bit-Bilder werden als 3 x 8-Bit-Bilder behandelt (die übrigen 8 Bit werden ignoriert).
- schwarzweiß und RGB-Farbe (indiziert oder direkt);
- unkomprimiert sowie 4-Bit- und 8-Bit-RLE-Kompression;
- PDFlib spiegelt keine Bilder, deren Pixel in »Bottom-Up«-Reihenfolge gespeichert sind (dabei handelt es sich um eine in BMP selten genutzte Funktion, die von Anwendungen nicht einheitlich interpretiert wird).

CCITT-Bilder und Rohdaten. Mit Gruppe 3 oder Gruppe 4 komprimierte Faxdaten werden prinzipiell im Pass-Through-Modus verarbeitet. Beachten Sie dabei, dass sich die Bezeichnung *CCITT-Bilder* auf mit CCITT komprimierte Rohbilddaten bezieht und *nicht* auf TIFF-Dateien mit CCITT-Kompression. Mit CCITT komprimierte Rohbilddaten werden in Anwendungen für Endbenutzer normalerweise nicht unterstützt und können nur mit Faxsoftware generiert werden.

Nicht komprimierte Bilddaten (Rohbilddaten) können in besonderen Fällen nützlich sein. Die Art des Bildes leitet sich aus der Anzahl der Farbkomponenten ab: Eine Komponente impliziert ein Graustufenbild, drei Komponenten ein RGB-Bild und vier Komponenten ein CMYK-Bild.

CCITT- oder Rohbilddaten müssen Sie unter Angabe der Optionen *width, height, components* und *bpc* übergeben, da PDFlib diese nicht aus den Bilddaten ableiten kann. Sie müssen dafür sorgen, dass die Optionswerte zu dem jeweiligen Bild passen. Ansons-

ten kann es zu fehlerhafter PDF-Ausgabe kommen und Acrobat kann folgende Meldung ausgeben: *Nicht genügend Daten für ein Bild*.

Bei *imagetype=raw* muss die Länge der übergebenen Bilddaten gleich *[width x components x bpc / 8] x height* Bytes sein, wobei der Klammerausdruck auf die nächste Ganzzahl nach oben korrigiert wird. Die Pixel müssen von oben nach unten und von links nach rechts angewendet werden (sofern keine Koordinatentransformationen angewendet wurden). 16-Bit-Bilder müssen mit dem höherwertigen Byte zuerst angegeben werden (Big-Endian-Bytereihenfolge). Die Polarität der Pixelwerte ist die gleiche wie bei den Farboptionen (siehe PDFlib-Referenz). Wenn *bpc* kleiner ist als 8, beginnt jede Pixelreihe auf einer Byte-Grenze, und Farbwerte müssen von links nach rechts in einem Byte gepackt sein. Die Farbwerte der Pixel sind immer verschachtelt, das heißt, zuerst werden alle Farbwerte für das erste Pixel übergeben, anschließend alle Farbwerte für das zweite Pixel usw.

7.1.3 Beschneidungspfade

Beschneidungspfade *(clipping paths)* werden von PDFlib in TIFF- und JPEG-Bildern unterstützt, die mit Adobe Photoshop erstellt wurden. Eine Bilddatei kann mehrere benannte Pfade enthalten. Mit der Option *clippingpathname* von *PDF_load_image( )* kann ein benannter Pfad ausgewählt und damit als Beschneidungspfad verwendet werden: Nur Bildbereiche innerhalb des Beschneidungspfades bleiben dann sichtbar, während alle Bereiche außerhalb ausgeblendet werden. Diese Funktion eignet sich zum Beispiel zur Trennung von Vorder- und Hintergrund, zum Verbergen unerwünschter Bildbereiche usw.

Alternativ dazu kann in der Bilddatei ein Standard-Beschneidungspfad definiert sein. Entdeckt PDFlib diesen in einer Bilddatei, so wird er automatisch auf das Bild angewendet (siehe Abbildung 7.1). Wenn Sie dies verhindern möchten, setzen Sie die Option *honorclippingpath* in *PDF_load_image( )* auf *false*. Falls Sie ein Bild mehrfach verwenden wollen, der Beschneidungspfad aber nicht auf alle Instanzen angewendet werden soll, können Sie in *PDF_fit_image( )* die Option *ignoreclippingpath* übergeben, um den Beschneidungspfad zu deaktivieren. Bei der Anwendung eines Beschneidungspfades werden alle Berechnungen zur Bildplatzierung anhand der Bounding-Box des freigestellten Bildes vorgenommen.

Abb. 7.1
Hintergrund durch Beschneidungspfad vom Vordergrund trennen

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `images/integrated_clipping_path`.

Bei jedem Aufruf von *PDF_fit_image()* werden die vektorbasierten Operationen zur Beschreibung des Beschneidungspfades in die PDF-Ausgabe geschrieben. Wenn ein Bild mit Beschneidungspfad mehrfach im Dokument vorkommt, empfehlen wir, das Bild in einem Template zu verpacken, um die Größe der Ausgabedatei zu reduzieren. Verwenden Sie dazu die Option *createtemplate* von *PDF_load_image()*.

7.1.4 Bildmasken und Transparenz

PDFlib unterstützt drei Arten der Transparenzinformation in Bildern: implizite Transparenz, explizite Transparenz und Bildmasken.

Implizite Transparenz mit Alphakanälen. Rasterbilder können teilweise transparent sein, das heißt, der Hintergrund scheint durch das Bild hindurch. Dies ist beispielsweise nützlich, wenn der Hintergrund eines Bildes ausgeblendet werden soll und nur die Personen oder Objekte im Vordergrund angezeigt werden sollen. Transparenzinformation kann in einem separaten Alphakanal oder (in palettenbasierten Bildern) als transparenter Palet-teneintrag gespeichert werden. Transparente Bilder sind für PDF/A-1, PDF/X-1 und PDF/X-3 nicht erlaubt. Transparenzinformation wird bei PDFlib in folgenden Dateiformaten erkannt:

- GIF-Bilder können einen einzelnen transparenten Farbwert (Paletteneintrag) enthalten, der von PDFlib berücksichtigt wird.
- TIFF-Bilder können einen einzelnen verknüpften Alphakanal enthalten, der in PDFlib ausgewertet wird. Alternativ dazu kann ein TIFF-Bild eine beliebige Anzahl von nicht verknüpften Kanälen enthalten, die anhand ihres Names identifiziert werden. Diese Kanäle können zur Übermittlung von Transparenz- oder anderen Informationen verwendet werden. Wenn nicht verknüpfte Kanäle in einem TIFF-Bild gefunden werden, verwendet PDFlib automatisch den ersten Kanal als Alphakanal. Sie können jedoch einen nicht verknüpften Alphakanal direkt durch Angabe seines Namens auswählen:

  ```
  image = p.load_image("tiff", filename, "alphachannelname={apple}");
  ```

- PNG-Bilder können einen verknüpften Alphakanal enthalten, der automatisch von PDFlib verwendet wird.
- Als Alternative zu einem vollständigen Alphakanal können PNG-Bilder einen einzelnen transparenten Farbwert enthalten, der in PDFlib erhalten bleibt. Liegen mehrere Farbwerte in einem beigefügten Alphakanal vor, wird nur der erste mit einem Alphawert unter 50 Prozent verwendet.

Hinweis *Photoshop kann zusätzlich zu einem vollständigen Alphakanal transparente Hintergründe in einem proprietären Format erzeugen, das von PDFlib nicht ausgewertet wird. Um solche transparenten Bilder mit PDFlib zu verwenden, müssen Sie sie mit Photoshop im Dateiformat TIFF speichern und* `Transparenz speichern` *im Dialogfeld TIFF-Optionen auswählen.*

Manchmal ist es wünschenswert, alle Transparenzinformationen in einer Bilddatei zu ignorieren. Die automatische Interpretation von Transparenzinformationen in PDFlib beim Öffnen der Bilddatei kann mit der Option *ignoremask* deaktiviert werden:

```
image = p.load_image("tiff", filename, "ignoremask");
```

Acrobat 7/8/9 kann 16-Bit-Bilder mit einem Alphakanal nicht korrekt verarbeiten, sondern interpretiert den 16-Bit-Alphakanal als einen 8-Bit-Alphakanal. Dieser Fehler tritt ab Acrobat X nicht mehr auf. Um dieses Problem zu umgehen, verwenden Sie die Option *downsamplemask*. Ist die Option auf *true* gesetzt, rechnet PDFlib den Alphakanal von 16 Bit auf 8 Bit herunter. Standardmäßig wird jedoch kein Downsampling durchgeführt, was in älteren Versionen von Acrobat zu einer falschen Darstellung führen kann.

Explizite Transparenz. Im expliziten Fall sind zwei Schritte erforderlich, die jeweils getrennte Bildoperationen erfordern. Zuerst muss ein Graustufenbild zur späteren Verwendung als Transparenzmaske vorbereitet werden. Dies lässt sich durch Öffnen des Bildes mit der Option *mask* bewerkstelligen. Die folgenden Arten von Bildern können zum Anlegen einer Maske verwendet werden:

- PNG-Bilder
- TIFF-Bilder: die Option *nopassthrough* von *PDF_load_image()* wird empfohlen, um Bilder vom Typ *multi-strip* zu vermeiden.
- rohe Bilddaten

Überall dort, wo sich in der Maske der Pixelwert 0 (null) befindet, wird der entsprechende Bereich des maskierten Bildes gezeichnet, während hohe Pixelwerte den Hintergrund durchscheinen lassen. Ist mehr als 1 Bit pro Pixel vorhanden, mischen die zwischenliegenden Werte das Vordergrundbild mit dem Hintergrund und bewirken so einen Transparenzeffekt.

Im zweiten Schritt wird diese Maske auf ein anderes Bild mit der Option *masked* angewendet:

```
mask = p.load_image("png", maskfilename, "");
if (mask == -1)
        throw new Exception("Error: " + p.get_errmsg());

String optlist = "masked=" + mask;
image = p.load_image(type, filename, optlist)
if (image == -1)
        throw new Exception("Error: " + p.get_errmsg());

p.fit_image(image, x, y, "");
```

Bild und Maske können unterschiedliche Pixelmaße aufweisen; die Maske wird automatisch auf die Bildgröße skaliert.

Hinweis *PDFlib konvertiert »multi-strip« TIFF-Bilder in mehrere PDF-Bilder, die dann einzeln maskiert würden. Da dies in der Regel aber nicht beabsichtigt ist, wird diese Art von Bildern sowohl als Maske als auch zur Maskierung abgelehnt. Außerdem dürfen impliziter und expliziter Fall nicht vermischt, also keine Bilder mit transparenten Farbwerten als Maske verwendet werden.*

Hinweis *Die Maske muss dieselbe Ausrichtung wie das zugrunde liegende Bild haben; andernfalls wird sie zurückgewiesen. Da die Ausrichtung vom Dateiformat des Bildes sowie anderen Faktoren abhängt, ist sie nicht einfach zu entdecken. Aus diesem Grund ist es empfehlenswert, die Maske und das Bild mit derselben Software in demselben Dateiformat zu erstellen.*

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `images/image_mask`.

Bildmasken und Transparenzmasken. Bildmasken sind Bilder mit einer Bit-Tiefe von 1 (Bitmaps), in denen die Bits mit dem Wert 0 als transparent behandelt werden: Der auf der Seite vorhandene Inhalt scheint durch die transparenten Bestandteile des Bildes hindurch. Pixel mit dem Wert 1 dagegen werden mit der aktuellen Füllfarbe eingefärbt.

Transparenzmasken erweitern das Konzept der Bildmasken auf Masken mit mehr als einem Bit. Sie mischen ein Bild mit einem vorhandenen Hintergrund. PDFlib akzeptiert alle Arten von einkanaligen (Graustufen-)Bildern als Transparenzmasken. Beachten Sie, dass nur richtige Graustufen-Bilder als Masken geeignet sind, nicht aber Bilder mit indizierter (palettenbasierter) Farbe. Sie sind wie Bildmasken einsetzbar. Die folgenden Arten von Bildern können als Bildmasken verwendet werden:

- PNG-Bilder
- JBIG2-Bilder
- TIFF-Bilder (*single-* oder *multi-strip*)
- JPEG-Bilder (nur als Transparenzmasken)
- BMP; beachten Sie, dass BMP-Bilder anders als andere Bildtypen ausgerichtet sind. BMP-Bilder müssen deshalb erst an der x-Achse gespiegelt werden, bevor Sie als Maske einsetzbar sind.
- rohe Bilddaten

Bildmasken werden einfach mit der Option *mask* geöffnet und auf der Seite platziert, nachdem die gewünschte Füllfarbe gesetzt wurde:

```
mask = p.load_image("tiff", maskfilename, "mask");
p.set_graphics_option("fillcolor=red);
if (mask != -1)
{
        p.fit_image(mask, x, y, "");
}
```

Wenn Sie eine Farbe auf ein Bild anwenden möchten, ohne die Bits mit dem Wert 0 transparent zu machen, müssen Sie die Option *colorize* verwenden (siehe Abschnitt 7.1.5, »Einfärben von Bildern«, Seite 192).

7.1.5 Einfärben von Bildern

Ähnlich wie Bildmasken, wo eine Farbe auf die nicht-transparenten Bestandteile eines Bildes angewandt wird, unterstützt PDFlib das Einfärben eines Bildes mit einer Schmuckfarbe.

Bei Bildern, die mit einer RGB-Farbpalette arbeiten, ist ein Einfärben nur sinnvoll, wenn die Palette ausschließlich Graustufenwerte enthält und der Palettenindex dem Graustufenwert entspricht.

Um ein Bild mit einer Schmuckfarbe einzufärben, müssen Sie beim Laden des Bildes die Option *colorize* und ein Handle für die gewünschte Schmuckfarbe übergeben, welches von *PDF_makespotcolor( )* zurückgegeben wurde:

```
spot = p.makespotcolor("PANTONE Reflex Blue CV");

String optlist = "colorize=" + spot;
image = p.load_image("tiff", "image.tif", optlist);
if (image != -1)
{
        p.fit_image(image, x, y, "");
}
```

7.2 SVG-Grafik

7.2.1 Unterstützte SVG-Varianten

PDFlib unterstützt SVG-Grafiken folgendermaßen:

- PDFlib implementiert SVG 1.1 (Second Edition), wie vom W3C veröffentlicht. Für nicht unterstützte Aspekte der SVG-Spezifikation siehe Abschnitt 7.2.7, »Nicht unterstützte SVG-Funktionen«, Seite 200.
- Folgende Unicode-Formate und -Encodings werden unterstützt: UTF-8, UTF-16, ISO 8859-1 ... ISO 8859-15, ASCII
- CSS-Styling ist verfügbar, einige CSS-Elemente werden allerdings nicht unterstützt.
- Zusätzlich zu SVG-Dateien im Klartext-Format werden SVG-Dateien mit Flate-Kompression *(*.svgz)* unterstützt.
- Fonts im Format CEF werden unterstützt. CEF-Fonts sind zwar nicht Teil der SVG-Spezifikation, werden aber von einigen Adobe-Anwendungen in SVG-Grafiken eingebettet.

7.2.2 SVG-Verarbeitung

Grundlegende Vorgehensweise. Die Einbettung von Vektorgrafik ist mit PDFlib leicht zu bewerkstelligen. Zuerst muss die Grafikdatei mit einer PDFlib-Funktion geöffnet werden, die die Grafiken interpretiert und in einer internen Darstellung speichert. Die Funktion *PDF_load_graphics()* gibt ein Handle zurück, das als Grafikdeskriptor dient. Dieses Handle kann im Aufruf von *PDF_fit_graphics()* verwendet werden, zusammen mit Optionen zur Positionierung und Skalierung:

```
graphics = p.load_graphics("auto", "graphics.svg", "");
if (graphics == -1)
        throw new Exception("Error: " + p.get_errmsg());

if (p.info_graphics(graphics, "fittingpossible", optlist) == 1)
        p.fit_graphics(graphics, 0.0, 0.0, "");
else
        System.err.println("Cannot place graphics: " + p.get_errmsg());

p.close_graphics(graphics);
```

Der letzte Parameter von *PDF_fit_graphics()* ist eine Optionsliste, die viele Optionen zur Positionierung, Skalierung und Rotation der Grafik unterstützt. Für weitere Informationen zu diesen Optionen siehe Abschnitt 7.4, »Platzieren von Bildern, Grafiken und importierten PDF-Seiten«, Seite 207.

Cookbook *Codebeispiele zur SVG-Bearbeitung finden Sie in der Kategorie* `graphics` *des PDFlib Cookbook.*

Wiederverwendung von Grafiken im Dokument. PDFlib unterstützt die folgenden Methoden zum Importieren von Vektorgrafiken:

- Grafikdaten werden direkt in den Content-Stream von Seite, Muster, Template oder Glyphenbeschreibung geschrieben. Dies ist das empfohlene Standardverhalten für Grafiken, die genau einmal im Dokument platziert werden. Wenn *PDF_fit_graphics()* mehrfach aufgerufen wird, werden die Grafikdaten wieder und wieder in die PDF-Ausgabe geschrieben, wodurch sich die Ausgabedatei vergrößert.

- Soll dieselbe Grafik mehrfach im Dokument platziert werden, sollten Sie die Option *templateoptions* von *PDF_load_graphics( )* verwenden. Dadurch wird ein sogenanntes PDF-Form-XObject (Template) angelegt, das heißt, die Grafik wird als separate Einheit gespeichert und kann beliebig oft referenziert werden. Die Grafikdaten für das Template werden am Ende in die PDF-Ausgabe des Dokuments geschrieben oder wenn *PDF_close_graphics( )* aufgerufen wird. So wird die Dateigröße optimiert. Links innerhalb der Grafiken werden allerdings nicht mehr in PDF-Anmerkungen umgewandelt.

Wiederverwendung von Grafiken in mehreren Dokumenten. Grafiken können unabhängig vom aktuellen Ausgabedokument geöffnet und geschlossen werden. Wird *PDF_load_graphics( )* aufgerufen, wird eine interne Darstellung der Grafiken angelegt. Bis zum entsprechenden Aufruf von *PDF_close_graphics( )* bleibt sie im Arbeitsspeicher erhalten. Grafiken über mehrere Dokumente hinweg im Speicher zu halten, erhöht die Performance in Situationen, in denen dieselbe Grafik in vielen Ausgabedokumenten platziert wird, da die Grafik nur einmal geladen werden muss. Eine Anwendung kann zum Beispiel Grafiken mit Symbolen, Hintergrundbildern oder Briefpapier einmal laden und diese in jedem erforderlichen Dokument mit *PDF_fit_graphics( )* platzieren.

Probleme bei der SVG-Verarbeitung erkennen. Erst nach dem Laden mit *PDF_load_graphics( )* erfolgt die vollständige Verarbeitung und Analyse der SVG-Grafik, je nach Erzeugung des Form XObjects und der Art des Ladens über die Funktionen *PDF_fit_graphics( ), PDF_close_graphics( )* oder *PDF_end_document( )*. Da einige Fehlersituationen erst während dieser Verarbeitung erkannt werden können, könnten diese Funktionen bei einem Fehler eine entsprechende Exception auslösen (da sie keine Fehlercodes ausgeben können). Um dies zu vermeiden, können die Grafiken mit dem Schlüsselwort *fittingpossible* von *PDF_info_graphics( )* überprüft werden. Es führt alle Verarbeitungsschritte aus, erzeugt aber keine Ausgabe und meldet, ob die SVG-Verarbeitung erfolgreich ist oder nicht. Ist die Prüfung erfolgreich, löst *PDF_fit_graphics( )* keine Exception aus, wenn die Grafik platziert wird. Wenn während der Prüfung durch *fittingpossible* ein Fehler auftritt, gibt *PDF_info_graphics( )* den Wert -1 aus (in PHP: 0), sofern *errorpolicy=return*. Fazit:

- Die Prüfung durch *fittingpossible* vermeidet spätere Exceptions in *PDF_fit_graphics( ), PDF_close_graphics( )* oder *PDF_end_document( )*. Diese Vorgehensweise ist zu empfehlen, da die PDF-Ausgabe nach einer Exception sonst unbrauchbar wäre.
- Ohne die Prüfung durch *fittingpossible* kann die SVG-Grafik zwar schneller geladen werden, jedoch können später durch SVG-Daten ausgelöste Exceptions auftreten. Wenn Exceptions in *PDF_fit_graphics( )* in Kauf genommen werden können, kann mit dieser Einstellung das Laden der SVG-Daten beschleunigt werden. Wenn die Anwendung beispielsweise eine einzelne SVG-Grafikdatei in ein einzelnes PDF-Dokument ohne zusätzliche Seiteninhalte konvertiert, wäre diese Vorgehensweise akzeptabel.

Die Prüfung durch *fittingpossible* verwendet die gerade aktiven globalen Dokument- und Seitenoptionen sowie den aktuellen Output-Intent. Sie sollten diese Prüfung deshalb erst unmittelbar vor dem eigentlichen Aufruf von *PDF_fit_graphics( )* durchführen.

7.2.3 Größe von SVG-Grafiken

SVG-Grafiken definieren die Breite und Höhe im Element *svg*, das die Abbildung einer SVG-Grafik im sogenannten Viewport definiert (zum Beispiel das Browser-Fenster oder ein Teil einer PDF-Seite). Oft wird dieser Viewport in absoluten Einheiten angegeben, zum Beispiel:

```
<svg xmlns="http://www.w3.org/2000/svg" width="640mm" height="480mm">
```

PDFlib konvertiert die Attribute *width* und *height* in Punkte (points) und stellt sie über die Schlüsselwörter *graphicswidth* und *graphicsheight* von *PDF_info_graphics( )* zur Verfügung. Ist die Größe in Pixeln *(px)* angegeben, verwendet PDFlib 1pt=1px. Diese Werte werden auch zur Berechnung des Objektrahmens beim Einpassen verwendet. Das Element *svg* kann auch das Attribut *viewBox* enthalten, das ein Fenster innerhalb des Viewports festlegt.

Manche SVG-Grafiken enthalten jedoch keine absoluten Größenangaben, da *width* und *height* entweder fehlen oder nur relative Werte enthalten, zum Beispiel:

```
<svg xmlns="http://www.w3.org/2000/svg" width="100%" height="100%">
```

In diesem Fall liest PDFlib das Attribut *viewBox* aus, falls vorhanden, und verwendet die dort angegebene Größe für den Anzeigebereich. Diese Werte können überschrieben werden, was besonders nützlich ist, wenn das Attribut *viewBox* fehlt. In diesem Fall können Sie für die Größenangabe die Optionen *fallbackwidth* und *fallbackheight* von *PDF_load_graphics( )* verwenden. Zum Überschreiben von Größenangaben aus Grafikdateien können Sie die Optionen *forcedwidth* und *forcedheight* verwenden.

7.2.4 Fontauswahl

Algorithmus für die Fontauswahl. Die Fontauswahl wird bei SVG durch die folgenden Properties gesteuert:

```
font-family
font-style
font-weight

font-stretch
font-variant
font-size
font-size-adjust
```

Relevant für die Auswahl eines externen Fonts sind nur die ersten drei Properties. PDFlib bildet dazu die folgenden Fontnamen:

```
<font-family>,<font-weight>,<font-style>
<font-family>-<font-weight><font-style>
<font-family>,<font-normweight>,<font-style>
<font-family>,<font-weight>,<font-normstyle>
<font-family>,<font-normweight>,<font-normstyle>
```

<*font-normweight*> hat dabei einen der folgenden Werte:

```
Regular, Thin, Extralight, Light, Medium, Semibold, Bold, Extrabold, Black
```

<font-normstyle> ist:

```
Italic
```

Beispiel für eine SVG-Fontspezifikation:

```
font-family="Tahoma" font-weight="Bold" font-style="Italic"
```

Im obigen Beispiel sucht PDFlib den Font *Tahoma,Bold,Italic,* wobei die Fontstile in der PDFlib-Syntax wie bei der Adressierung von Windows Host-Fonts durch ein Komma voneinander getrennt werden.

PDFlib versucht, nacheinander die oben angegebenen Fontnamen zu laden, bis der Vorgang erfolgreich ist und ein Font geladen werden konnte. Die Fontnamen in dieser Liste können in Ressourcen für Fontspezifikationen verwendet werden, zum Beispiel:

```
p.set_option("FontOutline={<fontname>=<filename>}")
p.set_option("FontNameAlias={<fontname>=ArialMT}")
```

Wenn alle Versuche erfolglos bleiben, versucht PDFlib, den Font mit dem Namen *<font-family>* zu laden und simuliert bei Bedarf die Eigenschaften *Bold* und *Italic*.

Manche Browser ignorieren die Properties zur Fontauswahl, wenn die Fontfamilie nicht gefunden werden kann. Da das bei PDFlib nicht der Fall ist, müssen Sie über die Fontkonfiguration von PDFlib für geeignete Fonts sorgen (siehe Abschnitt 5.4, »Laden von Fonts«, Seite 129).

Die Property *font-family* kann auch mehrere Namen von Fontfamilien enthalten, z.B.:

```
font-family="Georgia, 'Minion Web', 'Times New Roman', Times, 'MS PMincho', serif"
```

Wenn in diesem Fall eine bestimmte Fontfamilie nicht geladen werden kann, versucht PDFlib den nächsten Font aus der Liste zu laden. Wenn ein Font für *font-family* geladen werden konnte, versucht PDFlib, die verbleibenden *font-families* in der Liste als Fallback-Fonts für den zuerst geladenen Font zu verwenden (den sogenannten Master-Font, siehe Abschnitt 5.4.6, »Fallback-Fonts«, Seite 143). Hat der Master-Font bereits Fallback-Fonts, weil er schon früher geladen wurde, werden die neuen Fallback-Fonts an die Liste der existierenden Fallback-Fonts angehängt.

Beim Laden eines Fonts für SVG-Grafiken verwendet PDFlib standardmäßig die folgenden Optionen:

```
embedding skipembedding={latincore standardcjk} subsetting
```

Diese Optionen können mit der Option *defaultfontoptions* von *PDF_load_graphics( )* überschrieben werden.

Fontkonfiguration. Auf Windows-Systemen kann PDFlib auf alle auf dem System installierten Fonts zugreifen (siehe Abschnitt 5.4.5, »Host-Fonts unter Windows und OS X«, Seite 141). Beispiel für eine SVG-Fontspezifikation:

```
font-family="Verdana" font-weight="bold"
```

Das obige Beispiel ergibt bei PDFlib den Fontnamen *Verdana,Bold*. Auf anderen Betriebssystemen findet PDFlib einen Font, wenn eine Ressource vom Typ *FontOutline* folgendermaßen angegeben wurde:

```
<fontnamepattern>=<filename.xxx>
```

<fontnamepattern> ist dabei eins der obigen Muster für Fontnamen und *xxx* ist die entsprechende Dateinamenserweiterung der Fontdatei. Bei Type-1-Fonts muss die Ressource *FontAFM* oder *FontPFM* gesetzt sein.

Geeignete Ressourcen vom Typ *FontOutline* mit Fontnamen, die einem oder mehreren Mustern für Fontnamen oben entsprechen, können automatisch mit der Option *enumeratefonts* von *PDF_set_option( )* erzeugt werden. Für weitere Informationen zur Fontkonfiguration siehe Abschnitt 5.4.4, »Suche nach Fonts«, Seite 136,.

Abbildung generischer SVG-Fontfamilien auf PDF-Standardfonts. PDFlib bildet die generischen SVG-Fontfamilien *monospace*, *sans-serif* und *serif* beim ersten Auftreten auf lateinische Standardfonts ab. Dabei wird die Ressource *FontNameAlias* folgendermaßen angewendet:

```
p.set_option("FontNameAlias={monospace=Courier}")
p.set_option("FontNameAlias={monospace,Bold=Courier-Bold}")
```

Im Folgenden finden Sie die komplette Liste der Zuordnungen für generische Fontnamen (für die generischen Fontfamilien *cursive* und *fantasy* gibt es keine Standard-Zuordnungen):

```
monospace               Courier
monospace,Bold          Courier-Bold
monospace,Italic        Courier-Oblique
monospace,Bold,Italic   Courier-BoldOblique

sans                    Helvetica
sans,Bold               Helvetica-Bold
sans,Italic             Helvetica-Oblique
sans,Bold,Italic        Helvetica-BoldOblique

sans-serif              Helvetica
sans-serif,Bold         Helvetica-Bold
sans-serif,Italic       Helvetica-Oblique
sans-serif,Bold,Italic  Helvetica-BoldOblique

serif                   Times-Roman
serif,Bold              Times-Bold
serif,Italic            Times-Italic
serif,Bold,Italic       Times-BoldItalic
```

Diese Zuordnungen werden nur ausgeführt, wenn vorher keine andere geeignete Ressource durch den Benutzer festgelegt wurde.

7.2.5 Umgang mit fehlenden Fonts und Glyphen

Fehlende Fonts und der Default-Font. Wenn ein Font nicht geladen werden kann, versucht PDFlib einen Standardfont zu laden, bei dem der Fontname *font-family* in der Option *defaultfontfamily* von *PDF_load_graphics( )* definiert ist. Standardmäßig wird der Font *Arial Unicode MS* verwendet, falls vorhanden, ansonsten Helvetica. Wir empfehlen daher dringend, entweder den Font *Arial Unicode MS* in der Fontkonfiguration von PDFlib bereitzustellen oder einen anderen Font mit einem großen Sortiment an Gly-

phen in *defaultfontfamily* festzulegen. Um zum Beispiel den Font *Code2000* in der Fontdatei *CODE2000.TTF* als Ersatzfont zu konfigurieren, verwenden Sie die folgende Option:

```
defaultfontfamily={CODE2000}
```

Ersetzen einzelner Fonts. Um einen Font zu vermeiden, der entweder nicht vorhandenen oder nicht erwünscht ist (weil er zum Beispiel zu wenig Glyphen enthält), können Sie ihn auf einen bereits vorhandenen oder passenderen Font abbilden. Verwenden Sie dazu die Alias-Funktion für Fontnamen und *PDF_set_option( )* (siehe auch Abschnitt »Aliasnamen für Fonts«, Seite 136). Wenn Sie zum Beispiel nicht korrekt gesetzten chinesischen Text im Font *Trebuchet MS* haben, der keine chinesischen Glyphen darstellen kann, können Sie ihn folgendermaßen auf *Arial Unicode MS* abbilden:

```
p.set_option("FontnameAlias={ {Trebuchet MS}={Arial Unicode MS} }");
```

Beachten Sie, dass Fontattribute nicht automatisch hinzugefügt werden. Wenn zum Beispiel der Font *Trebuchet MS* mit dem Attribut *font-weight="bold"* verwendet wird, müssen Sie für die fette Variante des Fonts einen Alias erzeugen:

```
p.set_option("FontnameAlias={ {Trebuchet MS,Bold}={Arial Unicode MS} }");
```

Visualisieren fehlender Glyphen. Wenn in einem ausgewählten Font eine erforderliche Glyphe fehlt, wird stattdessen die Standard-Ersatzglyphe verwendet, was bedeutet, dass mit den Standardeinstellungen überhaupt kein Text sichtbar ist. Um fehlende Glyphen zu visualisieren, können Sie eine sichtbare Ersatzglyphe mit der Option *defaultfontoptions* festlegen. Die folgende Option für *PDF_load_graphics( )* stellt zum Beispiel alle fehlenden Glyphen als Fragezeichen dar:

```
defaultfontoptions={replacementchar=?}
```

Festlegen eines Fallback-Fonts für fehlende Glyphen. Wenn der in der SVG-Grafik ausgewählte Font keine zum Text passenden Glyphen enthält, wird kein Text angezeigt. Ein typisches Beispiel ist chinesischer Text, der in einem westlichen Font ohne chinesische Glyphen angezeigt werden soll. Die beste Lösung wäre natürlich, geeignete Fonts im SVG zu verwenden. Bei ungeeigneten Fonts, hingegen, können Sie in PDFlib einen Fallback-Font festlegen, der immer dann verwendet wird, wenn im ursprüngliche Font die erforderliche Glyphe fehlt.

Die folgende Option für *PDF_load_graphics( )* legt *Arial Unicode MS* als Fallback-Font fest:

```
defaultfontoptions={fallbackfonts={{fontname={Arial Unicode MS} encoding=unicode}}}
```

Beachten Sie, dass die im Font angegebene Option *defaultfontfamily* wie oben beschrieben einen nicht vorhandenen Font komplett ersetzt, während die Fallback-Technik angewendet wird, wenn ein Font zwar verfügbar ist, jedoch nicht alle benötigten Glyphen enthält.

Festlegen einer globalen Fallback-Fontfamilie. Mit den Optionen *fallbackfontfamily* und *fallbackfontoptions* können Sie eine Familie von Fallback-Fonts und entsprechende Optionen festlegen. Während mit der Option *fallbackfonts* in *defaultfontoptions* ein einzelner Font als Fallback-Font ausgewählt wird, kann mit *fallbackfontfamily* eine ganze Fontfamilie als Fallback-Fonts festgelegt werden. Dabei werden Stilattribute auf den Na-

men dieser Fontfamilie angewendet, sofern die benötigten Stilvarianten darin überhaupt vorhanden sind. Beispiel:

```
fallbackfontfamily={Arial} fallbackfontoptions={encoding=unicode}
```

7.2.6 Weitere SVG-Inhalte

Eingebettete Rasterbilder in SVG. PDFlib verarbeitet in SVG das Element *image* und akzeptiert alle Bildformate, die in Abschnitt 7.1, »Rasterbilder«, Seite 183 beschrieben werden. Außerdem werden verschachtelte SVG-Grafiken unterstützt. Bilddaten können in die SVG-Datei eingebettet sein oder in einer externen Datei vorliegen.

Rasterbilder in SVG-Grafiken werden automatisch verarbeitet. Wenn Sie jedoch bestimmte Optionen zur Bildverarbeitung übergeben möchten, können Sie dazu die Option *defaultimageoptions* von *PDF_load_graphics( )* verwenden. Mit der folgenden Option können Sie Antialiasing anwenden, das die Darstellung von Bildern mit geringer Auflösung verbessert (dies entspricht der SVG-Anzeige der meisten Browser, die Antialiasing bei Rasterbildern anwenden):

```
defaultimageoptions={interpolate}
```

Ist ein Bild nicht verfügbar (weil zum Beispiel die referenzierte externe Bilddatei fehlt), erzeugt PDFlib ein transparentes graues Schachbrettmuster. Mit der Option *fallbackimage* können Sie dieses Muster anpassen oder ein benutzerdefiniertes Bild oder Template als Ersatzdarstellung angeben.

Farbe in SVG. Entsprechend der SVG-Spezifikation werden für Text und Vektorgrafik in SVG standardmäßig im sRGB-Farbraum Farbangaben interpretiert. Das bedeutet, dass SVG-Grafiken bei PDF/X und PDF/A als geräteunabhängige Farbe behandelt werden. Denken Sie daran, dass in SVG eingebettete Rasterbilder andere Farbräume verwenden können. Ein eingebettetes oder referenziertes JPEG-Bild kann zum Beispiel den CMYK-Farbraum mit oder ohne ein ICC-Profil verwenden.

Wenn die Option *devicergb* an *PDF_load_graphics( )* übergeben wurde, werden SVG-Grafiken in einfachen RGB-Farben dargestellt, was bei PDF/A und PDF/X manchmal nicht erlaubt ist.

Links in SVG. Links in SVG-Grafiken werden in der erzeugten PDF-Ausgabe normalerweise in interaktive Anmerkungen vom Typ Link umgewandelt; unter manchen Umständen wird die Erzeugung von Links jedoch deaktiviert (siehe PDFlib-Referenz). Externe Links werden ignoriert. Die Option *contents* der PDF-Anmerkung wird mit dem Attribut *xlink:title* des SVG-Links versehen, sofern vorhanden, ansonsten mit der Ziel-URI. Bei Tagged PDF wird das Element *Link* und ein zugehöriges Element OBJR für den generierten Link erzeugt, sofern es sich beim aktiven Element nicht um ein Artefakt oder Pseudo-Element handelt. Mit der Option *convertlinks* von *PDF_fit_graphics( )* kann die Umwandlung von SVG-Links in PDF-Links deaktiviert werden.

Metadaten in SVG. SVG-Grafik kann XMP-Metadaten enthalten. PDFlib ignoriert SVG-Metadaten in Grafiken, um die Größe der Ausgabedatei zu reduzieren. Wenn jedoch aus SVG ein Template erzeugt wird, können die XMP-Metadaten mit der folgenden Option von *PDF_load_graphics( )* an das erzeugte XObject angehängt werden:

```
templateoptions={metadata={keepxmp=true}}
```

Die Inhalte der Elemente *metadata*, *desc* und *title* einer SVG-Grafik können nach folgendem Muster mit *PDF_info_graphics( )* abgefragt werden:

```
idx = (int) p.info_graphics(svg, "description", "");
if (idx != -1)
        description = p.get_string(idx, "");
```

7.2.7 Nicht unterstützte SVG-Funktionen

Umgang mit nicht unterstützten Funktionen. Nicht unterstützte SVG-Funktionen werden von PDFlib ignoriert. Es wird zwar eine Ausgabe erzeugt, manche Bereiche der Grafik können aber fehlerhaft sein oder gar ganz fehlen. Dieses Verhalten kann mit der Option *errorconditions* von *PDF_load_graphics( )* geändert werden. Die Unteroptionen legen Bedingungen fest, die einen Fehler auslösen statt ignoriert zu werden. Zum Beispiel schlägt *PDF_load_graphics( )* mit der folgenden Optionsliste fehl, wenn die SVG-Grafiken animierte Elemente oder Script-Elemente enthalten:

```
errorconditions = {element={animate script}}
```

Allgemeine Einschränkung. Die folgende Einschränkung gilt für verschiedene Elemente:

- Verweise auf externe URLs werden nicht aufgelöst (Rasterbild, Font usw.)

Nicht unterstützte SVG-Elemente. Die folgenden SVG-Elemente werden nicht unterstützt und einfach ignoriert:

- Elemente für Animation und Scripting:

  `animate, animateColor, animateMotion, animateTransform, script, mpath, set`

- Elemente für SVG-Filter:

  `feBlend, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feDistantLight, feFlood, feFuncA, feFuncB, feFuncG, feFuncR, feGaussianBlur, feImage, feMerge, feMergeNode, feMorphology, feOffset, fePointLight, feSpecularLighting, feSpotLight, feTile, feTurbulence, filter`

- Elemente für die Auswahl von Glyphen:

  `altGlyph, altGlyphDef, altGlyphItem, glyphRef`

- weitere Elemente:

  `cursor, foreignObject, vkern`

Einschränkungen bei SVG-Attributen und -Properties. Für die folgenden Attribute und Properties gelten bestimmte Einschränkungen:

- Manche CSS-Regeln werden nicht unterstützt, einschließlich *@import* und *@font-face*.
- Die Property zur Fontauswahl *font-variant* wird nur zusammen mit dem Schlüsselwort *small-caps* und nur für Fonts mit der OpenType-Funktion *smcp* unterstützt.
- Die Kombination von Werten der Property *text-decoration* wird nicht unterstützt. PDFlib zeichnet die Dekorationselemente nicht als Bereiche mit separater Füll- und Linienfarbe, sondern als Linien. Diese werden mit der Füllfarbe gezeichnet, wenn vorhanden, und sonst mit der Linienfarbe.

- Das Attribut *rotate* für das Element *textPath* wird nicht unterstützt.
- Die Property *unicode-bidi* wird nur ausgewertet bei Fonts vom Typ TrueType oder OpenType mit den erforderlichen Tabellen für bidirektionales Textlayout. PDFlib setzt die Optionen *shaping* und *script=_auto* in der Optionsliste von *PDF_fit_textline( )*.
- Das Attribut *preserveAspectRatio* für das Element *view* wird ignoriert.

Nicht unterstützte SVG-Properties. Die folgenden SVG-Properties werden nicht unterstützt und einfach ignoriert:

```
alignment-baseline, color-interpolation, color-interpolation-filters, color-profile,
color-rendering, cursor, dominant-baseline, enable-background, filter, flood-color,
flood-opacity, font, glyph-orientation-horizontal, glyph-orientation-vertical,
image-rendering, lighting-color, pointer-events, shape-rendering, text-rendering
```

Nicht unterstützte Attribute unterstützter SVG-Elemente. Die folgenden Attribute unterstützter SVG-Elemente werden nicht unterstützt und einfach ignoriert:

```
baseProfile (svg)
contentScriptType (svg)
contentStyleType (svg)
externalResourcesRequired (all elements)
method (textPath)
on* (all elements)
requiredExtensions (all elements)
requiredFeatures (all elements)
spacing (textPath)
spreadMethod (linearGradient, radialGradient)
version (svg)
zoomAndPan (svg)
xlink:role (all elements)
xlink:show (all elements)
xlink:type (all elements)
```

7.3 Import von PDF-Seiten mit PDI

Hinweis *Alle in diesem Abschnitt beschriebenen Funktionen setzen PDFlib+PDI oder den PDFlib Personalization Server PPS (der PDI enthält) voraus. Die PDF-Importbibliothek (PDI) ist nicht im Basisprodukt PDFlib enthalten. PDI ist zwar in allen binären PDFlib-Editionen integriert, es ist aber ein eigener Lizenzschlüssel für PDFlib+PDI oder für PPS erforderlich.*

7.3.1 PDI-Funktionen und -Anwendungen

Mit PDI (PDF-Importbibliothek) können Seiten aus vorhandenen PDF-Dokumenten importiert werden. PDI bereitet PDF-Seiten für einen einfachen Einsatz mit PDFlib auf. Der Umgang mit importierten PDF-Seiten unterscheidet sich vom Konzept her kaum von importierten Rasterbildern: Sie öffnen ein PDF-Dokument, wählen die zu importierende Seite und platzieren sie auf einer Ausgabeseite. Dabei können Sie sie mit PDFlib-Transformationsfunktionen zum Verschieben, Skalieren, Drehen oder Scheren bearbeiten. Importierte Seiten können anhand von PDFlib-Text- und Grafikfunktionen mit neuem Inhalt kombiniert werden, nachdem sie auf der Ausgabeseite platziert wurden (man kann sich die importierte Seite als Hintergrund für neuen Inhalt vorstellen). Mit PDFlib+PDI lassen sich problemlos folgende Aufgaben erledigen:

- zwei oder mehr Seiten aus verschiedenen PDF-Dokumenten überlagern (zum Beispiel Briefpapier mit variablen Inhalten kombinieren, um vorgedrucktes Papier zu simulieren)
- PDF-Kleinanzeigen in vorhandenen Dokumenten platzieren
- den sichtbaren Bereich einer PDF-Seite beschneiden, um unerwünschte Elemente (zum Beispiel Schnittmarken) zu verbergen oder Seiten zu skalieren
- mehrere Seiten zum Druck auf einem Bogen montieren
- aus mehreren PDF/X- oder PDF/A-kompatiblen Dokumenten eine neue PDF/X- oder PDF/A-Datei erzeugen
- PDF/X- oder PDF/A-Druckausgabebedingung einer Datei kopieren
- Text (zum Beispiel Kopfzeilen, Fußzeilen, Stempel oder Seitenzahlen) oder Bilder (zum Beispiel ein Firmenlogo) zu vorhandenen PDF-Seiten hinzufügen
- alle Seiten von einem Eingabedokument in ein Ausgabedokument kopieren und Barcodes auf den Seiten platzieren
- mit der pCOS-Schnittstelle beliebige Eigenschaften eines PDF-Dokuments abfragen (für weitere Informationen siehe die pCOS-Pfadreferenz).

Um eine PDF-Hintergrundseite zu platzieren und mit dynamisch generierten Daten zu füllen (zum Beispiel Serienbriefe, personalisierte PDF-Dokumente im Web oder Ausfüllen von Formularen), empfehlen wir den Einsatz von PDI in Kombination mit PDFlib-Blöcken (siehe Kapitel 12, »PPS und das PDFlib Block-Plugin«, Seite 363).

7.3.2 Einsatz von PDFlib+PDI

Cookbook *Codebeispiele zum PDF-Import finden Sie in der Kategorie* `pdf_import` *des PDFlib Cookbook.*

Allgemeine Hinweise. Sie müssen unbedingt beachten, dass PDI nur den eigentlichen Seiteninhalt importiert und keinerlei gegebenenfalls im PDF-Dokument vorhandene interaktive Elemente (wie zum Beispiel Sound, Movies, eingebettete Dateien, Hypertext-Verknüpfungen, Formularfelder, JavaScript, Lesezeichen, Miniaturseiten oder Noti-

zen). Solche interaktiven Funktionen lassen sich mit den entsprechenden PDFlib-Funktionen generieren.

Die folgenden Elemente können Sie optional importieren:

- Strukturelement-Tags (siehe Abschnitt »Importieren von PDF-Seiten aus Tagged PDF«, Seite 205)
- Ebenen-Definitionen (siehe Abschnitt »Importieren von PDF-Seiten mit Ebenen (Layer)«, Seite 205)
- (nur bei PPS) Importieren von PDFlib-Blöcke mittels *PDF_process_pdi( )* und der Option *action=copyallblocks* oder *copyblock* (siehe Abschnitt 12.8.2, »Importieren von PDFlib-Blöcken«, Seite 405).

Sie können einzelne Elemente der importierten Seite nicht in PDFlib-Funktionen weiterverwenden. So lassen sich Fonts aus importierten Dokumenten nicht für andere Inhalte benutzen, da alle benötigten Fonts in PDFlib konfiguriert sein müssen. Enthalten mehrere importierte Dokumente eingebettete Fontdaten desselben Fonts, so werden die mehrfach vorhandenen Fonts von PDI nicht entfernt. Fehlen die Fonts dagegen in einem importierten PDF, dann fehlen sie auch in der generierten PDF-Ausgabe. Zur Optimierung sollten Sie das importierte Dokument so lange offen halten, wie Sie noch Seiten daraus benötigen, damit dieselben Fonts nicht mehrmals im Ausgabedokument eingebettet werden.

PDFlib+PDI platziert importierte PDF-Seiten auf der Ausgabeseite mittels der Template-Funktion (Form XObjects). Dokumente, die importierte Seiten aus anderen PDF-Dokumenten enthalten, können auch ein weiteres Mal mit PDFlib+PDI bearbeitet werden.

Codefragment zum Importieren von PDF-Seiten. Der Umgang mit Seiten aus vorhandenen PDF-Dokumenten ist mit sehr einfach strukturiertem Code möglich. Das folgende Codefragment öffnet eine vorhandene Dokumentseite und kopiert den Seiteninhalt auf eine neue Seite des PDF-Ausgabedokuments (das vorher geöffnet werden muss):

```
int     doc, page, pageno = 1;
String  filename = "input.pdf";

if (p.begin_document(outfilename, "") == -1) {...}
...

doc = p.open_pdi_document(infilename, "");
if (doc == -1)
        throw new Exception("Error: " + p.get_errmsg());

page = p.open_pdi_page(doc, pageno, "");
if (page == -1)
        throw new Exception("Error: " + p.get_errmsg());

/* Dummy-Seitengröße, wird mit der Option adjustpage angepasst */
p.begin_page_ext(20, 20, "");
p.fit_pdi_page(page, 0, 0, "adjustpage");
p.close_pdi_page(page);
...weitere Seiteninhalte mit PDFlib-Funktionen hinzufügen...
p.end_page_ext("");
p.close_pdi_document(doc);
```

PDF_fit_pdi_page( ) erhält als letztes Argument eine Optionsliste, die zahlreiche Optionen zur Positionierung, Skalierung und Drehung der importierten Seite unterstützt. Weitere Informationen hierzu finden Sie in Abschnitt 7.4, »Platzieren von Bildern, Grafiken und importierten PDF-Seiten«, Seite 207.

7.3.3 Dokument- und seitenbezogene Prüfungen

Dokumentbezogene Prüfungen. PDI verarbeitet in der Regel problemlos alle Arten von PDF-Dokumenten, die sich auch in Acrobat öffnen lassen, unabhängig von der PDF-Versionsnummer oder den im Dokument verwendeten Funktionen.

In PDFlib+PDI ist ein Reparaturmodus für beschädigte PDF-Dateien implementiert, so dass sich selbst manche beschädigten Dokumente öffnen lassen. Nur in seltenen Fällen wird ein PDF-Dokument oder eine bestimmte Seite eines Dokuments von PDI zurückgewiesen.

Wenn ein PDF-Dokument oder eine PDF-Seite nicht erfolgreich importiert werden kann, geben *PDF_open_pdi_document( )* und *PDF_open_pdi_page( )* einen Fehlercode zurück. Mit *PDF_get_errmsg( )* können Sie genauere Informationen zur Fehlerursache abfragen. Alternativ können Sie die Option *errorpolicy* auf *exception* setzen. Dies führt zu einer Exception, wenn das Dokument nicht geöffnet werden kann.

Seitenbezogenene Prüfungen. Die folgenden Prüfungen werden bei *PDF_open_pdi_page( )* durchgeführt:

- Seiten aus PDF-Dokumenten mit einer höheren PDF-Versionsnummer als die aktuell generierte PDF-Ausgabe können nicht importiert werden. Denn PDFlib könnte danach nicht mehr sicherstellen, dass die Ausgabe auch tatsächlich der geforderten PDF-Version entspricht. Die Lösung besteht darin, die Version der PDF-Ausgabe mit der Option *compatibility* von *PDF_begin_document( )* an den erforderlichen Level entsprechend anzupassen.
  Dokumente für *PDF 1.7ext 3* (Acrobat 9) und *PDF 1.7ext8* (Acrobat X/XI) sind aus Sicht von PDI kompatibel PDF 1.7.
  Im PDF/A-Modus ist die PDF-Versionsnummer der Eingabe nicht relevant, da der Header mit der PDF-Version bei PDF/A ignoriert werden muss.
  Wenn ein Dokument einen Header mit einer höheren PDF-Version trägt, obwohl es bekanntermaßen zu einer älteren PDF-Version konform geht, können Sie die Option *ignorepdfversion* von *PDF_open_pdi_document( )* verwenden.
- PDF-Dokumente gemäß PDF/A, PDF/X, PDF/VT oder PDF/UA, die nicht zum entsprechenden Level von PDF/A, PDF/X, PDF/VT oder PDF/UA des aktuellen Ausgabedokuments passt. Für weitere Informationen siehe die folgenden Abschnitte:
  - Abschnitt 11.3.7, »Import von PDF/A-Dokumenten mit PDI«, Seite 332;
  - Abschnitt 11.4.5, »Import von PDF/X-Dokumenten mit PDI«, Seite 342;
  - Abschnitt 11.5.7, »Import von PDF/X- und PDF/VT-Dokumenten mit PDI«, Seite 353;
  - Abschnitt 11.6.3, »Zusätzliche Anforderungen für bestimmte Inhaltstypen«, Seite 360.
- Enthält das Dokument eine inkonsistente Druckausgabebedingung für PDF/A oder PDF/X, können keine Seiten importiert werden.

7.3.4 Besonderheiten bei importierten PDF-Dokumenten

Abmessungen importierter PDF-Seiten. Importierte PDF-Seiten werden im Prinzip wie importierte Rasterbilder behandelt und können mit *PDF_fit_pdi_page( )* auf der Ausgabeseite platziert werden. PDI importiert die Seite standardmäßig genau so, wie sie in Acrobat angezeigt wird:

- Eine eventuelle Beschneidung der Seite bleibt erhalten (technisch ausgedrückt: Ist eine Größenangabe für die CropBox vorhanden, dann zieht PDI diese der MediaBox vor; siehe Abschnitt 3.2.2, »Seitengröße«, Seite 76).
- Eine eventuell auf die Seite angewandte Drehung bleibt erhalten.

Mit der Option *cloneboxes* kann PDFlib+PDI alle Seiten-Boxen einer importierten Seite auf die erzeugte Ausgabeseite kopieren. Im Ergebnis werden alle Aspekte der Seitengröße geklont.

Alternativ dazu können Sie PDI mit der Option *pdiusebox* explizit anweisen, zur Ermittlung der Größe der importierten Seite eine der Angaben MediaBox, CropBox, BleedBox, TrimBox oder ArtBox zu verwenden, falls vorhanden.

Farbbehandlung. PDFlib+PDI ändert die Farbe der importierten PDF-Dokumente in keiner Weise. Bei einem PDF mit ICC-Farbprofilen bleiben diese zum Beispiel im Ausgabedokument erhalten.

Importieren von PDF-Seiten aus Tagged PDF. Tags werden automatisch importiert, wenn sowohl Eingabe- als auch Ausgabedokument getagged sind. Der Import von Tags lässt sich jedoch mit der Option *usetags* von *PDF_open_pdi_document( )* und *PDF_open_pdi_page( )* deaktivieren. Für weitere Informationen siehe Abschnitt 10.4.5, »Import von Tagged PDF mit PDI«, Seite 310.

Importieren von PDF-Seiten mit Ebenen (Layer). PDI importiert immer die Inhalte aller Ebenen auf einer Seite (der technische Begriff ist *optional content)*. Auch die Ebenen-Definitionen einschließlich des Sichtbarkeitsstatus der Ebenen werden importiert, sofern die Ebene auf einer der importierten Seiten verwendet wird. Dabei kann der Import der Ebenen-Definitionen mit den Optionen *uselayers* von *PDF_open_pdi_document( )* deaktiviert werden. Sie können über die Anordnung der importierten Ebenen mit der Option *parenttitle* von *PDF_open_pdi_document( )* genauer steuern. Diese Option erzeugt eine hierarchische Titel-Ebene in der Ebenen-Liste oberhalb der importierten Ebenen (z.B. zur Übergabe eines Dateinamens). Die Option *parentlayer* funktioniert ähnlich, erwartet allerdings ein Handle für eine benutzerdefinierte Ebene.

Hinweis *Ebenen-Varianten gemäß PDF/X-4:2008 werden nicht importiert, da diese ab Acrobat X nicht mehr unterstützt werden.*

Importieren von georeferenziertem PDF. Beim Import von georeferenziertem PDF mit PDI bleiben die Geodaten erhalten, wenn sie mit einer der folgenden Methoden erzeugt wurden (bildbezogene Georeferenz):

- mit PDFlib und der Option *georeference* von *PDF_load_image( )*
- durch Import eines Bildes mit Geodaten in Acrobat

Die Geodaten gehen nach dem Import der Seite verloren, wenn sie auf eine der folgenden Arten erzeugt wurden (seitenbezogene Geo-Referenz):

- mit PDFlib und der Option *viewports* von *PDF_begin/end_page_ext( )*
- durch manuelle Geo-Registrierung einer PDF-Seite in Acrobat

Optimierung über mehrere importierte Dokumente. Während PDFlib höchstgradig optimierte PDF-Ausgabe erzeugt, enthalten importierte PDF-Dokumente nicht selten redundante Datenstrukturen, die sich noch optimieren lassen. Gehen mehrere importierte PDF-Dokumente in die Ausgabe ein, so wird diese durch identische Ressourcen, etwa Fonts, unter Umständen unnötig aufgebläht. Mit der Option *optimize* in *PDF_begin_document( )* lässt sich die Ausgabe in solchen Fällen verkleinern, indem redundante Objekte ermittelt und ohne Auswirkungen auf das Erscheinungsbild oder die Qualität der generierten Ausgabe entfernt werden.

Verschlüsseltes PDF und das Feature »shrug«. Um Seiten aus verschlüsselten Dokumenten (die Berechtigungseinstellungen oder ein Kennwort verwenden) zu importieren, muss das zugehörige Master-Kennwort übergeben werden.Verschlüsselte PDF-Dokumente werden ohne das Master-Kennwort zurückgewiesen. Sie können jedoch zur Abfrage von Informationen mit pCOS geöffnet werden (im Gegensatz zum Import von Seiten), indem die Option *infomode* von *PDF_open_pdi_document( )* auf *true* gesetzt wird. Ausnahme zur *infomode*-Regel: Dokumente, bei deren Erstellung die Distiller-Einstellung *Komprimierung auf Objektebene* auf *Maximum* gesetzt wurde; diese lassen sich noch nicht einmal im Infomodus öffnen.

Mit dem Feature *shrug* lassen sich Seiten aus geschützten Dokumenten ohne das Master-Kennwort importieren, vorausgesetzt der Benutzer übernimmt die Verantwortung für die Einhaltung der Urheberrechte des Dokuments. Mit dem Feature *shrug* versichert der Benutzer, dass er oder sie nicht gegen die Urheberrechte des Dokuments verstößt. Die Allgemeinen Geschäftsbedingungen von PDFlib GmbH verlangen dies.

Wenn alle folgenden Voraussetzungen erfüllt sind, wird *shrug* aktiviert:
- Die Option *shrug* wurde an *PDF_open_pdi_document( )* übergeben.
- Das Dokument verlangt ein Master-Kennwort, aber dieses wurde nicht an *PDF_open_pdi_document( )* übergeben.
- Verlangt das Dokument ein Benutzer-Kennwort zum Öffnen, muss dieses an *PDF_open_pdi_document( )* übergeben worden sein.

shrug hat folgende Wirkung:
- Seiten können importiert werden, auch wenn das Master-Kennwort nicht übergeben wurde.
- Das pCOS-Pseudo-Objekt *shrug* wird auf *true*/1 gesetzt.
- pCOS läuft im Vollmodus (statt im eingeschränkten Modus), das heißt, das Pseudo-Objekt *pcosmode* wird auf 2 gesetzt.

7.4 Platzieren von Bildern, Grafiken und importierten PDF-Seiten

Die Funktionen *PDF_fit_image( )* zur Platzierung von Rasterbildern und Templates, *PDF_fit_graphics( )* zur Platzierung von Grafiken sowie *PDF_fit_pdi_page( )* zur Platzierung importierter PDF-Seiten bieten eine Fülle von Optionen zur Platzierung auf einer Seite. Diese werden im Folgenden anhand einiger häufig vorkommenden Anwendungsfälle erläutert. Eine vollständige Auflistung aller Optionen finden Sie in der PDFlib-Referenz.

Alle in diesem Abschnitt erläuterten Beispiele gelten gleichermaßen für Rasterbilder, Templates, Grafiken und importierte PDF-Seiten. Die Codebeispiele beziehen sich zwar ausschließlich auf Rasterbilder, die Beschreibung gilt aber allgemein für die Platzierung von Objekten. Beachten Sie, dass vor dem Aufruf einer *fit*-Funktion eine der Funktionen *PDF_load_image( ), PDF_load_graphics( )* oder *PDF_open_pdi_document( )* und *PDF_open_pdi_page( )* ausgeführt werden muss. Der Einfachheit halber werden diese Aufrufe hier nicht wiederholt.

Cookbook Codebeispiele zu Rasterbildern, Grafiken und importierten PDF-Seiten finden Sie in den Kategorien `images`, `graphics` *und* `pdf_import` *des PDFlib Cookbook.*

7.4.1 Einfache Platzierung von Objekten

Positionierung eines Bildes am Referenzpunkt. Ein Objekt wird standardmäßig in Originalgröße mit der linken unteren Ecke am Referenzpunkt platziert. In diesem Beispiel platzieren wir ein Bild unten mittig am Referenzpunkt *(0, 0)*:

```
p.fit_image(image, 0, 0, "position={center bottom}");
```

In ähnlicher Weise können Sie die Option *position* mit einer anderen Kombination der Schlüsselwörter *left, right, center, top* und *bottom* nutzen, um das Objekt am Referenzpunkt zu platzieren.

Platzierung eines Bildes mit Skalierung. Folgende Variante positioniert das Bild in veränderter Größe:

```
p.fit_image(image, 0, 0, "scale=0.5");
```

Dieses Codefragment platziert das Objekt mit der linken unteren Ecke am Punkt *(0, 0)* des Benutzerkoordinatensystems. Das Objekt wird außerdem in *x*- und *y*-Richtung um den Faktor 0,5 skaliert, das heißt auf 50% verkleinert.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `images/starter_image`.

7.4.2 Positionieren eines Objekts an einem Punkt, einer Linie oder in einer Box

Zur Positionierung eines Objekts kann eine zusätzliche Box von definierter Höhe und Breite verwendet werden. Beachten Sie, dass die graue Box oder Linie in Abbildung 7.2 nur zur Veranschaulichung der Boxmaße dient und nicht Bestandteil der tatsächlichen Ausgabe ist.

Die Platzierung eines Objekts in einer Box macht bei *fitmethod=nofit* keinen Sinn, da das Objekt in diesem Fall nur positioniert, aber nicht skaliert wird. Mit der Option *boxsize* kann eine horizontale Linie, vertikale Linie oder echte Box zur Platzierung eines Objekts angegeben werden:

```
boxsize={100 0}          horizontale Linie
boxsize={0 100}          vertikale Linie
boxsize={100 200}        Box
```

In den Beispielen unten wird das Objekt anhand verschiedener Methoden in eine Box eingepasst.

Einpassen eines Bildes in eine Box. Mit *fitmethod=auto* skaliert PDFlib das Bild so, dass es ohne Verzerrung in die Box passt: Wenn es in die Box passt, wird es nicht skaliert. Andernfalls wird es proportional verkleinert. Abbildung 7.2a, Abbildung 7.2b und Abbildung 7.2c zeigen, wie PDFlib die Größe des Bildes an die Größe der Fitbox anpasst, die sich von ursprünglich *boxsize={70 45}* auf *boxsize={70 30}* und weiter auf *boxsize={30 30}* verkleinert.

Abb. 7.2 Einpassen eines Bildes in eine Box anhand verschiedener Methoden

Generierte Ausgabe	*Optionsliste für PDF_fit_image*
a) 	`boxsize={70 45} position=center fitmethod=auto` `(keine Skalierung erforderlich)`
b) 	`boxsize={70 30} position=center fitmethod=auto` `(auf geringere Bildhöhe der Box passend verkleinert)`
c) 	`boxsize={30 30} position=center fitmethod=auto` `(auf geringere Bildhöhe und -breite der Box passend verkleinert)`
d) 	`boxsize={70 45} position=center fitmethod=meet`
e) 	`boxsize={35 45} position=center fitmethod=meet`

Generierte Ausgabe	*Optionsliste für PDF_fit_image*
f) 	`boxsize={70 45} position=center fitmethod=entire`
g) 	`boxsize={30 30} position=center fitmethod=clip`
h) 	`boxsize={30 30} position={right top} fitmethod=clip`

Einpassen eines Bildes in die Boxmitte. Um ein Bild in einem vordefinierten Rechteck zu zentrieren, brauchen Sie keine Berechnungen durchzuführen, sondern müssen nur die passenden Optionen verwenden. Mit *position=center* platzieren wir das Bild in der Mitte der Box mit einer Breite von 70 Einheiten und einer Höhe von 45 Einheiten *(boxsize={70 45})*. Mit *fitmethod=meet* wird die Bildgröße proportional angepasst, so dass die Bildhöhe die Box vollständig ausfüllt (siehe Abbildung 7.2d).

Wenn wir die Boxbreite von 70 auf 35 Einheiten reduzieren, wird das Bild so weit verkleinert, dass die Bildbreite vollständig in die Box passt (siehe Abbildung 7.2e).

Mit *fitmethod=meet* kann dafür gesorgt werden, dass das Bild nicht verzerrt und so groß wie möglich in der Box platziert wird.

Vollständiges Einpassen des Bildes in eine Box. Mit der Option *fitmethod=entire* können wir das Bild so einpassen, dass es die Box vollständig ausfüllt. Diese Methode ist aber nur selten hilfreich, da sie das Bild in der Regel verzerrt (siehe Abbildung 7.2f).

Beschneiden eines Bildes beim Einpassen in die Box. Mit *fitmethod=clip* können wir das Objekt beschneiden, wenn es über die Box hinausreicht. Wir reduzieren die Box auf eine Breite und Höhe von 30 Einheiten und positionieren das Bild in Originalgröße in der Boxmitte (siehe Abbildung 7.2g).

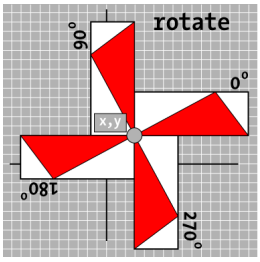
Durch Positionierung des Bildes in der Boxmitte wird es gleichmäßig an allen Seiten beschnitten. Genauso könnten Sie es, um den rechten oberen Teil des Bildes vollständig zu zeigen, mit *position={right top}* platzieren (siehe Abbildung 7.2h).

7.4.3 Orientierung eines Objekts

Platzierung mit Orientierung. Im nächsten Codefragment richten wir ein Bild nach Westen aus *(orientate=west)*. Dabei wird das Bild um 90° gegen den Uhrzeigersinn gedreht und die linke untere Ecke des gedrehten Bildes auf den Referenzpunkt *(0, 0)* verschoben. Das Objekt wird um seinen Mittelpunkt gedreht (siehe Abbildung 7.5a). Da wir keine spezielle Platzierungsmethode verwenden, wird das Bild in Originalgröße ausgegeben und ragt aus der Box hinaus.

Proportionales Einpassen eines Bildes mit Orientierung. Unser nächstes Ziel besteht darin, das Bild in vorgegebener Größe nach Westen auszurichten. Wir definieren eine Box der gewünschten Größe und platzieren das Bild, ohne seine Proportionen zu verändern *(fitmethod=meet)*. Die Orientierung wird mit *orientate=west* festgelegt. Standard-

Abb. 7.3
Option rotate



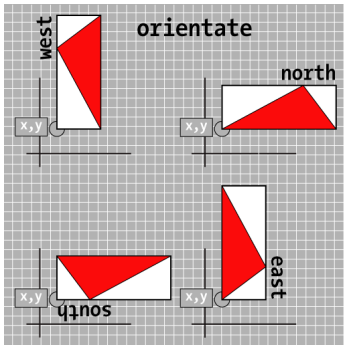


Abb. 7.4
Option orientate

mäßig wird das Bild in der linken unteren Ecke der Box platziert (siehe Abbildung 7.5b). Abbildung 7.5c zeigt das Bild mit einer Orientierung nach Osten, Abbildung 7.5d mit einer Orientierung nach Süden.

Wie Abbildung 7.4 veranschaulicht, unterstützt die Option *orientate* die Schlüsselwörter *north, east, west* und *south*.

Beachten Sie, dass die Option *orientate* keinen Einfluss auf das Koordinatensystem im Ganzen, sondern nur auf ein einzelnes Objekt besitzt.

Abb. 7.5 Orientierung eines Bildes

Generierte Ausgabe	*Optionsliste für PDF_fit_image()*
a) 	`boxsize {70 45} orientate=west`[1]
b) 	`boxsize {70 45} orientate=west fitmethod=meet`
c) 	`boxsize {70 45} orientate=east fitmethod=meet`
d) 	`boxsize {70 45} orientate=south fitmethod=meet`
e) 	`boxsize {70 45} position={center bottom} orientate=east fitmethod=clip`

1. Die Option `boxsize` *ist eigentlich wegen der Voreinstellung* `fitmethod=nofit` *nicht erforderlich.*

Einpassen eines Bildes in eine Box mit Beschneiden. Wir orientieren das Bild nach Osten *(orientate=east)* und positionieren es in der Boxmitte unten *(position={center bottom})*. Außerdem platzieren wir das Bild in seiner Originalgröße und beschneiden es, wenn es aus der Box hinausragt *(fitmethod=clip)* (siehe Abbildung 7.5e).

7.4.4 Drehen eines Objekts

Mit der Option *rotate* lässt sich ein Objekt durch Drehung des Koordinatensystems im Referenzpunkt drehen. Die Box wird dadurch ebenfalls gedreht. Abbildung 7.3 zeigt das allgemeine Verhalten der Option *rotate*.

Platzieren eines Bildes mit Drehung. Unser erstes Ziel besteht darin, ein Bild um 90° gegen den Uhrzeigersinn zu drehen. Vor der Platzierung des Objekts wird das Koordinatensystem im Referenzpunkt um 90° gegen den Uhrzeigersinn gedreht. Im Referenzpunkt liegt dann die rechte untere Ecke des gedrehten (also die linke untere Ecke des nicht gedrehten) Objekts. Abbildung 7.6a zeigt diesen Fall.

Da sich die Drehung auf das gesamte Koordinatensystem bezieht, wird die Box ebenfalls gedreht. Genauso können wir das Bild um 30° gegen den Uhrzeigersinn drehen (siehe Abbildung 7.6b).

Einpassen eines Bildes mit Drehung. Unser nächstes Ziel besteht darin, das um 90° gegen den Uhrzeigersinn gedrehte Bild unter Beibehaltung seiner Proportionen in die Box einzupassen. Dies wird mit *fitmethod=meet* bewerkstelligt (siehe Abbildung 7.6c). Dementsprechend können wir das Bild auch um 30° gegen den Uhrzeigersinn drehen und proportional in die Box einpassen (siehe Abbildung 7.6d).

Abb. 7.6 Drehen eines Bildes

Generierte Ausgabe	*Optionsliste für PDF_fit_image( )*
a) 	`boxsize {70 45} rotate=90`[1]
b) 	`boxsize {70 45} rotate=30`[1]
c) 	`boxsize {70 45} rotate=90 fitmethod=meet`
d) 	`boxsize {70 45} rotate=30 fitmethod=meet`

1. Die Option `boxsize` *ist eigentlich wegen der Voreinstellung* `fitmethod=nofit` *nicht erforderlich.*

7.4.5 Anpassen der Seitengröße

Anpassen der Seitengröße an ein Bild. Im nächsten Beispiel passen wir die Seitengröße automatisch an die Bildgröße an. Dies kann bei der Archivierung von Bildern im PDF-Format nützlich sein. Der Referenzpunkt *(x, y)* wird verwendet, um festzulegen, ob die Seite exakt so groß wie das Objekt oder etwas größer oder kleiner angelegt wird. Bei einer Vergrößerung der Seite (siehe Abbildung 7.7) erhalten wir einen Rand um das Bild. Ist die Seite kleiner als das Bild, reicht das Bild über den Seitenrand hinaus, das heißt auf jeder Seite wird ein Teil des Bildes abgeschnitten. Beginnen wir mit der exakten Größenanpassung:

```
p.fit_image(image, 0, 0, "adjustpage");
```

Das folgende Codefragment legt die Seite in *x*- und *y*-Richtung um 40 Punkt größer an als das Objekt. Man erhält eine Umrandung um das Objekt:

```
p.fit_image(image, 40, 40, "adjustpage");
```

Das folgende Codefragment legt die Seite in *x*- und *y*-Richtung um 40 Punkt kleiner an als das Objekt. Das Objekt wird an den Seitenrändern beschnitten. Innerhalb des Bildes bleibt so ein Rand von 40 Punkt Breite unsichtbar:

```
p.fit_image(image, -40, -40, "adjustpage");
```

Neben der Platzierung mit Hilfe von *x*- und *y*-Koordinaten, die den Abstand des Objekts vom Seitenrand oder im allgemeinen Fall die Koordinatenachsen definieren, kann zusätzlich eine Zielbox festgelegt werden. Dies ist ein rechteckiger Bereich, in den sich das Objekt anhand verschiedener Formatierungsarten einpassen lässt. Dazu stehen zusätzlich die Optionen *boxsize*, *fitmethod* und *position* zur Verfügung.

Abb. 7.7 Anpassen der Seitengröße. Von links nach rechts: exakt, vergrößert, verkleinert.

Klonen von Seitenboxen einer importierten PDF-Seite. Sie können alle relevanten Seitenboxen (MediaBox, CropBox usw.) einer importierten PDF-Seite auf die aktuelle Ausgabeseite kopieren. Dazu müssen Sie die Option *cloneboxes* zuerst an *PDF_open_pdi_page( )* übergeben, um alle relevanten Werte für die Box auszulesen, und dann an *PDF_fit_pdi_page( ),* um die Werte der Box auf die aktuelle Seite anzuwenden:

```
/* Öffnen der Seite und Klonen der Seitenangaben für die Box */
inpage = p.open_pdi_page(indoc, 1, "cloneboxes");
...
/* Starten der Ausgabeseite mit einer Dummy-Seitengröße */
p.begin_page_ext(10, 10, "");
...
/*
* Platzieren der importierten Seite auf der Ausgabeseite und Klonen aller
* Seitenboxen auf der aktuellen Eingabeseite; dies überschreibt die Dummy-
* Größe aus begin_page_ext().
```

```
*/
p.fit_pdi_page(inpage, 0, 0, "cloneboxes");
```

Auf diese Art können Sie sicherstellen, dass die Seiten des generierten PDF genau die gleiche Größe, Beschneidung usw. wie die Seiten des importierten Dokuments haben. Dies ist besonders wichtig für die Druckvorstufe.

7.4.6 Abfrage von Informationen über platzierte Bilder und PDF-Seiten

Informationen über platzierte Bilder und Templates. Mit der Funktion *PDF_info_image( )* lassen sich Informationen über Bilder und Templates abfragen. Mit den unterstützten Schlüsselwörtern lassen sich allgemeine Bilddaten wie Breite und Höhe in Pixeln sowie geometrische Informationen über die Anordnung des Bildes auf der Ausgabeseite abfragen (zum Beispiel Breite und Höhe in absoluten Werten nach der Einpassung).

Das folgende Codefragment ruft sowohl die Pixelgröße als auch die absolute Größe nach dem Platzieren des Bildes mit bestimmten Einpassoptionen ab:

```
String optlist = "boxsize={300 400} fitmethod=meet orientate=west";
p.fit_image(image, 0.0, 0.0, optlist);

imagewidth  = (int) p.info_image(image, "imagewidth", optlist);
imageheight = (int) p.info_image(image, "imageheight", optlist);
System.err.println("image size in pixels: " + imagewidth + " x " + imageheight);

width  = p.info_image(image, "width", optlist);
height = p.info_image(image, "height", optlist);
System.err.println("image size in points: " + width + " x " + height);
```

Informationen über platzierte PDF-Seiten. Mit der Funktion *PDF_info_pdi_page( )* lassen sich Informationen über platzierte PDF-Seiten abfragen. Mit den unterstützten Schlüsselwörtern lassen sich Informationen über die Originalseite wie Breite und Höhe sowie geometrische Informationen über die Anordnung des importierten PDF auf der Ausgabeseite abfragen (zum Beispiel Breite und Höhe nach der Einpassung).

Das folgende Codefragment ruft sowohl die Originalgröße der importierten Seite als auch die Größe nach dem Platzieren der Seite mit bestimmten Einpassoptionen ab:

```
String optlist = "boxsize={400 500} fitmethod=meet";
p.fit_pdi_page(page, 0, 0, optlist);

pagewidth  = p.info_pdi_page(page, "pagewidth", optlist);
pageheight = p.info_pdi_page(page, "pageheight", optlist);
System.err.println("original page size: " + pagewidth + " x " + pageheight);

width  = p.info_pdi_page(page, "width", optlist);
height = p.info_pdi_page(page, "height", optlist);
System.err.println("size of placed page: " + width + " x " + height);
```

8 Text- und Tabellenformatierung

8.1 Platzieren und Einpassen von einzeiligem Text

Die Funktion *PDF_fit_textline( )* zum Platzieren einer Textzeile auf der Seite bietet eine Fülle von Optionen. Diese werden im Folgenden anhand einiger häufig vorkommender Anwendungsbeispiele erläutert. Eine vollständige Auflistung aller Optionen finden Sie in der PDFlib-Referenz. Viele der zu *PDF_fit_textline( )* gehörenden Optionen sind mit denen für *PDF_fit_image( )* identisch. Wir werden hier nur textbezogene Beispiele anführen. Sie sollten sich deshalb – vielleicht auch zur Einführung in die Bildformatierung – die Beispiele in Abschnitt 7.4, »Platzieren von Bildern, Grafiken und importierten PDF-Seiten«, Seite 207 ansehen.

Die folgenden Beispiele zeigen nur den jeweiligen Aufruf der Funktion *PDF_fit_textline( )*. Es wird vorausgesetzt, dass der gewünschte Font bereits geladen und in der gewünschten Größe gesetzt wurde.

Um die Platzierung des Textes zu berechnen, verwendet *PDF_fit_textline( )* die so genannte Textbox: Die Breite dieser Textbox entspricht der Länge des Schriftzugs und ihre Höhe der Höhe eines Großbuchstabens des Fonts. Mit der Option *matchbox* lässt sich die Textbox anpassen.

In den folgenden Beispielen werden die Koordinaten des Referenzpunkts als Parameter *x, y* an *PDF_fit_textline( )* übergeben. Die Fitbox ist bei Textzeilen der Bereich, in dem der Text platziert wird. Die Fitbox entspricht dem rechteckigen Bereich, der durch die Parameter *x, y* von *PDF_fit_textline( )* und passende Optionen *(boxsize, position, rotate)* festgelegt ist. Mit der Option *margin* kann die Fitbox nach links und rechts oder oben und unten verkleinert werden.

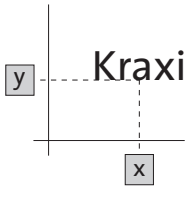
Cookbook *Codebeispiele zu Textausgabe finden Sie in der Kategorie* `text_output` *des PDFlib Cookbook.*

8.1.1 Einfaches Platzieren von Textzeilen

Platzieren von Text am Referenzpunkt. Text wird standardmäßig mit der linken unteren Ecke am Referenzpunkt platziert. In diesem Beispiel wollen wir den Text jedoch so am Referenzpunkt positionieren, dass die Textbox in der Mitte ihres unteren Randes auf dem Punkt zu liegen kommt. Das folgende Codefragment platziert die Textbox in der Mitte des unteren Randes am Referenzpunkt *(30, 20)*.

```
p.fit_textline(text, 30, 20, "position={center bottom}");
```

Abb. 8.1
Zentrierter Text



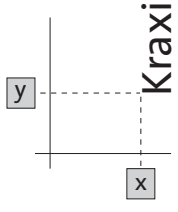


Abb. 8.2
Text mit Orientierung nach Westen

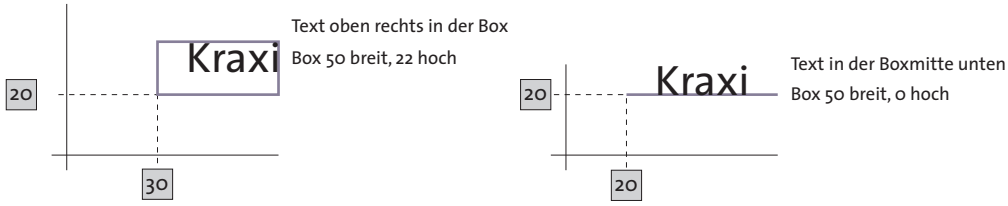


Abb. 8.3 Positionierung des Textes in einer Box

Abbildung 8.1 zeigt die mittige Positionierung des Textes. Sie können die Option *position* auch mit beliebigen anderen Kombinationen der Schlüsselwörter *left, right, center, top* und *bottom* verwenden.

Platzieren von Text mit Orientierung. Nun platzieren wir den Text mit der linken unteren Ecke am Referenzpunkt, und zwar nachdem wir ihn gedreht haben. Das folgende Codefragment orientiert den Text nach Westen (90° gegen den Uhrzeigersinn) und verschiebt dann die untere linke Ecke des gedrehten Textes zum Referenzpunkt *(0, 0)*:

```
p.fit_textline(text, 0, 0, "orientate=west");
```

Abbildung 8.2 veranschaulicht die einfache Platzierung von nach Westen orientiertem Text.

8.1.2 Platzieren von Text in einer Box

Zur Positionierung des Textes können wir zusätzlich eine Box mit definierter Breite und Höhe verwenden und den Text relativ zu dieser Box positionieren. Abbildung 8.3 zeigt das allgemeine Verfahren.

Positionierung von Text in der Box. Wir definieren eine rechteckige Box und platzieren den Text in dieser Box oben rechts. Das folgende Codefragment definiert eine Box mit einer Breite von 50 Einheiten und einer Höhe von 22 Einheiten am Referenzpunkt *(30, 20)*. In Abbildung 8.4a wird der Text in der Box oben rechts platziert.

Wie Abbildung 8.4b zeigt, lässt sich der Text in gleicher Weise in der Mitte unten positionieren.

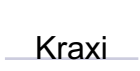
Mit der Option *margin* können wir den Text mit einem gewissen Abstand zur Box platzieren (siehe Abbildung 8.4c).

Beachten Sie, dass das Rechteck bzw. die Linie in den Abbildungen nur zur Visualisierung der Box dient und nicht zur tatsächlichen Ausgabe gehört.

Ausrichtung des Textes an einer horizontalen oder vertikalen Linie. Die Positionierung von Text entlang einer horizontalen oder vertikalen Linie (d.h. an einer Box mit der Breite oder Höhe 0) ist zwar eher ein Spezialfall, kann aber dennoch gelegentlich nützlich sein. In Abbildung 8.4d wird der Text in der Boxmitte unten platziert. Mit der Breite 50 und der Höhe 0 gleicht die Box einer horizontalen Linie.

Um den Text auf einer vertikalen Linie zu zentrieren, orientieren wir ihn nach Westen und positionieren ihn links in der Mitte der Box (siehe Abbildung 8.4e).

Abb. 8.4 Platzieren von Text in einer Box mit verschiedenen Positionierungsoptionen

Generierte Ausgabe	*Optionsliste für PDF_fit_textline( )*
a) 	`boxsize={50 22} position={right top}`
b) 	`boxsize={50 22} position={center bottom}`
c) 	`boxsize={50 22} position={center bottom} margin={0 3}`
d) 	`boxsize={50 0} position={center bottom}`
e) 	`boxsize={0 35} position={left center} orientate=west`

8.1.3 Einpassen von Text in eine Box

In diesem Abschnitt zeigen wir anhand der Option *fitmethod* verschiedene Methoden, um den Text in eine Box einzupassen. Wir gehen davon aus, dass der aktuell eingestellte Font und die Fontgröße in allen Beispielen gleich sind, so dass wir klar sehen, wie sich die Fontgröße und andere Eigenschaften je nach verwendeter Methode implizit ändern.

Beginnen wir mit dem Standardfall: Wir verwenden die Option *fitmethod* nicht, so dass der Text weder skaliert noch beschnitten wird. Wir platzieren ihn in der Mitte einer Box mit 100 Einheiten Breite und 35 Einheiten Höhe (siehe Abbildung 8.5a).

Es hat keinerlei Auswirkungen auf die Ausgabe, wenn wir die Boxbreite von 100 auf 50 Einheiten verringern. Die Fontgröße des Textes bleibt unverändert, und der Text ragt über die Box hinaus (siehe Abbildung 8.5b).

Proportionales Einpassen von Text in eine schmale Box. Wir möchten den Text nun so platzieren, dass er vollständig in die Box hineinpasst, seine Proportionen aber erhalten bleiben. Dies erreichen wir mit der Option *fitmethod=auto*. In Abbildung 8.5c ist die Box groß genug, um den Text unverändert in Originalgröße aufzunehmen.

Verringern wir die Boxbreite jedoch von 100 auf 58, ist der Text zu lang, um vollständig zu passen. Die Methode *auto* versucht in diesem Fall, den Text horizontal zu stauchen, wobei die Option *shrinklimit* (Standardwert: 0,75) berücksichtigt wird. Abbildung 8.5d zeigt den Text, wie er auf 75 Prozent seiner Originalgröße gestaucht ist.

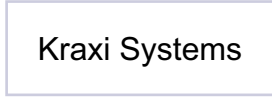
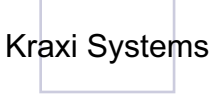
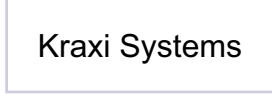
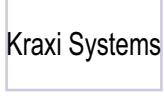
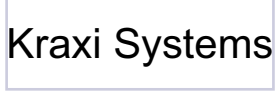
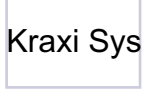
Wenn wir die Boxbreite weiter auf 30 Einheiten verringern, passt auch der gestauchte Text nicht mehr hinein. In diesem Fall kommt die Methode *meet* zum Einsatz, die die Fontgröße so lange verringert, bis der Text vollständig in die Box passt (siehe Abbildung 8.5e).

Einpassen von Text in eine Box bei Vergrößerung des Fonts. Um den Text so einzupassen, dass er unter Beibehaltung seiner Proportionen die gesamte Breite (oder Höhe) einer Box einnimmt, verwenden wir *fitmethod=meet*. Ist die Box breiter als der Text, vergrößert die Methode *meet* den Text, bis dieser so breit wie die Box ist (siehe Abbildung 8.5f).

Vollständiges Einpassen von Text in eine Box. Wir können den Text auch so platzieren, dass er die Box vollständig ausfüllt. Dies bewerkstelligt die Option *fitmethod=entire*. Diese Methode dürfte jedoch selten von Nutzen sein, da sie den Text meist verzerrt (siehe Abbildung 8.5g).

Einpassen von Text in eine Box mit Beschneiden. In einem anderen seltenen Fall möchte man den Text in Originalgröße platzieren und beschneiden, sofern er über die Box hinausragt. Dies bewerkstelligt die Option *fitmethod=clip*. In Abbildung 8.5h wird der Text in einer Box unten links platziert. Da die Box zu schmal ist, wird der Text rechts beschnitten.

Abb. 8.5 Einpassen von Text in eine Box auf der Seite mit verschiedenen Optionen

Generierte Ausgabe	*Optionsliste für PDF_fit_textline( )*
a) 	`boxsize={100 35} position=center fontsize=12`
b) 	`boxsize={50 35} position=center fontsize=12`
c) 	`boxsize={100 35} position=center fontsize=12 fitmethod=auto`
d) 	`boxsize={58 35} position=center fontsize=12 fitmethod=auto`
e) 	`boxsize={30 35} position=center fontsize=12 fitmethod=auto`
f) 	`boxsize={100 35} position=center fontsize=12 fitmethod=meet`
g) 	`boxsize={100 35} position=center fontsize=12 fitmethod=entire`
h) 	`boxsize={50 35} position={left center} fontsize=12 fitmethod=clip`

Vertikales Zentrieren von Text. Die Texthöhe in *PDF_fit_textline( )* entspricht standardmäßig der Versalhöhe *(capheight)*, d.h. der Höhe des Großbuchstabens *H*. Wird der Text in der Boxmitte positioniert, so wird er vertikal gemäß seiner Versalhöhe *(capheight)* zentriert (siehe Abbildung 8.6a).

Um die Höhe einer Textzeile anders festzulegen, nutzen wir das Matchbox-Feature (siehe auch Abschnitt 8.4, »Matchboxen«, Seite 262). Die Option *matchbox* für *PDF_fit_textline( )* definiert die Höhe einer Textzeile als die *capheight* der aktuellen Fontgröße. Die Höhe der Matchbox berechnet sich gemäß der Unteroption *boxheight*. Die Unteroption *boxheight* bestimmt die Ausdehnung des Textes über und unter der Grundlinie. *matchbox={boxheight={capheight none}}* ist die Standardeinstellung, d.h. der obere Rand der Matchbox reicht bis zur Versalhöhe über der Grundlinie, und der untere Rand der Matchbox ragt nicht über die Grundlinie hinaus.

Um die Größe der Matchbox zu veranschaulichen, versehen wir sie mit roter Farbe (siehe Abbildung 8.6b). Abbildung 8.6c zeigt den Text vertikal zentriert gemäß der x-Höhe *(xheight)*, wobei eine Matchbox mit entsprechender Boxhöhe definiert wird.

Abbildung 8.6d–f zeigt die Matchbox (rot) mit verschiedenen nützlichen Einstellungen der *boxheight* zur Festlegung der Höhe, anhand derer der Text in der Box zentriert werden soll.

Abb. 8.6 Proportionales Einpassen von Text in eine Box von unterschiedlicher Höhe

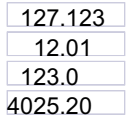
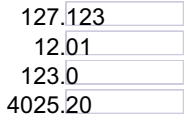
Generierte Ausgabe		*Optionsliste für PDF_fit_textline( )*
a)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto`
b)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={capheight none} fillcolor=mistyrose}`
c)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={xheight none} fillcolor=mistyrose}`
d)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={ascender none} fillcolor=mistyrose}`
e)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={ascender descender} fillcolor=mistyrose}`
f)	Kraxi Systems	`boxsize={80 20} position=center fitmethod=auto matchbox={boxheight={fontsize none} fillcolor=mistyrose}`

8.1.4 Ausrichten von Text an einem Zeichen

Sie können Text an einem bestimmten Zeichen ausrichten, z.B. am Dezimalpunkt einer Zahl. Abbildung 8.7a zeigt, wie der Text in der Mitte der Fitbox positioniert wird. Mit *PDF_fit_textline( )* und der Option *alignchar=.* werden die Zahlen am Zeichen ».« ausgerichtet.

Um die Punkte nicht in der Boxmitte zu platzieren, können Sie die Option *position* weglassen. In diesem Fall wird die Standardeinstellung *position={left bottom}* verwendet, die die Punkte am Referenzpunkt platziert (siehe Abbildung 8.7b). Das Ausrichtungszeichen wird generell mit der rechten unteren Ecke am Referenzpunkt platziert.

Abb. 8.7 Ausrichten einer Textzeile am Zeichen ».«

Generierte Ausgabe	*Optionsliste für PDF_fit_textline( )*
a) 	`boxsize={70 8} position={center bottom} alignchar=.`
b) 	`boxsize={70 8} position={left bottom} alignchar=.`

8.1.5 Platzieren eines Stempels

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `text_output/simple_stamp`.

Als Alternative zu gedrehtem Text bietet die Stempelfunktion eine bequeme Methode, um Text diagonal in einer Box zu platzieren. Um die Fontgröße und Drehung für den Text so zu ermitteln, dass die Box möglichst vollständig ausgefüllt wird, führt die Stempelfunktion komplexe Berechnungen durch. Um einen diagonalen Stempel z.B. auf dem Seitenhintergrund zu platzieren, verwenden Sie die Option *stamp* von *PDF_fit_textline( ). stamp=ll2ur* gibt den Text von der unteren linken zur oberen rechten Ecke der Fitbox aus. *stamp=ul2lr* gibt den Text von der oberen linken zur unteren rechten Ecke der Fitbox aus. Die Option *fitbox* wird ignoriert. Wie Abbildung 8.8 zeigt, werden die Fitbox sowie die Boundingbox des Stempels mit *showborder=true* veranschaulicht.

Abb. 8.8 Platzieren einer Textzeile als Stempel von links unten nach rechts oben

Generierte Ausgabe	*Optionsliste für PDF_fit_textline( )*
	`fontsize=8 boxsize={160 50} stamp=ll2ur showborder=true`

8.1.6 Verwendung von Führungszeichen

Führungszeichen *(leaders)* füllen den Abstand zwischen dem Text und den Rändern der Fitbox. Punkte dienen zum Beispiel häufig als Orientierungshilfe, um in einem Inhaltsverzeichnis die Einträge mit den Seitenzahlen zu verbinden.

Führungszeichen in einem Inhaltsverzeichnis. Mit *PDF_fit_textline( )* und der Option *leader* mit der Unteroption *alignment={none right}* werden die Führungszeichen auf der rechten Seite der Textzeile angefügt und bis zum rechten Rand der Textbox fortgeführt. Zwischen letztem Führungszeichen und rechtem Rand ist der Abstand einheitlich, während der Abstand zwischen Text und erstem Führungszeichen variieren kann (siehe Abbildung 8.9a).

Cookbook Ein vollständiges Codebeispiel zur Ausgabe von Führungszeichen in einer Textzeile finden Sie im Cookbook-Topic `text_output/leaders_in_textline`.

Cookbook Ein vollständiges Codebeispiel zur Ausgabe von Führungszeichen in einem Textflow finden Sie im Cookbook-Topic `text_output/dot_leaders_with_tabs`.

Führungszeichen in einem Newsticker. Um einen Newsticker-Effekt zu erzielen, verwenden wir ein Plus- und ein Leerzeichen »+ « als Führungszeichen. Die Textzeile wird mittig platziert, und die Führungszeichen werden vor und nach der Textzeile ausgegeben *(alignment={left right})*. Die linken und rechten Führungszeichen werden am linken und am rechten Rand ausgerichtet und können somit unterschiedlichen Abstand zum Text besitzen (siehe Abbildung 8.9b).

Abb. 8.9 Platzierung einer Textzeile mit Führungszeichen

Generierte Ausgabe	*Optionsliste für PDF_fit_textline( )*
a) Features of Giant Wing ..... Description of Long Distance Glider ..... Benefits of Cone Head Rocket	`boxsize={200 10}` `leader={alignment={none right}}`

b) + + + + + + + + + Giant Wing in purple! + + + + + + + + +
+ + Long Distance Glider with sensational range! + +
+ + + + + Cone Head Rocket incredibly fast! + + + + +
	`boxsize={200 10}`
	`position={center bottom}`
	`leader={alignment={left right}`
	`text={+ }}`

8.1.7 Text auf einem Pfad

Sie können Text auch auf einen beliebigen Pfad platzieren anstatt auf eine gerade Linie. PDFlib platziert die einzelnen Zeichen dann entlang des Pfades, sodass der Text entlang der Pfadkrümmung verläuft. Mit der Option *textpath* von *PDF_fit_textline( )* können Sie Text auf einem Pfad erzeugen. Der Pfad muss bereits vorher erzeugt worden sein und wird durch ein Pfad-Handle dargestellt. Pfad-Handles können entweder durch die Konstruktion eines Pfades mit *PDF_add_path_point( )* und verwandten Pfadobjekt-Funktionen erstellt werden oder durch Abruf eines Handles für den Beschneidungspfad in einem vorhandenen Rasterbild. Mit dem folgenden Codefragment lässt sich ein einfacher Pfad erzeugen und Text auf dem Pfad platzieren (siehe Abbildung 8.10):

```
/* Ursprungspunkt des Pfades definieren */
path = p.add_path_point(  -1,   0,   0, "move", "");
path = p.add_path_point(path, 100, 100, "control", "");
path = p.add_path_point(path, 200,   0, "circular", "");

/* Text auf dem Pfad platzieren */
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom}");

/* Pfad zeichnen */
p.draw_path(path, x, y, "stroke strokecolor=dodgerblue");
```

Cookbook Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic `text_output/text_on_a_path`.

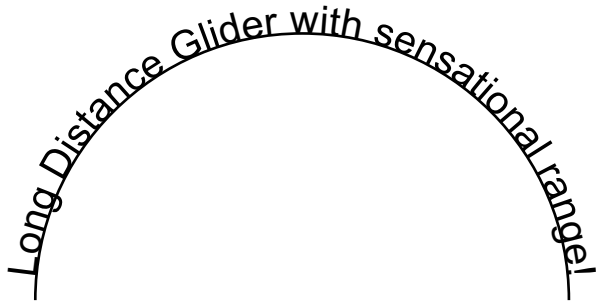


Abb. 8.10
Text auf einem Pfad

Bild-Beschneidungspfade zum Platzieren von Text. Alternativ zur manuellen Konstruktion eines Pfadobjekts mittels Pfadfunktionen können Sie den Beschneidungspfad aus einem Bild extrahieren und Text auf diesem Pfad platzieren. Das Bild muss mit der Option *honorclippingpath* geladen werden und wenn der gewünschte Pfad nicht als Standard-Beschneidungspfad im Bild festgelegt ist, muss die Option *clippingpathname* ebenfalls an *PDF_load_image( )* übergeben werden:

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* Pfadobjekt aus dem Beschneidungspfad des Bildes erzeugen */
path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
        throw new Exception("Error: clipping path not found!");

/* Text auf dem Pfad platzieren */
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom}");
```

Erzeugen von Abstand zwischen Pfad und Text. PDFlib platziert normalerweise einzelne Zeichen direkt auf den Pfad, das heißt ohne Abstand zwischen Glyphen und Pfad. Um Abstand zu erzeugen, können Sie die Zeichenboxen vergrößern. Dies lässt sich mit der Unteroption *boxheight* der Option *matchbox* erreichen, die die vertikale Ausdehnung der Zeichenboxen festlegt. Die folgende Optionsliste berücksichtigt die Unterlängen (siehe Abbildung 8.11):

```
p.fit_textline("Long Distance Glider with sensational range!", x, y,
    "textpath={path=" + path + "} position={center bottom} " +
    "matchbox={boxheight={capheight descender}}");
```

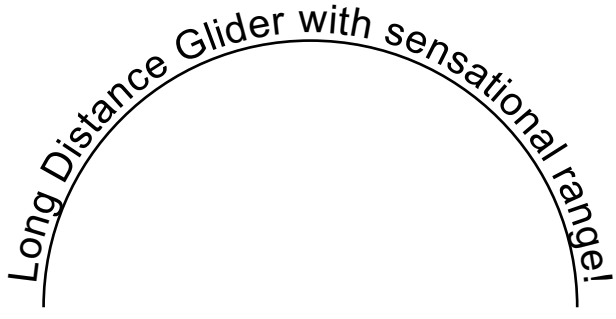


Abb. 8.11
Text auf einem Pfad, mit Abstand zum Pfad

8.1.8 Schattentext

Mit der Option *shadow* lässt sich ein Schatteneffekt für den Text erzeugen. Sie können die Schattenfarbe sowie den horizontalen und vertikalen Abstand zum Haupttext in Unteroptionen festlegen:

```
p.fit_textline("Long Distance Glider", x, y,
    "fillcolor=rosybrown shadow={offset={3, -3}}");
```

8.2 Mehrzeilige Textflows

PDFlib bietet unter der Bezeichnung *Textflow* die Möglichkeit, nicht nur einzelne Textzeilen, sondern einen beliebig langen zusammenhängenden Text auszugeben. Der Text kann sich über beliebig viele Zeilen, Spalten oder Seiten erstrecken und anhand vieler Optionen formatiert werden. Zeicheneigenschaften wie Font, Schriftgröße oder Farbe sind auf beliebige Textabschnitte anwendbar. Textflow-Eigenschaften, wie zum Beispiel Blocksatz oder Flattersatz, Einzüge und Tabulatorabstände können eingestellt werden. Trennstellen, die im Text durch weiche Trennzeichen *(soft hyphen)* gekennzeichnet sind, werden berücksichtigt. Abbildung 8.12 und Abbildung 8.13 zeigen, wie verschiedene Elemente einer Rechnung als Textflows platziert ausgegeben werden. Die einzelnen Optionen werden in den folgenden Abschnitten genauer erläutert.

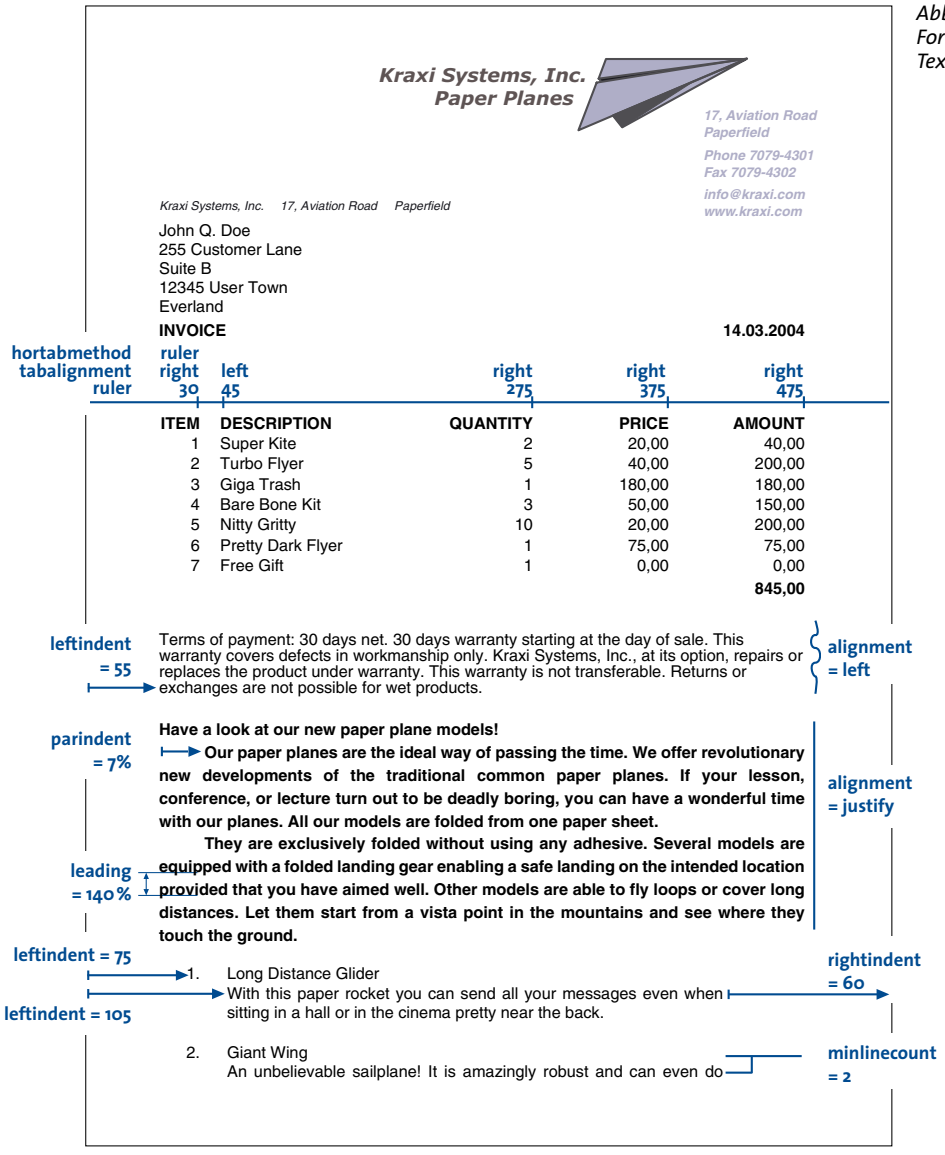


Abb. 8.12 Formatierung von Textflows

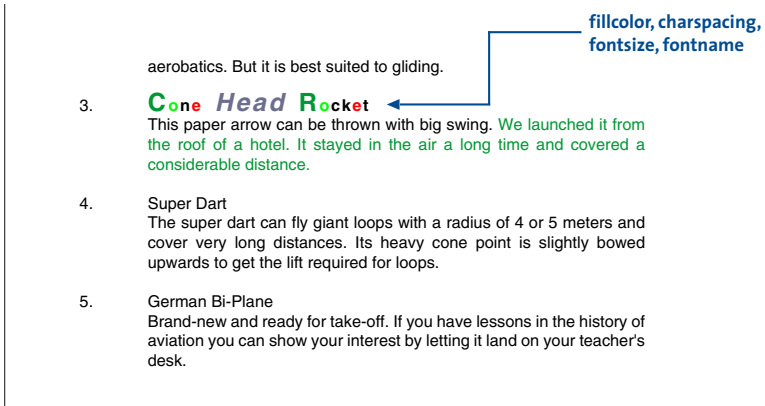


Abb. 8.13 Formatierung von Textflows

Ein mehrzeiliger Textflow wird in eines oder mehrere Rechtecke – die so genannte Fitbox – auf einer oder mehreren Seiten platziert. Zur Platzierung des Textflows sind folgende Schritte erforderlich:

- Die Funktion *PDF_add_textflow( )* nimmt den Text ganz oder portionsweise inklusive der Formatierungsanweisungen entgegen, erzeugt ein Textflow-Objekt und gibt ein Handle zurück. Alternativ dazu analysiert die Funktion *PDF_create_textflow( )* den Text vollständig in einem einzigen Aufruf, wobei der Text zusätzlich »Inline«-Optionen zur Formatierung enthalten kann. Bei diesen Funktionen wird noch kein Text auf der Seite platziert.
- Die Funktion *PDF_fit_textflow( )* platziert den Textflow ganz oder teilweise in der übergebenen Fitbox. Um den Textflow vollständig zu platzieren, muss dieser Schritt möglicherweise mehrmals wiederholt werden, wobei jeweils eine neue Fitbox – entweder auf derselben oder einer neuen Seite – übergeben wird.
- Die Funktion *PDF_delete_textflow( )* löscht das Textflow-Objekt nach seiner Platzierung im Dokument.

Die Funktionen *PDF_add/create_textflow( )* zur Erzeugung eines Textflows bieten eine Fülle von Optionen zur Steuerung der Formatierung. Diese können entweder in der Optionsliste übergeben oder bei *PDF_create_textflow( )* als »Inline«-Optionen in den Text eingebettet werden. Mit *PDF_info_textflow( )* lassen sich Details zur vorgenommenen Formatierung und zahlreiche andere Informationen zum Textflow abfragen. Die Platzierung eines Textflows wird im Folgenden anhand einiger häufig vorkommender Anwendungsbeispiele erläutert. Eine vollständige Liste aller Textflow-Optionen finden Sie in der PDFlib-Referenz.

Viele der Optionen, die in *PDF_add/create_textflow( )* genutzt werden können, sind mit denen für *PDF_fit_textline( )* identisch. Sie sollten sich deshalb bereits die Beispiele in Abschnitt 8.1, »Platzieren und Einpassen von einzeiligem Text«, Seite 215, angesehen haben. Wir werden hier nur Optionen beschreiben, die sich auf mehrzeiligen Text beziehen.

Cookbook *Codebeispiele zu Textausgabe finden Sie in der Kategorie* `text_output` *des PDFlib Cookbook.*

8.2.1 Platzierung eines Textflows in der Fitbox

Die Fitbox entspricht bei Textflows dem Bereich, in den der Text platziert wird. Sie definiert sich als der rechteckige Bereich, der durch die Parameter *llx, lly, urx, ury* von *PDF_fit_textflow( )* festgelegt ist.

Platzierung in einer Fitbox. Beginnen wir mit einem einfachen Beispiel. Das folgende Codefragment kombiniert mit zwei Aufrufen von *PDF_add_textflow( )* fett gesetzten und normalen Text. Schrift, Schriftgröße und Encoding werden explizit angegeben. Im ersten Aufruf von *PDF_add_textflow( )* wird -1 übergeben. Die Funktion gibt ein Textflow-Handle zurück, das bei Bedarf in weiteren Aufrufen von *PDF_add_textflow( )* verwendet werden kann. *text1* und *text2* enthalten den auszugebenden Text.

Mit *PDF_fit_textflow( )* wird der erhaltene Textflow anhand von Standardformatierungsoptionen in einer Fitbox auf der Seite platziert.

```
/* Text in Fettschrift hinzufügen */
tf = p.add_textflow(-1, text1, "fontname=Helvetica-Bold fontsize=9 encoding=unicode");
if (tf == -1)
        throw new Exception("Error: " + p.get_errmsg());

/* Text in normaler Schrift hinzufügen */
tf = p.add_textflow(tf, text2, "fontname=Helvetica fontsize=9 encoding=unicode");
if (tf == -1)
        throw new Exception("Error: " + p.get_errmsg());

/* Gesamten Text platzieren */
result = p.fit_textflow(tf, left_x, left_y, right_x, right_y, "");
if (!result.equals("_stop"))
        { /* ... */}

p.delete_textflow(tf);
```

Platzierung in zwei Fitboxen auf mehreren Seiten. Passt ein mit *PDF_fit_textflow( )* ausgegebener Textflow nicht vollständig in die Fitbox, so wird die Ausgabe unterbrochen und der String *_boxfull* zurückgegeben. PDFlib merkt sich die bereits ausgegebenen Zeichen und fährt beim nächsten Aufruf von *PDF_fit_textflow( )* an der unterbrochenen Stelle fort. Unter Umständen muss eine neue Seite angelegt werden. Das folgende Codefragment zeigt die Ausgabe des Textflows in zwei Fitboxen pro Seite, wobei so lange neue Seiten erzeugt werden, bis der Text vollständig platziert wurde (siehe Abbildung 8.14).

Cookbook Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic `text_output/starter_textflow`.

```
/* Schleife durchlaufen, bis der Text vollständig platziert ist; neue Seite anlegen,
 * solange noch Text zu platzieren ist. Auf jeder Seite zwei Spalten erstellen.
*/
do
{
        String optlist = "verticalalign=justify linespreadlimit=120%";

        p.begin_page_ext(0, 0, "width=a4.width height=a4.height");

        /* Erste Spalte füllen */
        result = p.fit_textflow(tf, llx1, lly1, urx1, ury1, optlist);
```

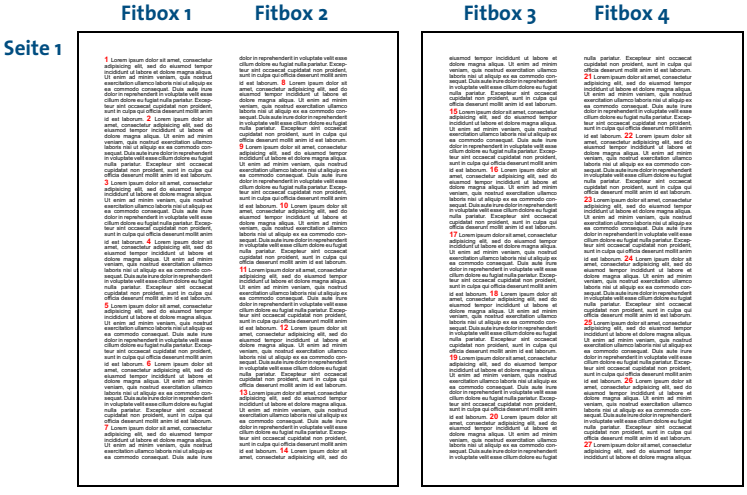


Abb. 8.14
Platzieren eines Textflows in mehreren Fitboxen

```
        /* Bei weiterem Text zweite Spalte füllen */
        if (!result.equals("_stop"))
            result = p.fit_textflow(tf, llx2, lly2, urx2, ury2, optlist);

        p.end_page_ext("");

        /* "_boxfull" zeigt an, dass noch Text zur Platzierung ansteht;
         * "_nextpage" wird als "neue Spalte beginnen" interpretiert
         */
} while (result.equals("_boxfull") || result.equals("_nextpage"));

/* Auf Fehler überprüfen */
if (!result.equals("_stop"))
{
        /* "_boxempty" tritt auf, wenn die Box so klein ist, dass sie keinen Text
         * aufnehmen kann.
        */
        if (result.equals( "_boxempty"))
            throw new Exception("Error: " + p.get_errmsg());
        else
        {
            /* Jeder andere Rückgabewert ist benutzerspezifisch durch die Option "return"
             * definiert und erfordert entsprechend angepassten Code. */
        }
}
p.delete_textflow(tf);
```

8.2.2 Optionen für die Absatzformatierung

Die obigen Beispiele wurden mit Standardformatierungen ausgegeben. So ist die Ausrichtung *(alignment)* des Textes zum Beispiel standardmäßig linksbündig, der Zeilenabstand *(leading)* ist gleich 100% und entspricht damit der Fonthöhe.

Um die Absatzformatierung individuell zu gestalten, können wir die Funktion *PDF_add_textflow()* mit weiteren Optionen aufrufen. Beispielsweise können wir den Text links und rechts mit einem Einzug vom Seitenrand versehen, und zwar links um 15 und rechts um 20 Einheiten. Die erste Zeile jedes Absatzes soll zusätzlich um 20 Einheiten

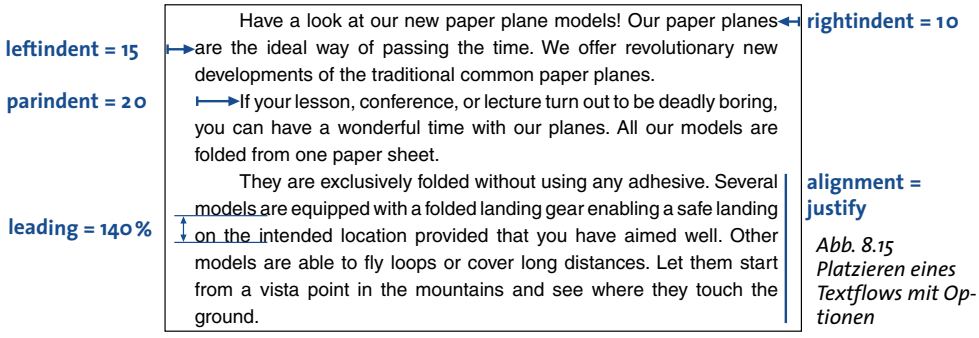


Abb. 8.15 Platzieren eines Textflows mit Optionen

eingerückt sein. Als Textausrichtung wählen wir Blocksatz, und den Abstand zwischen den einzelnen Zeilen erhöhen wir auf 140%. Außerdem reduzieren wir die Schriftgröße auf 8 Einheiten. Die erweiterte Optionsliste für *PDF_add_textflow( )* sieht dann wie folgt aus (siehe Abbildung 8.15):

```
String  optlist =
        "leftindent=15 rightindent=10 parindent=20 alignment=justify " +
        "leading=140% fontname=Helvetica fontsize=8 encoding=unicode";
```

8.2.3 Inline-Optionen und Makros

Der Text in Abbildung 8.15 ist noch nicht perfekt. Die Überschrift »Have a look at our new paper plane models!« gehört in eine eigene Zeile. Außerdem soll sie in größerer Schrift erscheinen und mittig ausgerichtet sein. Um dies zu bewerkstelligen, gibt es mehrere Lösungen.

Inline-Optionslisten für PDF_create_textflow( ). Bislang haben wir die Formatierungsoptionen im Funktionsaufruf übergeben. Um mit dieser Technik fortzufahren, müssten wir den Text aufteilen und in zwei Aufrufen platzieren – einen für die Überschrift und einen für die restlichen Absätze. Dies wäre aber recht umständlich, insbesondere dann, wenn es noch weitere Formatwechsel gibt.

In solchen Fällen ist es sinnvoll, *PDF_create_textflow( )* statt *PDF_add_texflow( )* zu verwenden. *PDF_create_textflow( )* interpretiert den Text inklusive so genannter Inline-Optionen, die direkt in den Text eingebettet sind. Inline-Optionslisten werden innerhalb des Textes übergeben und standardmäßig zwischen den Zeichen »<« und »>« eingeschlossen. Wir integrieren also die Formatierungen für die Überschrift und für die restlichen Absätze wie folgt in unseren Text.

Hinweis *Inline-Optionslisten sind in allen folgenden Beispielen farblich hervorgehoben; Absatzendezeichen werden durch Pfeile angedeutet.*

```
<leftindent=15 rightindent=10 alignment=center fontname=Helvetica fontsize=12
encoding=winansi>Have a look at our new paper plane models! ↵
<alignment=justify fontname=Helvetica leading=140% fontsize=8 encoding=winansi>
Our paper planes are the ideal way of passing the time. We offer
revolutionary new developments of the traditional common paper planes. ↵
<parindent=20>If your lesson, conference, or lecture
turn out to be deadly boring, you can have a wonderful time
with our planes. All our models are folded from one paper sheet. ↵
They are exclusively folded without using any adhesive. Several
models are equipped with a folded landing gear enabling a safe
```

```
landing on the intended location provided that you have aimed well.
Other models are able to fly loops or cover long distances. Let them
start from a vista point in the mountains and see
where they touch the ground.
```

Die Zeichen zur Klammerung von Optionslisten können mit den Optionen *begoptlistchar* und *endoptlistchar* umdefiniert werden. Wenn Sie für die Option *begoptlistchar* das Schlüsselwort *none* übergeben, wird die Ermittlung von Optionslisten unterbunden. Dies ist zum Beispiel sinnvoll, wenn der Text keine Inline-Optionslisten enthält und Sie gewährleisten möchten, dass »<« und »>« als normale Zeichen interpretiert werden.

Symbolzeichen und Inline-Optionslisten. Bei Textflow können Symbolzeichen auch zusammen mit Inline-Optionslisten verwendet werden. Der Code für das Zeichen, das eine Inline-Optionsliste einleitet (standardmäßig '<' U+003C) wird bei einem Font mit *encoding=builtin* innerhalb eines Textes nicht als Symbolcode interpretiert. Um die Symbolglyphe mit dem gleichen Code auszuwählen, können Sie das so lösen wie bei Textfonts, nämlich durch Neudefinition des Startzeichens mit der Option *begoptlistchar* oder durch Angabe der Anzahl der Symbolglyphen in der Option *textlen*. Beachten Sie, dass Character-Referenzen (z.B. *<)* nicht als Workaround verwendet werden können.

Makros. Wir haben es in obigem Text im Prinzip mit verschiedenen Absatztypen zu tun, zum Beispiel Überschrift, Text ohne Einzug oder Text mit Einzug. Jeder dieser Absatztypen ist individuell formatiert und tritt in längeren Texten mehrmals auf. Damit wir einem Absatztyp nicht jedes Mal die zugehörigen Inline-Optionen voranstellen müssen, können wir diese zu Makros zusammenfassen und in den Text einfach die Makronamen einbetten. Wie Abbildung 8.16 zeigt, definieren wir für das obige Beispiel die drei Makros *H1* für die Formatierung der Überschrift, *Body* für Absätze ohne Einzug und *Body_indented* für Absätze mit Einzug. Zur Verwendung eines Makros setzen wir das Zeichen & vor den Namen und schreiben ihn in eine Optionsliste. Das folgende Codefragment definiert drei Makros anhand der bereits verwendeten Inline-Optionen und setzt diese in den Text ein:

```
<macro {
H1 {leftindent=15 rightindent=10 alignment=center
fontname=Helvetica fontsize=12 encoding=winansi}
```

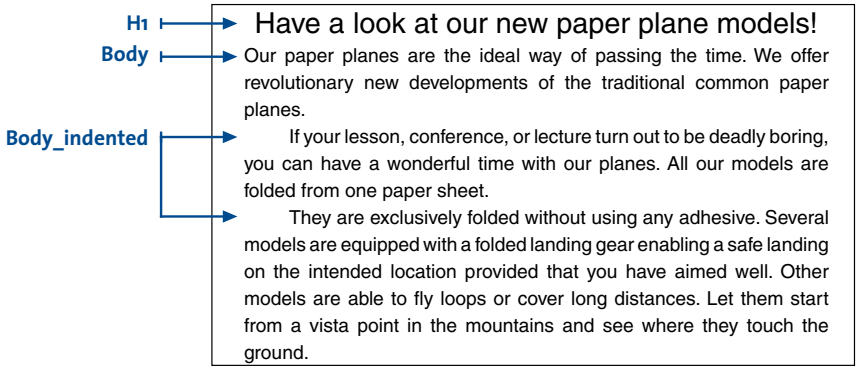


Abb. 8.16 Zusammenfassung von Inline-Optionen zu Makros

```
Body {leftindent=15 rightindent=10 alignment=justify leading=140%
fontname=Helvetica fontsize=8 encoding=winansi}

Body_indented {parindent=20 leftindent=15 rightindent=10 alignment=justify
leading=140% fontname=Helvetica fontsize=8 encoding=winansi}
}>
<&H1>Have a look at our new paper plane models! ↵
<&Body>Our paper planes are the ideal way of passing the time. We offer
revolutionary new developments of the traditional common paper planes. ↵
<&Body_indented>If your lesson, conference, or lecture
turn out to be deadly boring, you can have a wonderful time
with our planes. All our models are folded from one paper sheet. ↵
They are exclusively folded without using any adhesive. Several
models are equipped with a folded landing gear enabling a safe
landing on the intended location provided that you have aimed well.
Other models are able to fly loops or cover long distances. Let them
start from a vista point in the mountains and see
where they touch the ground.
```

Explizites Setzen von Optionen. Beachten Sie, dass alle Optionen, die in Makros nicht gesetzt werden, ihren alten Wert beibehalten. Um Nebenwirkungen durch unerwünschte »Vererbung« von Optionen zu vermeiden, sollten Sie daher alle gewünschten Einstellungen explizit in Makros festlegen. Dadurch stellen Sie sicher, dass sich die Makros unabhängig von der Reihenfolge oder Kombination mit anderen Optionslisten immer gleich verhalten.

Andererseits können Sie diese Eigenschaft auch für Makros nutzen, um bestimmte Einstellungen bewusst aus dem jeweiligen Kontext zu übernehmen, statt sie explizit festzulegen. So könnte ein Makro zum Beispiel die Schriftart festlegen, ohne die Option *fontsize* anzugeben. Die Schriftgröße entspricht dann automatisch der des vorangehenden Textes.

Inline-Optionen oder Optionen als Funktionsparameter? Bei der Ausgabe von Textflows ist es wichtig zu unterscheiden, ob der Text im Programm selbst kodiert ist oder aus einer externen Quelle stammt und ob die Formatierung und der Text getrennt gehalten werden. Der reine Textinhalt stammt in der Regel aus einer externen Quelle, etwa einer Datenbank. In der Praxis sind also die folgenden Anwendungsfälle zu berücksichtigen:

- Inhalt aus externer Quelle, Formatierungsoptionen im Programm: Aus einer externen Datenbank kommen kleinere Textfragmente, die im Programmcode zusammengesetzt und erst zur Laufzeit mit Formatierungsoptionen (im Funktionsaufruf) angereichert werden.
- Inhalt und Formatierungsoptionen aus externer Quelle: Größere Textmengen einschließlich der Formatierungsoptionen kommen aus einer externen Quelle. Die Formatierung wird dabei durch Inline-Optionen im Text bereitgestellt, die einfache Optionen oder Makros sein können. Bei der Verwendung von Makros ist zwischen Makrodefinitionen und Makroaufrufen zu trennen. Daraus ergibt sich eine interessante Mischform. Der Inhalt kommt von einer externen Quelle mit Makroaufrufen zur Formatierung; die Makrodefinitionen werden aber erst zur Laufzeit zugemischt. Mischt man die Makrodefinitionen erst zur Laufzeit hinzu, kann man die Formatierung mit minimalem Aufwand beeinflussen, ohne dass der extern zugelieferte Text geändert werden muss. Zum Erstellen von Grußkarten könnte man zum Beispiel

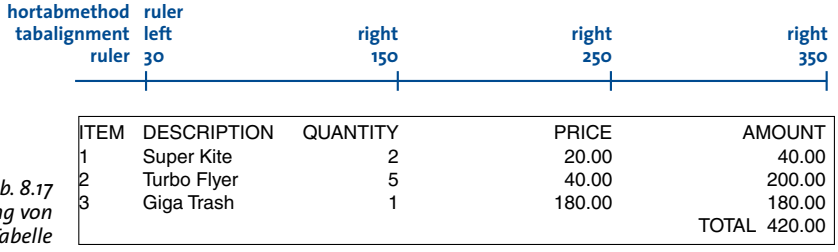


Abb. 8.17 Platzierung von Text als Tabelle

verschiedene Stile definieren (via Makros), die der Karte einen romantischen, nüchternen oder anderen Touch geben.

8.2.4 Tabulatoren

Im Folgenden geht es um die Platzierung einer einfachen Tabelle mit links- und rechtsbündigen Spalten unter Verwendung von Tabulatoren. Die Tabelle enthält folgende Zeilen, deren Einträge durch Tabulatoren getrennt sind (die Tabulatorzeichen werden durch Pfeile symbolisiert):

```
ITEM → DESCRIPTION → QUANTITY → PRICE → AMOUNT ↵
1 → Super Kite → 2 → 20.00 → 40.00 ↵
2 → Turbo Flyer → 5 → 40.00 → 200.00 ↵
3 → Giga Trash → 1 → 180.00 → 180.00 ↵
→ → → → TOTAL 420.00
```

Mit *PDF_add/create_textflow( )* und folgender Optionsliste können Sie diese einfache Tabelle platzieren. Die Option *ruler* definiert die Tabulatorabstände, *tabalignment* die Tabulatorausrichtung und *hortabmethod* die Methode zum Umgang mit Tabulatoren (siehe Abbildung 8.17):

```
String  optlist =
        "ruler       ={30     150   250   350} " +
        "tabalignment={left right right right} " +
        "hortabmethod=ruler leading=120% fontname=Helvetica fontsize=9 encoding=winansi";
```

Cookbook *Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic* `text_output/tabstops_in_textflow`.

Hinweis *Komplexere Tabellen lassen sich bequem mit der Tabellenfunktion von PDFlib erstellen (siehe Abschnitt 8.3, »Tabellenformatierung«, Seite 245).*

8.2.5 Nummerierte Listen und Abstände zwischen Absätzen

Das folgende Beispiel zeigt, wie anhand der Inline-Option *leftindent* für den linken Einzug eine nummerierte Liste formatiert wird (siehe Abbildung 8.18):

```
1.<leftindent 10>Long Distance Glider: With this paper rocket you can send all
your messages even when sitting in a hall or in the cinema pretty near the back. ↵
<leftindent 0>2.<leftindent 10>Giant Wing: An unbelievable sailplane! It is amazingly
robust and can even do aerobatics. But it is best suited to gliding. ↵
<leftindent 0>3.<leftindent 10>Cone Head Rocket: This paper arrow can be thrown with big
swing. We launched it from the roof of a hotel. It stayed in the air a long time and
covered a considerable distance.
```

1. Long Distance Glider: With this paper rocket you can send all your messages even when sitting in a hall or in the cinema pretty near the back.
2. Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do aerobatics. But it is best suited to gliding.
3. Cone Head Rocket: This paper arrow can be thrown with big swing. We launched it from the roof of a hotel. It stayed in the air a long time and covered a considerable distance.

Abb. 8.18
Nummerierte Liste

Cookbook *Vollständige Codebeispiele zu Aufzählungen und nummerierten Listen finden Sie in den Cookbook-Topics* `text_output/bulleted_list` *und* `text_output/numbered_list`.

Das Setzen und Zurücksetzen des Einzugs ist mühsam, zumal es in jedem Absatz erforderlich ist. Die elegantere Variante arbeitet mit dem Makro *list*, das der Bequemlichkeit halber das Makro *indent* verwendet, das als Konstante dient. Die Makros sind wie folgt definiert:

```
<macro {
indent {25}

list {parindent=-&indent leftindent=&indent hortabsize=&indent
hortabmethod=ruler ruler={&indent}}
}>
<&list>1. →  Long Distance Glider: With this paper rocket you can send all your messages
even when sitting in a hall or in the cinema pretty near the back. ↵
2. →  Giant Wing: An unbelievable sailplane! It is amazingly robust and can even do
aerobatics. But it is best suited to gliding. ↵
3. →  Cone Head Rocket: This paper arrow can be thrown with big swing. We launched
it from the roof of a hotel. It stayed in the air a long time and covered a
considerable distance.
```

Mit der Option *leftindent* wird der linke Einzug festgelegt. Mit der Option *parindent*, die genau dem negativen *leftindent* entspricht, wird der linke Einzug für die jeweils erste Zeile eines Absatzes rückgängig gemacht. Mit den Optionen *hortabsize*, *hortabmethod* und *ruler* wird der Tabulator festgelegt, der *leftindent* entspricht und bewirkt, dass der Text nach der Nummer genau um *leftindent* eingerückt wird. Abbildung 8.19 zeigt die Funktionsweise von *parindent* und *leftindent*.

Abstand zwischen zwei Absätzen. Meist soll der Abstand zwischen benachbarten Absätzen um einiges größer als der Zeilenabstand sein. Dies lässt sich durch Einfügen einer leeren Zeile erreichen (die mit der Option *nextline* erzeugt wird), wobei für die Leerzeile ein geeigneter Zeilenabstand *(leading)* angegeben wird. Dieser Wert bezeichnet den

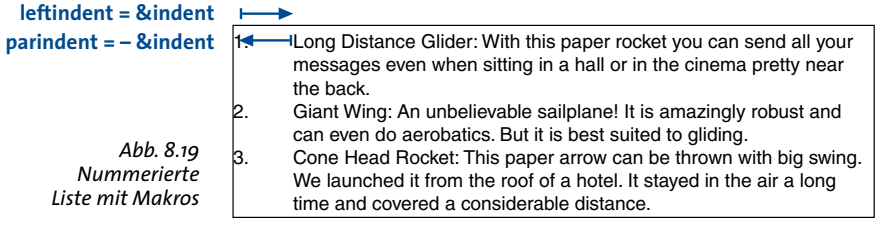


Abb. 8.19
Nummerierte Liste mit Makros

Abstand von der Grundlinie der letzten Zeile des vorangehenden Absatzes zur Grundlinie der Leerzeile. Das folgende Beispiel erzeugt 80% zusätzlichen Abstand zwischen den beiden Absätzen (wobei 100% der zuletzt gesetzten Fontgröße entspricht):

```
1. → Long Distance Glider: With this paper rocket you can send all your messages
even when sitting in a hall or in the cinema pretty near the back.
<nextline leading=80%><nextparagraph leading=100%>2. → Giant Wing: An unbelievable
sailplane! It is amazingly robust and can even do aerobatics. But it is best suited to
gliding.
```

Cookbook *Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic* `text_output/distance_between_paragraphs`.

8.2.6 Steuerzeichen und Zeichenersetzung

Steuerzeichen in Textflows. Es gibt eine Reihe von Zeichen, die in Textflows besonders behandelt werden. Außerdem unterstützt PDFlib symbolische Namen für die Option *charmapping*, die statt der Zeichencodes verwendet werden können (diese Option ersetzt Text vor der Verarbeitung, siehe unten). Tabelle 8.1 zeigt alle ausgewerteten Steuerzeichen mit ihren symbolischen Namen und erklärt ihre Bedeutung. Eine Option darf nur einmal pro Optionsliste angegeben werden, allerdings können mehrere Optionslisten direkt aufeinander folgen. Die folgende Sequenz erzeugt zum Beispiel eine Leerzeile:

```
<nextline><nextline>
```

Tabelle 8.1 Steuerzeichen und ihre Bedeutung in Textflows

Unicode-Zeichen	*Entity-Name*	*Textflow-Option*	*Bedeutung in Textflows*
U+0020	*SP, space*	*space*	*Leerzeichen, wird zum Wortausgleich und Umbrechen benutzt*
U+00A0	*NBSP, nbsp*	*(keine)*	*(no-break space) Umbruchgeschütztes Leerzeichen*
U+202F	*NNBSP, nnbsp*	*(none)*	*(narrow no-break space) Umbruchgeschütztes Leerzeichen mit fester Breite, die nicht gemäß der Formatierungsoptionen geändert wird*
U+0009	*HT, hortab*	*(keine)*	*Horizontaler Tabulator: wird gemäß der Optionen* `ruler`, `tabalignchar` *und* `tabalignment` *ausgewertet*
U+002D	*HY, hyphen*	*(keine)*	*Trennzeichen bei Worttrennung*
U+00AD	*SHY, shy*	*(keine)*	*(soft hyphen) Weiches Trennzeichen (mögliche Trennstelle), nur sichtbar bei einer Umbruchstelle*
U+000B *U+2028*	*VT, verttab* *LS, linesep*	*nextline*	*(next line) Erzwingt eine neue Zeile*
U+000A *U+000D* *U+000D und U+000A* *U+0085* *U+2029*	*LF, linefeed* *CR, return* *CRLF* *NEL, newline* *PS, parasep*	*nextparagraph*	*(next paragraph) Wie* `nextline`*; zusätzlich wirkt die Option* `parindent` *auf die nächste Zeile*
U+000C	*FF, formfeed*	*return*	*PDF_fit_textflow( ) stoppt und gibt den String* `_nextpage` *zurück.*

Ersetzung/Entfernen von Zeichen oder gleichartigen Zeichenfolgen. Mit der Option *charmapping* können die Zeichen eines Textes bei der Ausgabe durch andere Zeichen ersetzt oder entfernt werden. Beginnen wir mit einem einfachen Fall, in dem wir alle Tabulatorzeichen durch Leerzeichen ersetzen. Die *charmapping*-Option hierfür lautet:

```
charmapping {hortab space}
```

Dabei sind *hortab* und *space* symbolische Namen. Für mehrere Ersetzungen können wir folgende Ergänzung vornehmen, die jeden Tabulator und Zeilenumbruch durch ein Leerzeichen ersetzt:

```
charmapping {hortab space    CRLF space    LF space    CR space}
```

Das folgende Beispiel entfernt alle weichen Trennstellen:

```
charmapping {shy {shy 0}}
```

Jedes Tabulator-Zeichen wird durch vier Leerzeichen ersetzt:

```
charmapping {hortab {space 4}}
```

Jede beliebig lange Folge von Linefeed-Zeichen wird auf ein einziges Linefeed-Zeichen reduziert:

```
charmapping {linefeed {linefeed -1}}
```

Jede Folge von CRLF-Kombinationen wird durch ein einziges Leerzeichen ersetzt:

```
charmapping {CRLF {space -1}}
```

Führen wird das letzte Beispiel etwas genauer aus. Angenommen, man erhält einen Text, dessen Zeilen von anderer Software durch feste Zeilenumbrüche getrennt wurden und deswegen nicht mehr richtig umbrechen. Man möchte die Umbrüche durch Leerzeichen ersetzen, um wieder einen Umbruch am Rand der Fitbox zu erhalten. Dazu ersetzen wir beliebig lange Folgen von Zeilenumbrüchen durch ein einziges Leerzeichen. Der zu verbessernde Text lautet:

```
To fold the famous rocket looper proceed as follows: ↵ ↵
Take a sheet of paper. Fold it ↵
lengthwise in the middle. ↵
Then, fold down the upper corners. Fold the ↵
long sides inwards ↵
that the points A and B meet on the central fold.
```

Das folgende Codefragment zeigt die Ersetzung der überflüssigen Umbrüche und Formatierung des derart korrigierten Textes:

```
/* Optionsliste zusammenstellen */
String  optlist =
        "fontname=Helvetica fontsize=9 encoding=winansi alignment=justify " +
        "charmapping {CRLF {space -1}}"

/* Textflow in Fitbox platzieren */
textflow = p.add_textflow(-1, text, optlist);
if (textflow == -1)
        throw new Exception("Error: " + p.get_errmsg());
```

```
result = p.fit_textflow(textflow, left_x, left_y, right_x, right_y, "");
if (!result.equals("_stop"))
        { /* ... */ }

p.delete_textflow(textflow);
```

Abbildung 8.20 zeigt die Ausgabe des unveränderten Textes und seine »Reparatur« mit Hilfe der Option *charmapping*.

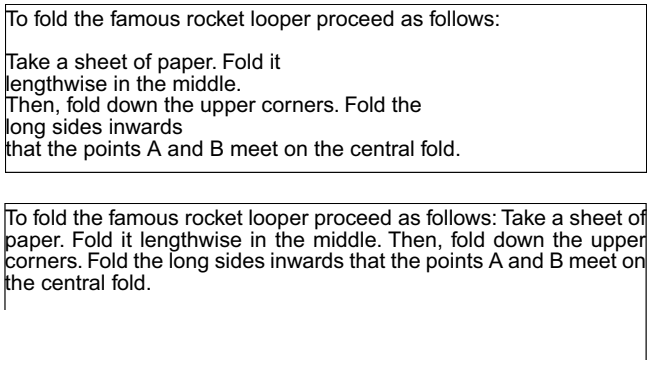


Abb. 8.20
Oben: Text mit überflüssigen Umbrüchen

Unten: Umwandlung der Umbrüche mit der Option `charmapping`

8.2.7 Silbentrennung

PDFlib trennt Text nicht automatisch, kann Wörter aber an Stellen trennen, die im Text explizit mit weichen Trennzeichen *(soft hyphen)* markiert sind. Das weiche Trennzeichen befindet sich in Unicode an Position *U+00AD*. In anderen Encodings als von Unicode kann das weiche Trennzeichen wie folgt angegeben werden:

- In den Encodings *cp1250 – cp1258* (einschließlich *winansi)* und *iso8859-1 – iso8859-16* entspricht das weiche Trennzeichen dezimal 173, oktal 255 bzw. hexadezimal 0xAD.
- Im Encoding *ebcdic* entspricht das weiche Trennzeichen dezimal 202, oktal 312 bzw. hexadezimal 0xCA.
- Wenn ein Encoding (z.B. *macroman)* über keine weichen Trennzeichen verfügt, kann eine Entity-Referenz verwendet werden: **

Als Trennzeichen wird die Glyphe für *U+00AD* verwendet, falls diese im Font enthalten ist, andernfalls *U+002D*. Außer an den Trennstellen, die durch weiche Trennzeichen gekennzeichnet sind, müssen Wörter manchmal gewaltsam getrennt werden, wenn andere Methoden der Formatierung, etwa das Ändern des Wortabstands oder Stauchen des Textes, nicht ausreichen.

Blocksatz mit und ohne Trennzeichen. Im ersten Beispiel soll der folgende Text, der bereits mit Softhyphens versehen ist, im Blocksatz ausgegeben werden (die Softhyphen-Zeichen sind hier der Deutlichkeit halber als kleine Balken dargestellt):

```
Our paper planes are the ideal way of pas – sing the time. We offer revolu – tionary
brand new dev – elop – ments of the tradi – tional common paper planes. If your lesson,
confe – rence, or lecture turn out to be deadly boring, you can have a wonder – ful time
with our planes. All our models are folded from one paper sheet. They are exclu – sively
folded without using any adhe – sive. Several models are equip – ped with a folded
landing gear enab – ling a safe landing on the intended loca – tion provided that you
```

Abb. 8.21
Blocksatz mit Standardeinstellungen in breiter Fitbox mit expliziten »weichen« Trennzeichen

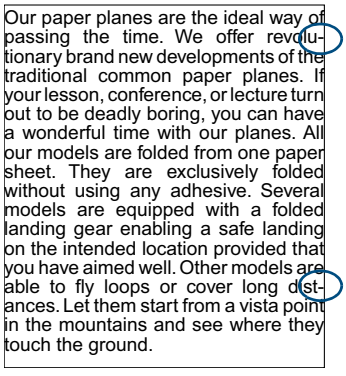


Abb. 8.22
Blocksatz mit Standardeinstellungen in breiter Fitbox ohne explizite Trennzeichen


```
have aimed well. Other models are able to fly loops or cover long dist – ances. Let them
start from a vista point in the mount – ains and see where they touch the ground.
```

Abbildung 8.21 zeigt die Ausgabe des Textes mit Standardeinstellungen für den Blocksatz. Die Ausgabe ist tadellos, da die Voraussetzungen optimal sind: Die Fitbox ist breit genug und es sind explizit im Text gesetzte Trennzeichen vorhanden. Wie Abbildung 8.22 zeigt, erhalten Sie in einer breiten Fitbox mit Standardeinstellungen auch dann eine gute Ausgabe, wenn im Text keine expliziten Trennzeichen vorhanden sind. Die Optionsliste lautet in beiden Fällen:

```
fontname=Helvetica fontsize=9 encoding=winansi alignment=justify
```

8.2.8 Steuerung des Algorithmus für den Zeilenumbruch

PDFlib verfügt über einen ausgefeilten Algorithmus für den Zeilenumbruch von Text. Tabelle 8.2 zeigt die Textflow-Optionen zur Steuerung des Zeilenumbruchs.

Umbruchregeln. Wenn ein Wort bzw. eine andere Zeichenfolge zwischen zwei Leerzeichen nicht vollständig in die Zeile passt, muss es in die nächste Zeile umbrochen werden. Dabei entscheidet der Umbruchalgorithmus, nach welchen Zeichen innerhalb einer Zeilenfolge ein Umbruch möglich ist.

Eine Formel wie *-12+235/8*45* wird zum Beispiel nie umbrochen, während der String *PDF-345+LIBRARY* nach dem Minuszeichen in die nächste Zeile umbrochen werden kann. Nach weichen Trennzeichen *(soft hyphen)*, sofern sie im Text vorhanden sind, kann ebenfalls ein Umbruch erfolgen.

Bei Klammern wird zwischen schließender und öffnender Klammer unterschieden, wobei auch Anführungszeichen wie Klammern behandelt werden. Nach einer öffnenden Klammer wie »(« oder einem öffnenden Anführungszeichen wird nicht umbrochen. Bei Anführungszeichen wird mittels Bilanzierung ermittelt, ob es sich um ein öffnendes oder schließendes Zeichen handelt.

Inline-Optionslisten stellen grundsätzlich keinen potentiellen Zeilenumbruch dar, damit man Optionen auch innerhalb eines Worts ändern kann. Befinden sich jedoch vor und nach der Optionsliste Leerzeichen, kann zu Beginn der Optionsliste ein Umbruch erfolgen. Wird vor der Optionsliste umbrochen und ist *alignment=justify*, werden

Tabelle 8.2 Optionen zur Steuerung des Zeilenumbruchs

Option	Erklärung
adjust-method	*(Schlüsselwort) Methode zur Berechnung der Zeilenumbrüche für Text, der nach Stauchen oder Dehnen der Wortabstände innerhalb der von* minspacing *und* maxspacing *gesetzten Grenzen nicht mehr vollständig in die Zeile passt. Standardwert:* auto
	auto *Versuche nacheinander die Methoden* shrink, spread, nofit, split
	clip *Wie* nofit *(siehe unten) mit dem Unterschied, dass lange Zeilen am rechten Rand der Fitbox (abzüglich der Option* rightindent*) abgeschnitten werden.*
	nofit *Das letzte Wort wird in die nächste Zeile gestellt, sofern die vorige Zeile dadurch nicht kürzer wird als der durch* nofitlimit *vorgegebene Prozentsatz. Der Blocksatz »flattert« dann an dieser Stelle ein wenig.*
	shrink *Passt ein Wort nicht mehr vollständig in die Zeile, wird versucht, den Text in dieser Zeile unter Berücksichtigung von* shrinklimit *so weit zu stauchen, bis das Wort doch noch hineinpasst. Falls der Text immer noch nicht passt, wird die Methode* nofit *angewandt.*
	split *Das letzte Wort wird nicht in die nächste Zeile gestellt, sondern zwangsweise getrennt. Bei Textschriften wird ein Trennzeichen eingefügt, bei Symbolschriften dagegen nicht.*
	spread *Das letzte Wort wird in die nächste Zeile gestellt und die verbleibende kurze Zeile durch Sperren des Textes auf Blocksatz gebracht, sofern die Option* spreadlimit *dies zulässt. Falls der Text immer noch nicht passt, wird die Methode* nofit *angewandt.*
advanced-linebreak	*(Boolean) Aktiviert den erweiterten Zeilenumbruch-Algorithmus, der für den Zeilenumbruch in komplexen Schriftsystemen erforderlich ist, die keine Leerzeichen zur Kennzeichnung von Wortgrenzen verwenden, z.B. Thai. Die Optionen* locale *und* script *werden berücksichtigt. Standardwert:* false
avoidbreak	*(Boolean) Bei* true *findet so lang kein Zeilenumbruch statt, bis* avoidbreak *wieder auf* false *gesetzt wird. Standardwert:* false
charclass	*(Liste von Paaren, wobei das erste Element eines Paares ein Schlüsselwort ist, und das zweite Element entweder ein Unichar oder eine Liste von Unichars) Die angegebenen Unichars werden durch das Schlüsselwort klassifiziert, um deren Zeilenumbruchverhalten festzulegen:*
	letter *verhält sich wie ein Buchstabe (z.B. a B)*
	punct *verhält sich wie ein Satzzeichen (z.B. + / ; :)*
	open *verhält sich wie eine öffnende Klammer (z.B. [ )*
	close *verhält sich wie eine schließende Klammer (z.B.])*
	default *setzt alle Zeichenklassen auf die internen PDFlib-Standardwerte zurück*
	Beispiel: `charclass={ close » open « letter {/ : =} punct & }`
hyphenchar	*(Unichar oder Schlüsselwort) Unicode-Wert des Zeichens, das ein weiches Trennzeichen beim Zeilenumbruch ersetzt. Beim Wert 0 bzw. dem Schlüsselwort* none *wird überhaupt kein Trennzeichen verwendet. Standardwert: U+00AD (SOFT HYPHEN) falls im Font enthalten, sonst U+002D (HYPHEN-MINUS)*
locale	*(Schlüsselwort) Bestimmt die Spracheinstellung für lokalisierte Zeilenumbruch-Methoden, wenn* advancedlinebreak=true*. Das Schlüsselwort besteht aus einer oder mehreren Komponenten, wobei die optionalen Komponenten durch einen Unterstrich '_' getrennt sind. (Die Syntax unterscheidet sich geringfügig von NLS / POSIX locale IDs):* ► *Ein erforderlicher Sprachcode aus zwei oder drei Kleinbuchstaben nach ISO 639-2 (siehe www.loc.gov/standards/iso639-2), z.B.* en*, (Englisch),* de *(Deutsch),* ja *(Japanisch). Dieser unterscheidet sich von der Option* language*.* ► *Eine optionaler vierbuchstabiger Schriftsystem-Code nach ISO 15924 (siehe www.unicode.org/iso15924/iso15924-codes.html), z.B.* Hira *(Hiragana),* Hebr *(Hebräisch),* Arab *(Arabisch),* Thai *(Thai).* ► *Ein optionaler Ländercode aus zwei Großbuchstaben nach ISO 3166 (siehe www.iso.org/iso/country_codes/iso_3166_code_lists), z. B.* DE *(Deutschland),* CH *(Schweiz),* GB *(Großbritannien)* *Mit dem Schlüsselwort* _none *wird keine Sprache eingestellt. Bei manchen Schriftsystemen, z.B. Thai, ist für den erweiterten Zeilenumbruch eine Spracheinstellung erforderlich: Standardwert:* _none *Beispiele: Thai,* de_DE, en_US, en_GB

Tabelle 8.2 Optionen zur Steuerung des Zeilenumbruchs

Option	Erklärung
maxspacing ***minspacing***	*(Float oder Prozentwert) Maximaler oder minimaler Abstand zwischen zwei Wörtern (in Benutzerkoordinaten oder als Prozentwert der Breite eines Leerzeichens); der jeweils berechnete Wortabstand wird zum Wert der Option* `wordspacing` *addiert. Standardwerte:* `minspacing=50%`, `maxspacing=500%`
nofitlimit	*(Float oder Prozentwert) Minimal erlaubte Länge einer Zeile bei der Methode* `nofit` *(in Benutzerkoordinaten oder als Prozentwert der Breite der Fitbox). Standardwert: 75%*
shrinklimit	*(Prozentwert) Untergrenze für das Stauchen des Textes bei der Methode* `shrink`*; der jeweils berechnete Stauchungsfaktor wird mit dem Wert der Option* `horizscaling` *multipliziert. Standardwert: 85%*
spreadlimit	*(Float oder Prozentwert) Obergrenze für den Abstand zwischen zwei Zeichen bei der Methode* `spread`*; der jeweils berechnete Zeichenabstand wird zum Wert der Option* `charspacing` *addiert. Standardwert: 0*

Leerzeichen vor der Optionsliste entfernt. Leerzeichen nach der Optionsliste bleiben erhalten und werden an den Anfang der nächsten Zeile positioniert.

Unterdrückung des Zeilenumbruchs. Mit der Option *charclass* können Sie verhindern, dass der Textflow nach bestimmten Zeichen in die nächste Zeile umbrochen wird. Folgende Option verhindert zum Beispiel einen Zeilenumbruch unmittelbar nach dem Zeichen /:

```
charclass={letter /}
```

Um zu verhindern, dass eine Zeichenfolge in die nächste Zeile umbrochen wird, können Sie sie in *avoidbreak…noavoidbreak* einschließen.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/avoid_linebreaking`.

Formatierung von CJK-Text in Textflows. Die Textflow-Implementierung ist auf CJK-Text vorbereitet und verarbeitet CJK-Zeichen korrekt als ideografische Zeichen gemäß Unicode. Folglich kann bei CJK-Text auch keine Trennung stattfinden. Zur besseren Formatierung von CJK-Text in Textflows sind folgende Optionen zu empfehlen; sie deaktivieren die Trennung von eingeschobenem lateinischem Text und erzeugen eine Ausgabe mit gleichmäßigen Abständen:

```
hyphenchar=none
alignment=justify
shrinklimit=100%
spreadlimit=100%
```

Vertikale Schreibrichtung wird in Textflows nicht unterstützt.

Blocksatz in schmaler Fitbox. Mit den Optionen zur Steuerung des Blocksatzes müssen Sie sich in der Regel um so mehr auseinandersetzen, je schmaler die Fitbox ist.

Abbildung 8.23 illustriert an einer schmalen Fitbox, an welchen Stellen die Blocksatzmethoden und -optionen besondere Wirkung zeigen. In Abbildung 8.23 sind die Einstellungen der drei Optionen an sich in Ordnung, lediglich *maxspacing* legt einen etwas großzügigen maximalen Wortabstand fest. Dies sollte man bei sehr schmaler Fitbox jedoch belassen, da sonst die hässliche (durch die Methode *split* verursachte) Zwangstrennung häufiger zur Anwendung kommt.

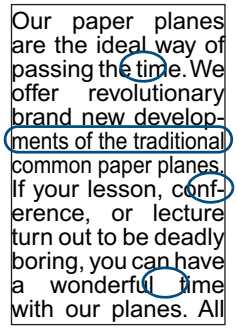


Abb. 8.23
Blocksatz mit Standardeinstellungen in schmaler Fitbox

Verkleinern des Wortabstands (Standardmethode, Option minspacing)

Stauchen der Zeile (Methode shrink, Option shrinklimit)

Zwangstrennung (Methode split)

Vergrößern des Wortabstands (Standardmethode, Option maxspacing)

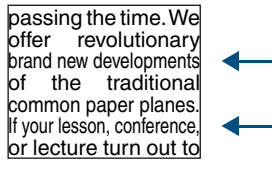
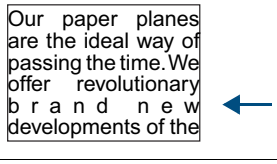
Ist die Fitbox so schmal, dass hin und wieder zwangsweise getrennt wird, sollten Sie entweder das Einfügen expliziter »weicher« Trennzeichen erwägen oder die Blocksatzoptionen verändern.

Blocksatzoption shrinklimit. Am unauffälligsten für das Auge ist es, die Option *shrinklimit* zu verkleinern, die festlegt, auf wieviel Prozent seiner normalen Breite Text durch die Methode *shrink* maximal gestaucht werden darf. Abbildung 8.24a zeigt, wie sich Zwangstrennungen durch Stauchung auf bis zu 50% vermeiden lassen.

Blocksatzoption spreadlimit. Das Sperren von Text, das mit der Methode *spread* bewerkstelligt und durch die Option *spreadlimit* zum Ändern des Zeichenabstands kontrolliert wird, ist ein weiteres Verfahren zur Beeinflussung des Zeilenumbruchs. Dieses optisch sehr auffällige Verfahren kommt aber nur selten zum Einsatz. Abbildung 8.24b zeigt einen der Deutlichkeit halber sehr auffälligen maximalen Zeichenabstand von 5 Einheiten.

Blocksatzoption nofitlimit. Mit der Option *nofitlimit* schließlich können Sie bestimmen, wie kurz der Text in einer Zeile minimal werden darf, wenn die Methode *nofit* zur Anwendung kommt. Eine Verkleinerung des Standardwertes von 75% ist bei sehr schmalen Zeilen einer Zwangstrennung vorzuziehen. Abbildung 8.24c zeigt die Ausgabe des Textes mit einer minimalen Textbreite von 50%.

Abb. 8.24 Blocksatz-Optionen bei einer schmalen Fitbox

	Generierte Ausgabe	***Optionsliste für PDF_fit_textflow( )***
a)		`alignment=justify shrinklimit=50%`
b)		`alignment=justify spreadlimit=5`

Generierte Ausgabe	*Optionsliste für PDF_fit_textflow( )*
c) ments of the traditional common paper planes. If your lesson, conference, or lecture turn out to be deadly boring, you can have ←	`alignment=justify nofitlimit=50`

8.2.9 Erweiterter Zeilenumbruch für spezielle Schriftsysteme

PDFlib verfügt zusätzlich über einen erweiterten Zeilenumbruch-Algorithmus. Dieser ist für manche Schriftsysteme erforderlich, verbessert aber auch bei einigen Kombinationen von Schriftsystem und Spracheinstellung, bei denen er eigentlich nicht benötigt wird, das Verhalten beim Zeilenumbruch. Er lässt sich mit der Option *advancedlinebreak* aktivieren. Da der Zeilenumbruch von der Sprache des Textes abhängt, berücksichtigt der erweiterte Zeilenumbruch-Algorithmus die Option *script* (siehe Tabelle 6.2) und die Option *locale* (siehe PDFlib-Referenz). Der erweiterte Zeilenumbruch bestimmt die richtigen Positionen für Zeilenumbrüche in folgenden Situationen:

- Bei Schriftsystemen, bei denen sich der Zeilenumbruch nicht an Leerzeichen im Text orientieren kann, z.B. Thai. Die folgende Textflow-Optionsliste aktiviert den erweiterten Zeilenumbruch-Algorithmus für Thai:

  ```
  <advancedlinebreak script=thai locale=tha>
  ```

- Bei Kombinationen von Schriftsystem und Spracheinstellung, die eine spezifische Behandlung bestimmter Satzzeichen erfordern, etwa die im Französischen als Anführungszeichen verwendeten Guillemet-Zeichen (z.B. « »). Die folgende Textflow-Optionsliste aktiviert den erweiterten Zeilenumbruch-Algorithmus für französischen Text. Als Ergebnis werden die ein Wort einschließenden Guillemet-Zeichen am Ende einer Zeile nicht vom Wort getrennt:

  ```
  <advancedlinebreak script=latn locale=fr>
  ```

Beachten Sie, dass sich die Textflow-Option *locale* von der Textoption *language* unterscheidet (siehe Tabelle 6.3): obwohl die Option *locale* mit den gleichen dreibuchstabigen Sprachcodes beginnen kann, kann sie zusätzlich ein oder zwei weitere Teilen enthalten. Diese werden bei PDFlib jedoch nur selten benötigt.

8.2.10 Umfließen von Pfaden und Bildern

Mit dem Feature *Umfließen* können beliebig begrenzte Bereiche mit Text gefüllt werden oder Text entlang eines Pfades platziert werden. Anhand von Matchboxen, Rechtecken, Polygonen, Kreisen, Kurven oder Pfadobjekten können Sie Bereiche definieren, die vom Textflow ausgespart bleiben. Wenn ein Bild einen integrierten Beschneidungspfad enthält, können Sie den Text um den Beschneidungspfad des Bildes herumfließen lassen.

Umfließen eines Bildes mit Matchbox. Im ersten Beispiel platzieren wir ein Bild innerhalb eines Textflows und lassen den Text um das Bild herumfließen. Das Bild wird zunächst geladen und an der gewünschten Position in die Box platziert. Um später über den Namen auf die Fitbox des Bildes zugreifen zu können, platzieren wir das Bild und definieren mit *matchbox={name=img margin=-5}* eine Matchbox namens *img* mit einem Rand von 5 Einheiten wie folgt:

Abb. 8.25
Umfließen eines Bildes mit Matchbox

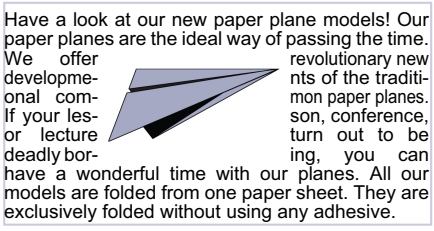
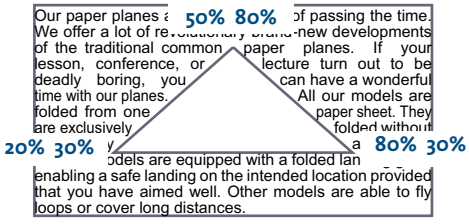


Abb. 8.26
Umfließen eines Dreiecks




```
p.fit_image(image, 50, 35,
        "boxsize={80 46} fitmethod=meet position=center matchbox={name=img margin=-5}");
```

Dann wird der Textflow hinzugefügt und mit der Option *wrap* und der Matchbox *img* des Bildes so platziert, dass er um die Matchbox herumfließt (siehe Abbildung 8.25).

```
p.fit_textflow(textflow, left_x, left_y, right_x, right_y,
        "wrap={usematchboxes={{img}}}");
```

Wenn Sie vor der Platzierung des Textes weitere Bilder unter Verwendung desselben Matchbox-Namens positionieren, fließt der Text um all diese Bilder herum.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/wrap_text_around_images`.

Umfließen von beliebigen Pfaden. Sie können Pfadobjekte erstellen (siehe Abschnitt 3.2.3, »Direkte Pfade und Pfadobjekte«, Seite 77) und diese als zu umfließende Bereiche verwenden. Mit dem folgenden Codefragment können Sie ein einfach geformtes Pfadobjekt (Kreis) erstellen und an die Option *wrap* von *PDF_fit_textflow( )* übergeben. Der Referenzpunkt für die Platzierung des Pfades ist als Prozentwert bezogen auf Breite und Höhe der Fitbox angegeben:

```
path = p.add_path_point(  -1,   0, 100, "move", "");
path = p.add_path_point(path, 200, 100, "control", "");
path = p.add_path_point(path,   0, 100, "circular", "");

/* Pfad sichtbar machen, wenn gewünscht */
p.draw_path(path, x, y, "stroke");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
        "wrap={paths={" +
            "{path=" + path + " refpoint={100% 50%} }" +
        "}}");

p.delete_path(path);
```

Mit der Option *inversefill* können Sie den Text innerhalb statt außerhalb entlang des Pfades verlaufen lassen (der Pfad dient also als Textcontainer, statt einen leeren Bereich im Textflow zu bewirken):

```
result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
        "wrap={inversefill paths={" +
```

```
            "{path=" + path + " refpoint={100% 50%} }" +
        "}}");
```

Umfließen eines Bildbeschneidungspfades. Bilder vom Typ TIFF und JPEG können einen integrierten Beschneidungspfad enthalten. Der Pfad muss mit einem Bildverarbeitungsprogramm erstellt worden sein und wird von PDFlib ausgewertet. Wenn im Bild ein Standard-Beschneidungspfad gefunden wird, wird dieser verwendet. Sie können aber auch einen beliebigen Pfad im Bild mit der Option *clippingpathname* von *PDF_load_image( )* angeben. Wenn das Bild mit einem Beschneidungspfad geladen wurde, können Sie diesen extrahieren und an die Option *wrap* von *PDF_fit_textflow( )* wie oben übergeben. Im Beispiel wird auch die Option *scale* zur Vergrößerung des importierten Beschneidungspfades übergeben:

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* Pfadobjekt aus Bildbeschneidungspfad erstellen */
path = (int) p.info_image(image, "clippingpath", "");
if (path == -1)
        throw new Exception("Error: clipping path not found!");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
        "wrap={paths={{path=" + path + " refpoint={50% 50%} scale=2}}}");

p.delete_path(path);
```

Platzieren und Umfließen eines Bildes. Während in den vorigen Abschnitten nur der Beschneidungspfad eines Bildes (aber nicht das Bild selbst) verwendet wurde, lassen Sie uns nun das Bild in die Fitbox des Textflows platzieren und den Text darum herumfließen. Um dies zu erreichen, müssen wir wieder die Fitbox mit der Option *clippingpathname* laden und sie mit *PDF_fit_image( )* auf der Seite platzieren. Um das richtige Pfadobjekt für das Umfließen mit Textflow zu erstellen, rufen wir *PDF_info_image( )* mit der gleichen Optionsliste wie *PDF_fit_image( )* auf. Zum Schluss müssen wir den Referenzpunkt (die Parameter *x/y* von *PDF_fit_image( ))* an die Unteroption *refpoint* der Unteroption *paths* der Option *wrap* übergeben:

```
image = p.load_image("auto", "image.tif", "clippingpathname={path 1}");

/* Bild mit einigen Einpassoptionen auf der Seite platzieren */
String imageoptlist = "scale=2";
p.fit_image(image, x, y, imageoptlist);

/* Pfadobjekt mit der gleichen Optionsliste aus Bildbeschneidungspfad erstellen */
path = (int) p.info_image(image, "clippingpath", imageoptlist);
if (path == -1)
        throw new Exception("Error: clipping path not found!");

result = p.fit_textflow(tf, llx1, lly1, urx1, ury1,
        "wrap={paths={{path=" + path + " refpoint={" + x + " " + y + "} }}}");

p.delete_path(path);
```

Sie können die gleiche *wrap*-Option in mehreren Aufrufen an *PDF_fit_textflow( )* übergeben. Die kann sinnvoll sein, wenn das platzierte Bild mehrere Textflow-Fitboxen überlappt, zum Beispiel bei mehrspaltigem Text.

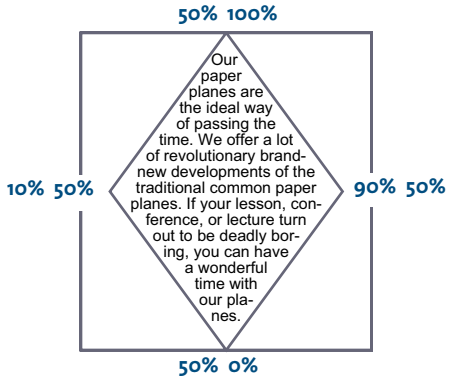


Abb. 8.27
Füllen eines rautenförmigen Bereichs mit Text

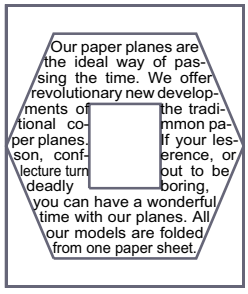


Abb. 8.28
Füllen von überlappenden Bereichen

Umfließen nicht rechteckiger Bereiche. Alternativ zum Erstellen eines Pfadobjekts als zu umfließendem Bereich, können Sie Pfadelemente direkt in Textflow-Optionen angeben.

Text kann nicht nur um einen rechteckigen, durch eine Matchbox vorgegebenen Bereich herumfließen, sondern um beliebige Polygone. Mit der folgenden Optionsliste können Sie zum Beispiel einen dreieckigen Bereich aussparen (siehe Abbildung 8.26):

```
wrap={ polygons={ {50% 80%   20% 30%   80% 30%   50% 80%} } }
```

Die Option *showborder=true* wurde zur Veranschaulichung der Ränder verwendet. Sie können in der Option *wrap* auch mehrere Polygone definieren. Mit folgender *wrap*-Option umfließt der Text zwei dreieckige Bereiche:

```
wrap={ polygons={ {50% 80%   20% 30%   80% 30%   50% 80%}
                  {20% 90%   10% 70%   30% 70%   20% 90%} } }
```

Statt der Prozentangaben, die relative Koordinaten innerhalb der Fitbox darstellen, können Sie auch absolute Koordinaten auf der Seite verwenden.

Hinweis *Wenn Sie Bereiche mit Segmenten verwenden, die weder horizontal noch vertikal ausgerichtet sind, sollten Sie* `fixedleading=true` *setzen.*

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/wrap_text_around_polygons`.

Füllen nicht rechteckiger Bereiche. Mit dem Feature *Umfließen* können beliebig definierte Bereiche genutzt werden, um den Inhalt eines Textflows darin zu platzieren. Dies erreichen Sie mit der Unteroption *addfitbox* der Option *wrap*, die zuerst die Fitbox selbst ausspart, und stattdessen alle mit nachfolgenden Unteroptionen definierten Bereiche mit Text füllt. Folgende Optionsliste füllt den Text in einen rautenförmigen Bereich, wobei die Koordinaten in Prozentwerten bezogen auf die Fitbox angegeben werden (siehe Abbildung 8.27):

```
wrap={ addfitbox polygons={ {50% 100%   10% 50%   50% 0%   90% 50%   50% 100%} } }
```

Die Option *showborder=true* wurde wieder zur Veranschaulichung der Ränder verwendet. Wenn Sie die Option *addfitbox* weglassen, bleibt der rautenförmige Bereich leer und der Text fließt um ihn herum.

Füllen überlappender Bereiche. Im nächsten Beispiel füllen wir zwei überlappende Bereiche, und zwar ein Sechseck, in dem ein Rechteck liegt. Mit der Option *addfitbox* sparen wir die Fitbox selbst aus. Die nachfolgend mit der Unteroption *polygons* definierten Polygone werden mit Text gefüllt, wobei überlappende Bereiche ebenfalls ausgespart bleiben (siehe Abbildung 8.28):

```
wrap={ addfitbox polygons=
       { {20% 10%   80% 10%   100% 50%   80% 90%   20% 90%   0% 50%   20% 10%}
         {35% 35%   65% 35%   65% 65%   35% 65%   35% 35%} } }
```

Wenn Sie die Option *addfitbox* weglassen, erhalten Sie das umgekehrte Ergebnis: Der vorher gefüllte Bereich bleibt leer, und stattdessen werden die vorher leeren Bereiche mit Text gefüllt.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/fill_polygons_with_text`.

8.3 Tabellenformatierung

Der Tabellenformatierer dient der automatischen Formatierung komplexer Tabellen. Tabellenzellen können ein- oder mehrzeiligen Text, Bilder, SVG-Grafiken oder PDF-Seiten enthalten. Tabellen sind nicht auf eine einzelne Fitbox beschränkt, sondern können sich über mehrere Seiten erstrecken.

Cookbook *Codebeispiele zum Einsatz von Tabellen finden Sie in der Kategorie* `tables` *des PDFlib Cookbook.*

Generelle Aspekte einer Tabelle. Zur Beschreibung des Tabellenformatierers werden folgende Konzepte und Begriffe herangezogen (siehe Abbildung 8.29):

- Eine *Tabelle* ist ein virtuelles Objekt mit rechteckigem Umriss. Sie besteht aus horizontalen *Zeilen* und vertikalen *Spalten*.
- Eine *einfache Zelle* besteht aus einem rechteckigen Bereich innerhalb einer Tabelle, der sich als Schnittmenge einer Zeile und einer Spalte definiert. Es gibt aber auch Zellen, die sich über mehrere Zeilen und/oder Spalten erstrecken. Der Begriff *Zelle* wird sowohl für eine einfache Zelle als auch für eine Zelle verwendet, die sich über mehrere Zeilen und/oder Spalten erstreckt.
- Eine Tabelle kann vollständig in eine Fitbox passen, oder es sind mehrere Fitboxen erforderlich. Eine *Tabelleninstanz* bezeichnet alle Zeilen einer Tabelle, die in einer Fitbox Platz finden. Jeder Aufruf von *PDF_fit_table( )* platziert eine *Tabelleninstanz* in einer Fitbox (siehe Abschnitt 8.3.5, »Tabelleninstanzen«, Seite 256).
- Die Kopfzeilen *(header)* bzw. Fußzeilen *(footer)* bestehen aus einer oder mehreren Zeilen am Anfang bzw. Ende der Tabelle, die am Anfang bzw. Ende jeder Tabelleninstanz wiederholt werden. Zeilen, die weder Kopf- noch Fußzeilen sind, heißen Body-Zeilen.
- Eine optionale Beschriftung (nicht dargestellt in Abbildung 8.29) ist ein Hilfselement zur Platzierung einer Tabellenbeschreibung. Sie kann auf jeder Seite der Tabelle platziert werden.

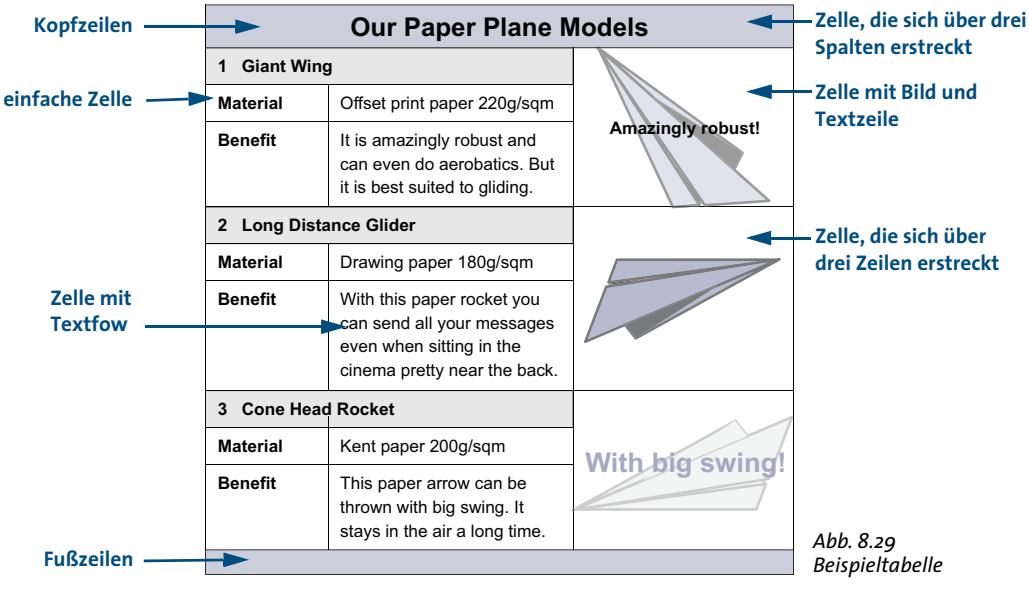


Abb. 8.29 Beispieltabelle

In unserem Tabellenbeispiel werden alle Konzepte erläutert, die zur Erstellung der Tabelle in Abbildung 8.29 erforderlich sind. Eine ausführliche Beschreibung aller Optionen zur Tabellenformatierung finden Sie in der PDFlib-Referenz. Beim Anlegen einer Tabelle definieren Sie mit *PDF_add_table_cell( )* zunächst den Inhalt und das Aussehen der Tabellenzelle. Dann platzieren Sie die Tabelle mit einem oder mehreren Aufrufen von *PDF_fit_table( )*.

Dabei kann die Größe der Fitbox sowie die Lineatur und Schattierung der Tabellenzeilen und -spalten festgelegt werden. Mit der Matchbox-Funktion können spezielle Merkmale wie die Schattierung einzelner Zellen vorgenommen werden (weitere Informationen finden Sie in Abschnitt 8.4, »Matchboxen«, Seite 262).

In diesem Abschnitt werden die wichtigsten Optionen zur Definition der Tabellenzellen und Platzierung der Tabelle erläutert. In allen Beispielen werden nur die jeweils relevanten Aufrufe von *PDF_add_table_cell( )* und *PDF_fit_table( )* beschrieben. Es wird vorausgesetzt, dass der gewünschte Font bereits geladen wurde.

Hinweis *Tabellen werden unabhängig vom aktuellen Grafikzustand verarbeitet. Tabellenzellen können im Geltungsbereich* `document` *definiert werden, die eigentliche Platzierung der Tabelle muss aber im Geltungsbereich* `page` *erfolgen (oder auch* `pattern`, `template` *oder* `glyph`*).*

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `tables/starter_table`.

Analyse tabellenbezogener Formatierungsprobleme. Abhängig von der Anzahl der Zellen und den Formatierungsoptionen für die Tabelle kann es vorkommen, dass die Ergebnisse des Tabellenformatierers von PDFlib manchmal nicht Ihren Erwartungen entsprechen. In fast allen Fällen kann dies mit geeigneten Optionen behoben werden. Allerdings kann es schwierig sein, die betreffenden Zellen oder Gruppe von Zellen mit den falschen Optionen zu identifizieren. Um die Fehleranalyse von tabellenbezogenen Formatierungsproblemen zu erleichtern, bietet PDFlib die folgenden Optionen in *PDF_fit_table( ):*

- Die Option *showcells* macht die Grenze jeder inneren Zellenbox sichtbar. Wird die Funktion im Geltungsbereich *page* aufgerufen und ist der PDF/A-Modus nicht aktiviert, wird eine Anmerkung mit Informationen zum Zelleninhalt in der Mitte jeder Tabellenzelle platziert.
- Ist die Option *debugshow* auf *true* gesetzt, werden alle Fehler für Tabellen, die zu hoch bzw. zu breit sind, oder zu kleine Zellen haben, unterdrückt und stattdessen protokolliert. Die daraus resultierende Tabelleninstanz wird als Debugging-Hilfe erstellt, obwohl die Tabelle beschädigt ist.
- Ist die Option *showgrid* auf *true* gesetzt, werden die vertikalen und horizontalen Begrenzungen aller Spalten und Zeilen gezeichnet, das heißt, das zugrundeliegende Tabellenraster wird sichtbar gemacht.

8.3.1 Platzieren einer einfachen Tabelle

Bevor wir genauer auf die verschiedenen Tabelleneigenschaften eingehen, zeigen wir ein einfaches Beispiel zur Erstellung einer Tabelle. Die Tabelle enthält sechs Zellen, die in drei Zeilen und zwei Spalten angeordnet sind. Vier Zellen enthalten einzeiligen Text, und eine Zelle enthält einen mehrzeiligen Textflow. Alle Zelleninhalte sind horizontal linksbündig ausgerichtet und vertikal zentriert mit einem Rand von 4 Punkt.

Beim Anlegen der Tabelle erstellen wir zunächst die Optionsliste für die Zellen mit Textzeilen, indem wir in der Unteroptionsliste *fittextline* die Optionen *font* und *fontsize*

sowie die Position *{left center}* festlegen. Außerdem legen wir Zellenränder von 4 Punkt fest. Dann fügen wir die Zellen mit Textzeile nacheinander in die jeweilige Spalte und Zeile ein, wobei der Text direkt im Aufruf von *PDF_add_table_cell( )* übergeben wird.

Im nächsten Schritt erzeugen wir einen Textflow, erstellen die Optionsliste für Textflow-Zellen mithilfe des Textflow-Handles und fügen diese Zelle zur Tabelle hinzu.

Als letztes platzieren wir die Tabelle mit *PDF_fit_table( ),* wobei wir den Tabellenrahmen und die Zellenränder mit schwarzer Lineatur versehen. Da wir keine Spaltenbreiten definiert haben, werden diese automatisch aus den übergebenen Textzeilen einschließlich der Ränder berechnet.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `tables/vertical_text_alignment`.

Das folgende Codefragment erzeugt eine einfache Tabelle. Abbildung 8.30a zeigt das Ergebnis.

```
/* Text für eine Tabellenzelle mit mehrzeiligem Textflow */
String tf_text = "It is amazingly robust and can even do aerobatics. " +
                 "But it is best suited to gliding.";

/* Spaltenbreite der ersten und zweiten Spalte definieren */
int c1 = 80, c2 = 120;

/* Linke untere und rechte obere Ecke der Tabelleninstanz definieren (Fitbox) */
double llx=100, lly=500, urx=300, ury=600;

/* Verhalten bei Fehlern*/
p.set_option("errorpolicy=exception");

/* Font laden */
font = p.load_font("Helvetica", "unicode", "");
if (font == -1)
        throw new Exception("Error: " + p.get_errmsg());

/* Optionsliste für Textline-Zellen in der ersten Spalte definieren */
optlist = "fittextline={position={left center} font=" + font + " fontsize=8} margin=4" +
          colwidth=" + c1;

/* Zelle mit Textzeile in Spalte 1 Zeile 1 einfügen */
tbl = p.add_table_cell(tbl, 1, 1, "Our Paper Planes", optlist);

/* Zelle mit Textzeile in Spalte 1 Zeile 2 einfügen */
tbl = p.add_table_cell(tbl, 1, 2, "Material", optlist);

/* Zelle mit Textzeile in Spalte 1 Zeile 3 einfügen */
tbl = p.add_table_cell(tbl, 1, 3, "Benefit", optlist);

/* Optionsliste für Textline-Zelle in der zweiten Spalte definieren */
optlist = "fittextline={position={left center} font=" + font + " fontsize=8} " +
          "colwidth=" + c2 + " margin=4";

/* Zelle mit Textzeile in Spalte 2 Zeile 2 einfügen */
tbl = p.add_table_cell(tbl, 2, 2, "Offset print paper 220g/sqm", optlist);

/* Textflow einfügen */
optlist = "font=" + tont + " fontsize=8 leading=110%";
tf = p.add_textflow(-1, tf_text, optlist);
```

```
/* Mit erhaltenem Handle Optionsliste für Zelle mit Textflow definieren */
optlist = "textflow=" + tf + " margin=4 colwidth=" + c2";

/* Zelle mit Textflow in Spalte 2 Zeile 3 einfügen */
tbl = p.add_table_cell(tbl, 2, 3, "", optlist);

p.begin_page_ext(0, 0, "width=200 height=100");

/* Optionsliste zur Tabellenplatzierung mit Rahmen- und Zellenlineatur definieren */
optlist = "stroke={{line=frame linewidth=0.3} {line=other linewidth=0.3}}";

/* Tabelleninstanz platzieren */
result = p.fit_table(tbl, llx, lly, urx, ury, optlist);

/* Ergebnis prüfen; "_stop" bedeutet "alles okay" */
if (!result.equals("_stop")) {
        if (result.equals( "_error"))
            throw new Exception("Error: " + p.get_errmsg());
        else {
            /* Jeder andere Rückgabewert erfordert eigenen Code */
        }
}
p.end_page_ext("");

/* Damit werden auch in der Tabelle verwendete Textflow-Handles gelöscht */
p.delete_table(tbl, "");
```

Exakte vertikale Ausrichtung von Zelleninhalten. Sobald wir verschiedene Inhaltstypen in Tabellenzellen vertikal zentrieren, werden sie mit uneinheitlichem Abstand zum Rand positioniert. Die vier Textzeilen in Abbildung 8.30a wurden mit folgender Optionsliste platziert:

```
optlist = "fittextline={position={left center} font=" + font +
          " fontsize=8} colwidth=80 margin=4";
```

Die Zelle mit Textflow wurde ohne besondere Optionen eingefügt. Da die Textzeilen vertikal zentriert wurden, rutscht die Zeile *Benefit* je nach Höhe des Textflows nach unten.

Abb. 8.30 Ausrichtung von Textzeilen und Textflows in Tabellenzellen

Generierte Ausgabe

a)

Our Paper Planes	
Material	Offset print paper 220g/sqm
Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.

Generierte Ausgabe

b)

Our Paper Planes	
Material	Offset print paper 220g/sqm
Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.

Wie in Abbildung 8.30b gezeigt, sollen alle Zelleninhalte denselben *vertikalen* Abstand von den Zellenrändern besitzen, unabhängig davon, ob es sich um einen Textflow oder eine Textzeile handelt. Um dies zu bewerkstelligen, stellen wir erst einmal die Optionsliste für die Textzeilen zusammen. Wir definieren eine feste Zeilenhöhe von 14 Punkt, und die Positionierung erfolgt links oben mit einem Randabstand von 4 Punkt.

Die Option *fontsize=8*, die wir früher übergeben haben, bezeichnet nicht die genaue Buchstabenhöhe, sondern enthält zusätzlichen Abstand nach oben und unten. Exakt definiert wird die Höhe eines Großbuchstabens durch den *capheight*-Wert des Fonts. Aus diesem Grund verwenden wir *fontsize={capheight=6}*, was in etwa einer Fontgröße von 8 Punkt entspricht und (zusammen mit *margin=4)* eine Gesamthöhe von 14 Punkt ergibt und somit der Option *rowheight* entspricht. Die vollständige Optionsliste von *PDF_add_table_cell( )* für die Zellen mit Textzeilen lautet wie folgt:

```
/* Optionsliste zum Einfügen von Zellen mit Textzeilen */
optlist = "fittextline={position={left top} font=" + font +
          " fontsize={capheight=6}} rowheight=14 colwidth=80 margin=4";
```

Zum Hinzufügen des Textflows verwenden wir *fontsize={capheight=6}*, was in etwa einer Fontgröße von 8 Punkt entspricht und (mit *margin=4)* eine Gesamthöhe von 14 Punkt ergibt, so wie für obige Textzeilen.

```
/* Optionsliste zum Hinzufügen des Textflows */
optlist = "font=" + font + " fontsize={capheight=6} leading=110%";
```

Darüber hinaus soll die Grundlinie des Textes *Benefit* an der ersten Zeile des Textflows ausgerichtet sein. Gleichzeitig soll der Text *Benefit* aber denselben Abstand vom oberen Zellenrand besitzen wie der Text *Material*. Um sicherzustellen, dass der Textflow ohne jeden Abstand vom oberen Zellenrand eingefügt wird, verwenden wir die Option *fittextflow={firstlinedist=capheight}*. Dann addieren wir einen Abstand von 4 Punkt, genauso wie für die Textzeilen:

```
/* Optionsliste für die Zelle mit Textflow */
optlist = "textflow=" + tf + " fittextflow={firstlinedist=capheight} "
          "colwidth=120 margin=4";
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `tables/vertical_text_alignment`.

8.3.2 Inhalt von Tabellenzellen

Beim Einfügen einer Tabellenzelle mit *PDF_add_table_cell( )* können Sie verschiedene Arten von Inhalten festlegen. Tabellenzellen können einen oder mehrere Inhaltstypen

gleichzeitig enthalten. Zusätzlich können sie mit Lineatur und Schattierung versehen werden, sowie Matchboxen zur Platzierung von zusätzlichem Inhalt in Tabellenzellen.

Die Zellen der *Paper Planes* Tabelle enthalten beispielsweise die in Abbildung 8.31 skizzierten Inhalte.

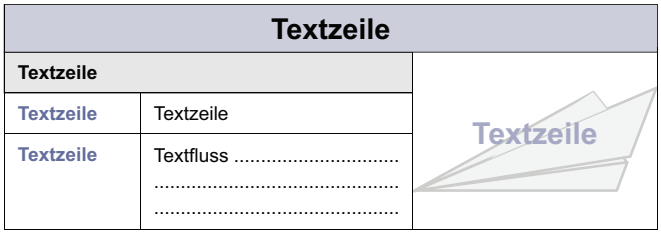


Abb. 8.31 Inhalt von Tabellenzellen

Einzeiliger Text. Der Text wird im Parameter *text* von *PDF_add_table_cell( )* übergeben. In der Option *fittextline* können alle Formatierungsoptionen von *PDF_fit_textline( )* verwendet werden. Zur Platzierung wird standardmäßig *fitmethod=nofit* verwendet. Passt der Text nicht vollständig in die Zelle, wird die Zelle verbreitert. Um dies zu vermeiden, kann der Text mit *fitmethod=auto* unter Berücksichtigung der Option *shrinklimit* gestaucht werden. Wurde keine Zeilenhöhe *(row height)* angegeben, so wird diese automatisch als doppelte Fontgröße berechnet (genauer: die doppelte *boxheight*, die den Standardwert *{capheight none}* hat, wenn nicht anders angegeben). Dies gilt ebenfalls für die Zeilenbreite bei gedrehtem Text.

Mehrzeiliger Text mit Textflow. Vor dem Aufruf von *PDF_add_table_cell( )* muss der Textflow außerhalb der Tabellenfunktionen vorbereitet und mit *PDF_create_textflow( )* oder *PDF_add_textflow( )* erzeugt worden sein. Das Textflow-Handle wird in der Option *textflow* übergeben. In der Option *fittextflow* können alle Formatierungsoptionen von *PDF_fit_textflow( )* verwendet werden.

Zur Platzierung wird standardmäßig *fitmethod=clip* verwendet. Dabei wird zunächst versucht, den Text vollständig in der Zelle zu platzieren. Ist die Zelle zu klein, wird ihre Höhe vergrößert. Passt der Text immer noch nicht vollständig in die Zelle, so wird er am unteren Rand beschnitten. Um dies zu vermeiden, kann der Text mit *fitmethod=auto* unter Berücksichtigung der Option *minfontsize* gestaucht werden.

Ist die Zelle zu schmal, kann es passieren, dass der Textflow innerhalb von Wörtern an unerwünschter Position umbricht. Ist die Option *checkwordsplitting* auf *true* gesetzt, wird die Zelle solange verbreitert, bis kein unerwünschter Wortumbruch mehr erfolgt.

Bilder und Templates. Vor dem Aufruf von *PDF_add_table_cell( )* müssen Bilder mit *PDF_load_image( )* geladen und Templates mit *PDF_begin_template_ext( )* angelegt worden sein. Das Bild- oder Template-Handle wird in der Option *image* übergeben. In der Option *fitimage* können alle Formatierungsoptionen von *PDF_fit_image( )* verwendet werden. Zur Platzierung wird standardmäßig *fitmethod=meet* verwendet. Dabei wird das Bild/Template unter Beibehaltung seiner Proportionen vollständig in der Zelle platziert. Die Zellengröße wird nicht an die Größe des Bildes/Templates angepasst.

Vektorgrafik. Vor dem Aufruf von *PDF_add_table_cell( )* müssen Grafiken mit *PDF_load_graphics( )* geladen werden. Das Grafik-Handle wird in der Option *graphics* übergeben. In der Option *fitgraphics* können alle Formatierungsoptionen von *PDF_fit_graphics( )*

verwendet werden. Zur Platzierung wird standardmäßig *fitmethod=meet* verwendet. Dabei wird die Grafik unter Beibehaltung ihrer Proportionen vollständig in der Zelle platziert. Die Zellengröße wird nicht an die Größe der Grafik angepasst.

Seiten aus einem importierten PDF-Dokument. Vor dem Aufruf von *PDF_add_table_cell( )* muss die PDI-Seite mit *PDF_open_pdi_page( )* geöffnet worden sein. Das PDI-Seiten-Handle wird in der Option *pdipage* übergeben. In der Option *fitpdipage* können alle Formatierungsoptionen von *PDF_fit_pdi_page( )* verwendet werden. Zur Platzierung wird standardmäßig *fitmethod=meet* verwendet. Dabei wird die PDI-Seite unter Beibehaltung ihrer Proportionen vollständig in der Zelle platziert. Die Zellengröße wird nicht an die Größe der PDI-Seite angepasst.

Pfadobjekte. Vor dem Aufruf von *PDF_add_table_cell( )* müssen Pfadobjekte mit *PDF_add_path_point( )* erzeugt worden sein. Das Pfad-Handle wird in der Option *path* übergeben. In der Option *fitpath* können alle Formatierungsoptionen von *PDF_ PDF_draw_path( )* verwendet werden. Die Boundingbox des Pfades wird in der Tabellenzelle platziert. Als Referenzpunkt zur Platzierung des Pfades wird die untere linke Ecke der inneren Zellenbox verwendet.

Annotationen. Annotationen in Tabellenzellen können mit der Option *annotationtype* von *PDF_add_table_cell( )* erstellt werden. Die Option entspricht dem Parameter *type* von *PDF_create_annotation( )* (diese Funktion muss aber nicht aufgerufen werden). In der Option *fitannotation* können alle Optionen von *PDF_create_annotation( )* verwendet werden. Die Zellenbox wird als Annotationsrechteck verwendet.

Formularfelder. Formularfelder in Tabellenzellen können mit den Optionen *fieldname* und *fieldtype* von *PDF_add_table_cell( )* erstellt werden. Diese Optionen entsprechen den Parametern *name* und *type* von *PDF_create_field( )* (diese Funktion muss aber nicht aufgerufen werden). In der Option *fitfield* können alle Optionen von *PDF_create_field( )* verwendet werden. Die Zellenbox wird als Feldrechteck verwendet.

Positionierung von Zelleninhalten. Standardmäßig wird der Zelleninhalt in Bezug auf die *Zellenbox* positioniert. Mit den *margin*-Optionen von *PDF_add_table_cell( )* kann ein Abstand zu den Zellenrändern definiert werden. Das daraus resultierende Rechteck wird *innere Zellenbox* genannt. Wird ein Abstand definiert, wird der Zelleninhalt bezüglich der inneren Zellenbox positioniert (siehe Abbildung 8.32). Ist kein Abstand definiert, entspricht die innere Zellenbox der Zellenbox.

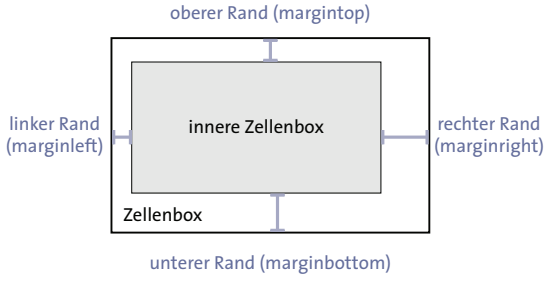


Abb. 8.32
Platzierung von Inhalten in der inneren Zellenbox

Zur Platzierung der Zelleninhalte können je nach Inhaltstyp weitere Optionen genutzt werden. Diese werden in Abschnitt 8.3.4, »Tabelle mit gemischtem Inhalt«, Seite 253, beschrieben.

8.3.3 Tabellen- und Spaltenbreite

Beim Hinzufügen einer Zelle zur Tabelle definieren Sie die Anzahl der Spalten und/oder Zeilen, über die sich die Zelle erstreckt, mit den Optionen *colspan* und *rowspan*. Standardmäßig erstreckt sich eine Zelle über eine Spalte und eine Zeile. Beim Hinzufügen einer Zelle erhöht sich die Gesamtzahl der Spalten und Zeilen in der Tabelle implizit um die entsprechenden Werte. Abbildung 8.33 zeigt eine Beispieltabelle mit drei Spalten und vier Zeilen.

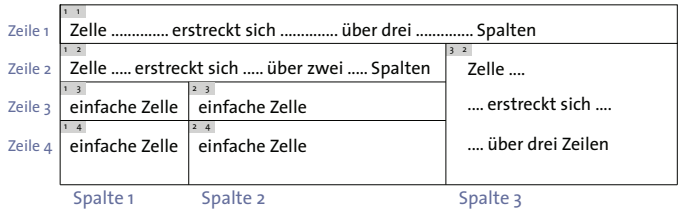


Abb. 8.33
Einfache Zellen und Zellen, die sich über mehrere Zeilen oder Spalten erstrecken

Mit der Option *colwidth* können Sie die Breite der ersten Spalte, über die sich die Zelle erstreckt, explizit angeben. Wenn Sie für jede Zelle die Breite der ersten Spalte übergeben, summieren sich diese Werte implizit zur Gesamtbreite der Tabelle. Abbildung 8.34 zeigt ein Beispiel.

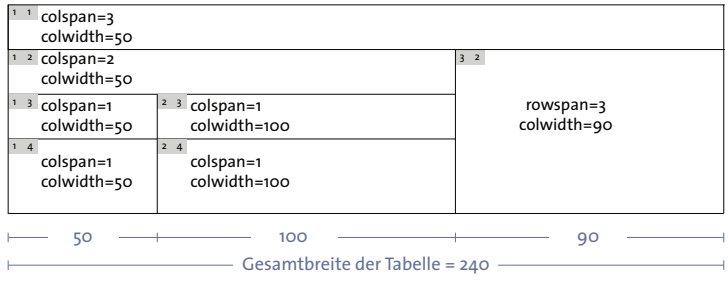


Abb. 8.34
Spaltenbreiten summieren sich zur Gesamtbreite der Tabelle

Alternativ dazu können Sie die Spaltenbreiten prozentual angeben. Die Prozentangaben beziehen sich in diesem Fall auf die Breite der Tabellen-Fitbox. Wird eine Spaltenbreite prozentual übergeben, so müssen alle Breiten Prozentangaben sein.

Werden Spalten mit der Option *colscalegroup* für *PDF_add_table_cell( )* zu einer Spaltenskalierungsgruppe zusammengefasst, so werden ihre Breiten vereinheitlicht und an die breiteste Spalte in der Gruppe angepasst (siehe Abbildung 8.35).

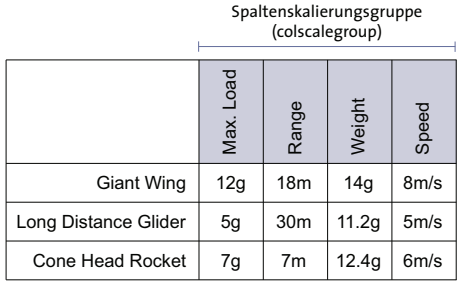


Abb. 8.35
Die letzten vier Zellen in der ersten Zeile befinden sich in derselben Spaltenskalierungsgruppe. Sie erhalten eine einheitliche Breite.

Werden absolute Koordinaten (und keine Prozentangaben) verwendet und gibt es Zellen ohne definierte Spaltenbreite, werden die fehlenden Breiten wie folgt berechnet: Zunächst wird die Breite von allen Zellen mit Textzeilen anhand deren Spalten- oder Textbreite (bzw. Texthöhe bei gedrehtem Text) berechnet. Die verbleibende Tabellenbreite wird dann gleichmäßig zwischen den noch fehlenden Spaltenbreiten aufgeteilt.

8.3.4 Tabelle mit gemischtem Inhalt

In den folgenden Abschnitten erzeugen wir schrittweise die Beispieltabelle aus Abbildung 8.36, die verschiedenartige Inhalte enthält.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `tables/mixed_table_contents`.

Zur Vorbereitung müssen wir zwei Fonts laden. Wir definieren die Tabellengröße über die Koordinaten der unteren linken und oberen rechten Ecke und bestimmen die Breite der drei Tabellenspalten. Dann beginnen wir eine neue Seite im A4-Format:

```
double llx = 100, lly = 500, urx = 360, ury = 600; // Tabellenkoordinaten

int c1 = 50, c2 = 120, c3 = 90; // Breite der drei Tabellenzellen

boldfont = p.load_font("Helvetica-Bold", "unicode", "");
normalfont = p.load_font("Helvetica", "unicode", "");

p.begin_page_ext(0, 0, "width=a4.width height=a4.height");
```

Schritt 1: Erste Zelle einfügen. Wir beginnen mit der ersten Zelle unserer Tabelle. Die Zelle wird in der ersten Spalte der ersten Zeile platziert und erstreckt sich über drei Spalten. Die erste Spalte hat eine Breite von 50 Punkt. Die Textzeile wird vertikal und horizontal zentriert, mit einem Abstand von 4 Punkt zu allen Rändern. Das folgende Codefragment zeigt das Einfügen der ersten Zelle:

```
optlist = "fittextline={font=" + boldfont + " fontsize=12 position=center} " +
          "margin=4 colspan=3 colwidth=" + c1;

tbl = p.add_table_cell(tbl, 1, 1, "Our Paper Plane Models", optlist);
```

Schritt 2: Zelle einfügen, die sich über zwei Spalten erstreckt. Im nächsten Schritt fügen wir die Zelle mit der Textzeile *1 Giant Wing* hinzu. Sie wird in der ersten Spalte der

zweiten Zeile platziert und erstreckt sich über zwei Spalten. Die erste Spalte hat eine Breite von 50 Punkt. Die Zeilenhöhe beträgt 14 Punkt. Die Textzeile wird links oben positioniert, mit einem Abstand von 4 Punkt zu allen Rändern. Mit *fontsize={capheight=6}* erzielen wir eine einheitliche vertikale Ausrichtung des Textes (siehe Abschnitt »Exakte vertikale Ausrichtung von Zelleninhalten«, Seite 248).

Die Zelle *Giant Wing* umfasst keine ganze Zeile, sondern nur zwei von drei Spalten. Deshalb kann sie nicht einfach mit den Optionen zur Zeilenschattierung mit Farbe gefüllt werden. Wir nutzen stattdessen die Matchbox-Funktion und füllen damit das von der Zelle bezeichnete Rechteck mit grauer Hintergrundfarbe. Eine ausführliche Beschreibung der Matchbox-Funktion finden Sie in Abschnitt 8.4, »Matchboxen«, Seite 262. Das folgende Codefragment zeigt das Hinzufügen der Zelle *Giant Wing:*

```
optlist = "fittextline={position={left top} font=" + boldfont +
          " fontsize={capheight=6}} rowheight=14 colwidth=" + c1 +
          " margin=4 colspan=2 matchbox={fillcolor={gray .92}}";

tbl = p.add_table_cell(tbl, 1, 2, "1  Giant Wing", optlist);
```

Abb. 8.36 Schrittweises Einfügen von Tabellenzellen mit verschiedenen Inhalten

Generierte Tabelle	*Generierungsschritte*

Our Paper Plane Models

1 Giant Wing	
Material	Offset print paper 220g/sqm
Benefit	It is amazingly robust and can even do aerobatics. But it is best suited to gliding.

1: Zelle über drei Spalten einfügen
2: Zelle über zwei Spalten einfügen
3: Drei Zellen mit Textzeilen einfügen
4: Zelle mit Textflow einfügen
5: Zelle mit Bild und Textzeile einfügen
6: Tabelle platzieren

Schritt 3: Drei weitere Zellen mit Textzeilen einfügen. Das folgende Codefragment fügt die Zellen *Material*, *Benefit* und *Offset print paper...* ein. Die Zelle *Offset print paper...* beginnt in der zweiten Spalte mit einer Spaltenbreite von 120 Punkt. Der Zelleninhalt wird links oben positioniert, mit einem Abstand von 4 Punkt von allen Rändern.

```
optlist = "fittextline={position={left top} font=" + normalfont +
          " fontsize={capheight=6}} rowheight=14 colwidth=" + c1 + " margin=4";

tbl = p.add_table_cell(tbl, 1, 3, "Material", optlist);
tbl = p.add_table_cell(tbl, 1, 4, "Benefit", optlist);

optlist = "fittextline={position={left top} font=" + normalfont +
          " fontsize={capheight=6}} rowheight=14 colwidth=" + c2 + " margin=4";

tbl = p.add_table_cell(tbl, 2, 3, "Offset print paper 220g/sqm", optlist);
```

Schritt 4: Zelle mit Textflow einfügen. Das folgende Codefragment fügt eine Zelle mit dem Textflow *It is amazingly...* ein. Bevor wir eine Zelle mit Textflow hinzufügen können, müssen wir zuerst den Textflow erstellen. Wir verwenden *fontsize={capheight=6}*, was in etwa einer Fontgröße von 8 Punkt entspricht und (zusammen mit *margin=4)* eine Gesamthöhe von 14 Punkt ergibt, so wie für obige Textzeilen.

```
tftext = "It is amazingly robust and can even do aerobatics. " +
          "But it is best suited to gliding.";
```

```
optlist = "font=" + normalfont + " fontsize={capheight=6} leading=110%";

tf = p.add_textflow(-1, tftext, optlist);
```

Mit dem zurückgegebenen Textflow-Handle fügen wir die Tabellenzelle hinzu. Die erste Zeile des Textflows soll an der Grundlinie der Textzeile *Benefit* ausgerichtet sein. Gleichzeitig soll der Text *Benefit* aber denselben Abstand vom oberen Zellenrand besitzen wie der Text *Material*. Um sicherzustellen, dass der Textflow ohne jeden Abstand vom oberen Zellenrand eingefügt wird, verwenden wir die Option *fittextflow={firstlinedist=capheight}*. Dann addieren wir einen Abstand von 4 Punkt, genauso wie für die Textzeilen:

```
optlist = "textflow=" + tf + " fittextflow={firstlinedist=capheight} " +
          "colwidth=" + c2 + " margin=4";

tbl = p.add_table_cell(tbl, 2, 4, "", optlist);
```

Schritt 5: Zelle mit Bild und Textzeile einfügen. Im fünften Schritt fügen wir eine Zelle ein, die das Bild des Papierfliegers Giant Wing sowie die Textzeile *Amazingly robust!* enthält. Die Zelle beginnt in der dritten Spalte der zweiten Zeile und erstreckt sich über drei Zeilen. Die Spaltenbreite beträgt 90 Punkt. Der Abstand der Zelle zu den Rändern ist auf 4 Punkt gesetzt. Für eine erste Variante platzieren wir ein TIFF-Bild in der Zelle:

```
image = p.load_image("auto", "kraxi_logo.tif", "");

optlist = "fittextline={font=" + boldfont + " fontsize=9} image=" + image +
          " colwidth=" + c3 + " rowspan=3 margin=4";

tbl = p.add_table_cell(tbl, 3, 2, "Amazingly robust!", optlist);
```

Alternativ dazu können wir das Bild als PDF-Seite importieren. Dabei ist darauf zu achten, die PDI-Seite erst nach dem Aufruf von *PDF_fit_table( )* zu schließen.

```
int doc = p.open_pdi("kraxi_logo.pdf", "", 0);

page = p.open_pdi_page(doc, pageno, "");

optlist = "fittextline={font=" + boldfont + " fontsize=9} pdipage=" + page +
          " colwidth=" + c3 + " rowspan=3 margin=4";

tbl = p.add_table_cell(tbl, 3, 2, "Amazingly robust!", optlist);;
```

Step 6: Platzieren der Tabelle. Im letzten Schritt platzieren wir die Tabelle mit *PDF_fit_table( )*. Um die erste Zeile als Tabellenkopf zu verwenden, setzen wir *header=1*. Mit der Option *fill* und den Unteroptionen *area=header* und *fillcolor={rgb 0.8 0.8 0.87}* legen wir fest, dass die Kopfzeile(n) mit der angegebenen Farbe gefüllt werden. Mit der Option *stroke* und den Unteroptionen *line=frame linewidth=0.8* versehen wir den Tabellenrahmen mit einer Lineatur der Stärke 0.8. Mit *line=other linewidth=0.3* legen wir fest, dass alle Zellen eine Lineatur der Stärke 0.3 erhalten:

```
optlist = "header=1 fill={{area=header fillcolor={rgb 0.8 0.8 0.87}}} " +
          "stroke={{line=frame linewidth=0.8} {line=other linewidth=0.3}}";

result = p.fit_table(tbl, llx, lly, urx, ury, optlist);
```

```
if (result.equals("_error"))
        throw new Exception("Error: " + p.get_errmsg());

p.end_page_ext("");
```

8.3.5 Tabelleninstanzen

Eine Tabelleninstanz besteht aus allen Zeilen einer Tabelle, die in einer Fitbox platziert werden. Um eine Tabelle vollständig darzustellen, sind eine oder mehrere Tabelleninstanzen erforderlich. Bei jedem Aufruf von *PDF_fit_table()* wird eine Tabelleninstanz in einer Fitbox platziert. Die Fitboxen können auf derselben Seite, zum Beispiel bei einem mehrspaltigen Layout, oder auf mehreren Seiten platziert werden.

Die Tabelle in Abbildung 8.37 erstreckt sich über drei Seiten. Jede Tabelleninstanz wird in einer Fitbox auf einer eigenen Seite platziert. Bei jedem Aufruf von *PDF_fit_table()* wird die erste Zeile als Kopfzeile und die letzte Zeile als Fußzeile definiert.

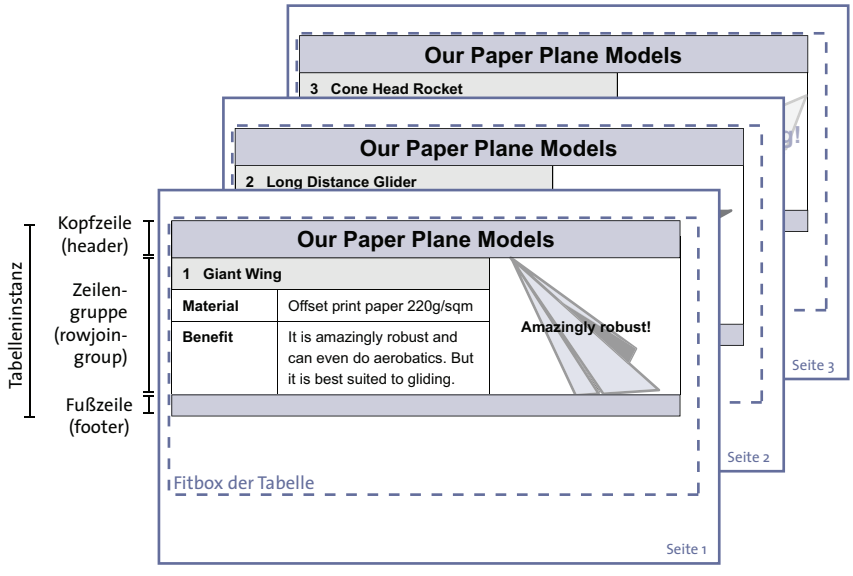


Abb. 8.37
Tabelle besteht aus Tabelleninstanzen, die in jeweils einer Fitbox platziert werden.

Das folgende Codefragment zeigt den generellen Aufbau der Schleife zur Platzierung von Tabelleninstanzen, die so lange durchlaufen wird, bis die Tabelle vollständig platziert wurde. Mit jeder Tabelleninstanz wird eine neue Seite angelegt.

```
do {
        /* Neue Seite anlegen */
        p.begin_page_ext(0, 0, "width=a4.width height=a4.height");

        /* Erste Zeile als Kopfzeile verwenden und alle Zellen mit Lineatur versehen */
        optlist = "header=1 stroke={{line=other}}";

        /* Tabelleninstanz platzieren */
        result = p.fit_table(tbl, llx, lly, urx, ury, optlist);
        if (result.equals("_error"))
            throw new Exception("Error: " + p.get_errmsg());
```

```
        p.end_page_ext("");

} while (result.equals("_boxfull"));

/* Ergebnis prüfen; "_stop" bedeutet "alles okay" */
if (!result.equals("_stop")) {
        if (result.equals( "_error"))
            throw new Exception("Error: " + p.get_errmsg());
        else {
            /* Jeder andere Rückgabewert wird benutzerspezifisch durch die Option
            "return" * bewirkt und erfordert entsprechend angepassten Code. */
            throw new Exception ("User return found in Textflow");
        }
}
/* Damit werden auch die in der Tabelle verwendeten Textflow-Handles gelöscht */
p.delete_table(tbl, "");
```

Kopf- und Fußzeilen. Mit den Optionen *header* und *footer* für *PDF_fit_table( )* können Sie die Anzahl der ersten oder letzten Tabellenzeilen definieren, die am Anfang oder Ende einer Tabelleninstanz platziert werden. Mit der Option *fill* und *area=header* oder *area=footer* lassen sich Kopf- und Fußzeilen individuell einfärben. Kopfzeilen bestehen aus den ersten *n* Zeilen der Tabellendefinition und Fußzeilen aus den letzten *m* Zeilen.

Kopf- und Fußzeilen werden für jede Tabelleninstanz in *PDF_fit_table( )* festgelegt. Demnach könnten manche Instanzen Kopf- und Fußzeilen enthalten und manche nicht. Dies wäre zum Beispiel nützlich, um eine bestimmte Zeile nur in der letzten Tabelleninstanz auszugeben.

Zusammenhalten von Zeilen. Um sicherzustellen, dass bestimmte Zeilen gemeinsam in einer Tabelleninstanz erscheinen, können sie mit der Option *rowjoingroup* derselben Zeilenverbindungsgruppe zugeordnet werden. Eine solche Zeilengruppe besteht aus mehreren aufeinanderfolgenden Zeilen. Alle Zeilen der Gruppe werden zusammengehalten und nicht auf verschiedene Tabelleninstanzen verteilt.

Die Zeilen einer Zelle, die sich über diese Zeilen erstreckt, bilden nicht automatisch eine Zeilenverbindungsgruppe.

Zu niedrige Fitbox. Ist die Fitbox zu niedrig, um die erforderlichen Kopf- und Fußzeilen sowie mindestens eine Body-Zeile oder Zeilengruppe aufzunehmen, werden die Zeilenhöhen gleichmäßig so lange verringert, bis die Tabelle in die Fitbox passt. Ist der dazu erforderliche Stauchungsfaktor jedoch kleiner als das in *vertshrinklimit* festgelegte Limit, so erfolgt keine Stauchung, und *PDF_fit_table( )* gibt den String *_error* oder eine entsprechende Fehlermeldung zurück. Um jegliche Stauchung zu unterbinden, verwenden Sie *vertshrinklimit=100%*.

Zu schmale Fitbox. Die Koordinaten der Fitbox der Tabelle werden beim Aufruf von *PDF_fit_table( )* explizit übergeben. Ist die Tabellenbreite, die sich aus der Summe der übergebenen Spaltenbreiten berechnet, größer als die Fitbox der Tabelle, werden alle Spalten so lange verkleinert, bis die Tabelle in die Fitbox passt. Ist der dazu erforderliche Stauchungsfaktor jedoch kleiner als das in *horshrinklimit* festgelegte Limit, so erfolgt keine Stauchung, und *PDF_fit_table( )* gibt den String *_error* oder eine entsprechende Fehlermeldung zurück. Um jegliche Stauchung zu unterbinden, verwenden Sie die Option *horshrinklimit=100%*.

Teilen einer Zelle. Passen nicht alle Zeilen, über die sich eine Zelle erstreckt, in die Fitbox, so wird die Zelle geteilt. Enthält die Zelle ein Bild, eine SVG-Grafik, eine PDI-Seite oder eine Textzeile, so wird der Zelleninhalt in der nächsten Tabelleninstanz wiederholt. Bei einer Zelle mit Textflow dagegen wird der Zelleninhalt in den übrigen Zeilen der Zelle fortgesetzt.

Abbildung 8.38 zeigt, wie die Zelle mit Textflow geteilt und der Textflow in der nächsten Zeile fortgeführt wird. Abbildung 8.39 zeigt eine Zelle mit Bild, das in der ersten Zeile der nächsten Tabelleninstanz wiederholt wird.

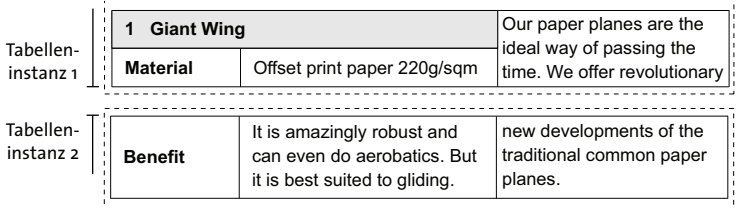


Abb. 8.38
Teilen einer Zelle

Aufspalten einer Zeile. Wenn die letzte Body-Zeile nicht mehr vollständig in die Fitbox der Tabelle passt, wird sie üblicherweise nicht aufgespalten. Dieses Verhalten wird über die Option *minrowheight* für *PDF_fit_table( )* gesteuert, die einen Standardwert von 100% besitzt. Im Standardfall wird die Zeile nicht aufgespalten, sondern vollständig in die nächste Tabelleninstanz übernommen.

Sie können den Wert für *minrowheight* verringern, so dass die letzte Body-Zeile mit dem angegebenen Prozentsatz des Inhalts in der ersten Instanz und mit dem Rest der Zeile in der nächsten Instanz platziert wird.

Abbildung 8.39 zeigt, wie der Textflow *It's amazingly robust...* geteilt und in der ersten Body-Zeile der nächsten Tabelleninstanz fortgeführt wird. Die Bildzelle, die sich über mehrere Zeilen erstreckt, wird geteilt und das Bild wird wiederholt. Die Textzeile *Benefit* wird ebenfalls wiederholt.

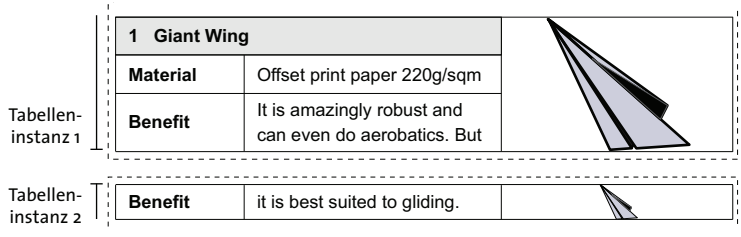


Abb. 8.39
Teilen einer Zeile

8.3.6 Formatierungsalgorithmus für Tabellen

Dieser Abschnitt beschreibt detailliert die Schritte, die vom Tabellenformatierer bei der Platzierung einer Tabelle durchgeführt werden. Die nachfolgende Beschreibung bezieht sich auf horizontalen Text. Sie gilt auch für vertikalen oder gedrehten Text, wenn man die Begriffe »Zeilenhöhe« und »Spaltenbreite« vertauscht.

Beim ersten Aufruf von *PDF_fit_table( )* werden für alle Zellen die Optionen *colwidth*, *rowheight*, *fittextline* und *fittextflow* untersucht und aus den Spaltenbreiten, Zeilenhöhen und Texten die Breite und Höhe der gesamten Tabelle berechnet.

Berechnen der Höhe und Breite von Tabellenzellen mit Textzeilen. Der Tabellenformatierer bestimmt zunächst die Größe all jener Tabellenzellen mit Textzeilen, die sich über solche Tabellenspalten oder -zeilen erstrecken, für die keine *colwidth* bzw. *rowheight* angegeben wurde. Dazu berechnet er die Breite der Textzeile und damit der Tabellenzelle gemäß der Option *fittextline*. Als Höhe der Tabellenzelle nimmt er die doppelte Größe des Texts (genauer: der *boxheight* mit dem Standardwert *{capheight none}*, sofern nicht anders spezifiziert). Bei vertikalem Text dient die Breite des breitesten Zeichens als Zellenbreite. Bei Text, der nach West oder Ost orientiert ist, dient die doppelte Texthöhe als Zellenbreite.

Die resultierende Breite und Höhe der Tabellenzelle wird dann auf alle die Zelle umfassenden Spalten und Zeilen verteilt, für die keine *colwidth* bzw. *rowheight* angegeben wurde.

Berechnen der vorläufigen Tabellengröße. Danach berechnet der Formatierer die vorläufige Tabellenbreite bzw. Tabellenhöhe als Summe aller Spaltenbreiten bzw. Zeilenhöhen. Prozentual angegebene Spaltenbreiten und Zeilenhöhen werden in absolute Werte umgerechnet, wobei sich diese auf die Breite bzw. Höhe der ersten Fitbox beziehen. Sind noch Spalten oder Zeilen verblieben, für die keine *colwidth* bzw. *rowheight* angegeben wurde, so erhalten diese jeweils denselben Anteil, bis die vorläufige Tabelle so breit bzw. so hoch wie die erste Fitbox ist (siehe Abbildung 8.33 und Abbildung 8.34).

In den meisten Fällen ist es deshalb sinnvoll, für jede Zelle zumindest eine minimale *rowheight* anzugeben, da die Tabelle sonst automatisch an die Höhe der Fitbox angepasst wird.

Vergrößern zu kleiner Zellen. Nun berechnet der Formatierer alle inneren Zellboxen (siehe Abbildung 8.32). Falls die kombinierten Ränder größer als Breite bzw. Höhe der Zelle sind, wird die Zellbox geeignet vergrößert, indem alle zur Zelle gehörenden Spalten und Zeilen gleichmäßig vergrößert werden.

Horizontales Einpassen von Textzeilen. Der Formatierer versucht, jede Tabellenzelle mit einer Textzeile so zu verbreitern, dass die Textzeile ohne Verkleinerung der Schrift in die Zelle passt. Ist dies nicht möglich, passt er die Textzeile automatisch mit der Option *fitmethod=auto* ein, sodass sie garantiert nicht aus der inneren Zellbox herausragt. Sie können jegliche Verbreiterung der Zelle verhindern, indem Sie in der Option *fittextline* die Unteroption *fitmethod=auto* setzen.

Mit der Option *colscalegroup* gewährleisten Sie, dass alle zu einer definierten Spaltenskalierungsgruppe gehörenden Spalten auf die gleiche Breite skaliert werden, d.h. ihre Breiten werden vereinheitlicht und an die breiteste Spalte in der Gruppe angepasst (siehe Abbildung 8.35).

Vermeiden zwangsweiser Worttrennung. Ist die berechnete Tabelle nicht so breit wie die Fitbox, versucht der Formatierer außerdem, eine Zelle mit Textflow soweit zu verbreitern, dass der Text ohne zwangsweise Worttrennung passt. Mit der Option *checkwordsplitting=false* kann dies unterbunden werden. Eine Verbreiterung solcher Zellen findet so lang statt, bis die Tabellenbreite der Fitboxbreite entspricht.

Die Differenz zwischen Tabellen- und Fitboxbreite kann über den Schlüssel *horboxgap* in *PDF_info_table( )* abgefragt werden.

Vertikales Einpassen von Text. Der Formatierer versucht, jede Zelle mit einer Textzeile oder einem Textflow vertikal zu vergrößern, sodass die Textzeile bzw. der Textflow ohne Verkleinerung der Schrift in die innere Zellbox passt. Die Zellenhöhe wird jedoch nicht vergrößert, wenn für Textzeile oder Textflow die Unteroption *fitmethod=auto* gesetzt ist oder der Textflow mit der Option *continuetextflow* in einer anderen Zelle fortgesetzt wird.

Die beschriebene Vergrößerung gilt nur für Textzeilen und Textflows, nicht aber für Zellen mit anderen Inhalten, also Bilder, Grafiken, PDI-Seiten, Pfadobjekte, Annotationen und Felder.

Mit der Option *rowscalegroup* können Sie sicherstellen, dass alle Zeilen, die zur gleichen Zeilenskalierungsgruppe gehören, auf die gleiche Höhe skaliert werden.

Fortsetzen der Tabelle in der nächsten Fitbox. Ist die resultierende Gesamthöhe der Tabelle größer als die Fitbox (passen also nicht alle Tabellenzeilen in die Fitbox), so bricht der Formatierer die Ausgabe der Tabelle vor der ersten Zeile ab, die nicht mehr in die Fitbox passt.

Passen dabei nicht alle Zeilen, über die sich eine Zelle erstreckt, in die Fitbox, so wird diese Zelle geteilt. Enthält die Zelle ein Bild, eine SVG-Grafik, eine PDI-Seite, ein Pfadobjekt, eine Annotation, ein Formularfeld oder eine Textzeile, so wird der Zelleninhalt in der nächsten Fitbox wiederholt, sofern nicht *repeatcontent=false* gesetzt wurde. Ein Textflow dagegen wird in den folgenden Zeilen der Zelle fortgesetzt (siehe Abbildung 8.38).

Mit der Option *rowjoingroup* können Sie sicherstellen, dass alle Zeilen, die zur gleichen Zeilenverbindungsgruppe gehören, immer zusammen in einer Fitbox erscheinen. Alle zum Header und zum Footer gehörenden Zeilen sowie eine Body-Zeile bilden automatisch eine Zeilenverbindungsgruppe. Der Formatierer kann die Ausgabe der Tabelle daher unter Umständen bereits vor der ersten Zeile, die nicht mehr in die Fitbox passt, abbrechen (siehe Abbildung 8.37).

Mit der Option *return* können Sie sicherstellen, dass die Ausgabe nach der betroffenen Zeile abgebrochen wird.

Aufspalten einer Zeile. Bei sehr hohen Zeilen oder nur einer einzigen Body-Zeile kann die Zeile unter Umständen aufgespalten werden. Passt die letzte Body-Zeile nicht mehr vollständig in die aktuelle Fitbox der Tabelle, so wird sie komplett in die nächste Fitbox übernommen. Dieses Verhalten wird über die Option *minrowheight* für *PDF_fit_table( )* gesteuert, die einen Standardwert von 100% hat. Sie können den Wert für *minrowheight* verringern, so dass die letzte Body-Zeile mit dem angegebenen Prozentsatz des Inhalts in der aktuellen Fitbox und mit dem Rest der Zeile in der nächsten Fitbox platziert wird (siehe Abbildung 8.39).

Mit dem Schlüsselwort *rowsplit* für *PDF_info_table( )* kann abgefragt werden, ob eine Zeile aufgespalten wurde.

Anpassen der berechneten Tabellenbreite. Die berechnete Tabellenbreite kann nach einem Berechnungsschritt, etwa nach dem horizontalen Einpassen einer Textzeile, größer werden als die Fitboxbreite. In diesem Fall werden alle Spaltenbreiten gleichmäßig verkleinert, bis die Tabelle so breit wie die Fitbox ist. Die Option *horshrinklimit* limitiert diese Verkleinerung.

Der angewandte horizontale Verkleinerungsfaktor kann über das Schlüsselwort *horshrinking* in *PDF_info_table( )* abgefragt werden.

Wird der durch *horshrinklimit* definierte Schwellwert überschritten, erscheint die Fehlermeldung:

```
Calculated table width $1 is too large (> $2, shrinking $3)
```

Hierbei steht *$1* für die berechnete Tabellenbreite, *$2* für die maximal mögliche Breite und *$3* für den Wert von *horshrinklimit*.

Anpassen der Tabellengröße an eine kleine Fitbox. Ist die Tabellenbreite, die für die Vorgänger-Fitbox berechnet wurde, zu groß für die aktuelle Fitbox, so verkleinert der Formatierer alle Spalten gleichmäßig, bis die Tabelle so breit wie die neue Fitbox ist. Es findet keinerlei Anpassung der Zellinhalte statt. Um die Tabellenbreite neu zu berechnen, verwenden Sie *PDF_fit_table( )* mit *rewind=1*.

8.4 Matchboxen

Matchboxen bieten Zugriff auf Koordinaten, die von PDFlib zur Platzierung bestimmter Seiteninhalte berechnet wurden. Matchboxen werden mit keiner besonderen API-Funktion definiert, sondern mit der Option *matchbox* für die Funktion festgelegt, die das betreffende Element platziert, zum Beispiel *PDF_fit_textline( )* und *PDF_fit_image( )*. Matchboxen sind zu verschiedenen Zwecken einsetzbar:

- Matchboxen können verziert, z.B. eingefärbt oder mit einem Rahmen versehen werden.
- Matchboxen können zur automatischen Erzeugung von Anmerkungen mit *PDF_create_annotation( )* verwendet werden.
- Matchboxen definieren die Höhe einer Textzeile, die mit *PDF_fit_textline( )* in eine Box eingepasst, oder die Höhe eines Textabschnitts in einem Textflow, der farblich oder anderweitig ausgezeichnet werden soll (Option *boxheight)*.
- Matchboxen definieren, wie ein Bild beschnitten wird.
- Die Koordinaten und andere Eigenschaften der Matchbox können mit *PDF_info_matchbox( )* abgefragt und z.B. zum Einfügen eines Bildes weiterverwendet werden.

PDFlib berechnet die Matchbox zu einem Element als Rechteck, das der Boundingbox entspricht, die die (durch alle relevanten Optionen bestimmte) Position des Elements auf der Seite beschreibt. Bei Rechtecken und Tabellenzellen kann eine Matchbox aus mehreren Rechtecken bestehen, wenn Zeilen umbrochen oder geteilt wurden.

Das oder die Rechtecke einer Matchbox werden vor dem zu platzierenden Element gezeichnet. Deswegen kann es vorkommen, dass der Rand oder die Füllung der Matchbox vom Element verdeckt werden. Umgekehrt ist dies nicht möglich. Insbesondere werden die Teile der Matchbox, die im Bereich eines Bildes liegen, vom Bild verdeckt. Wenn das Bild mit *fitmethod=slice* oder *fitmethod=clip* platziert wird, werden auch die außerhalb der Fitbox des Bildes liegenden Matchbox-Ränder beschnitten. Um diesen Effekt zu vermeiden, kann das Matchbox-Rechteck mit den elementaren Zeichenfunktionen (z. B. *PDF_rect( ))* nach dem Aufruf von *PDF_fit_image( )* explizit gezeichnet werden. Die Koordinaten der Matchbox können Sie mit *PDF_info_matchbox( )* abrufen, sofern Sie die Matchbox in *PDF_fit_image( )* mit einem Namen versehen haben.

In den folgenden Abschnitten werden einige Beispiele zum Einsatz von Matchboxen gezeigt. Details zu den Funktionen, die die Option *matchbox* unterstützen, finden Sie in der PDFlib-Referenz.

8.4.1 Verzierung einer Textzeile

Beginnen wir mit einer Beschreibung von Matchboxen in Textzeilen. In *PDF_fit_textline( )* entspricht die Matchbox der Textbox des übergebenen Textes. Die Breite der Textbox entspricht standardmäßig der Textbreite, und die Höhe entspricht der *capheight* der definierten Fontgröße. Um die Abmessungen der Matchbox zu veranschaulichen, füllt das folgende Codefragment die Matchbox mit Hintergrundfarbe (siehe Abbildung 8.40a):

```
String  optlist =
        "font=" + normalfont + " fontsize=8 position={left top} " +
        "matchbox={fillcolor={rgb 0.8 0.8 0.87} boxheight={capheight none}}";

p.fit_textline("Giant Wing Paper Plane", 2, 20, optlist);
```

Sie können die Option *boxheight* auch weglassen, da *boxheight={capheight none}* ohnehin die Standardeinstellung ist. Wir erzielen einen besseren Effekt, wenn wir mit der Option *boxheight* die Box so weit vergrößern, dass sie auch die Unterlängen abdeckt (siehe Abbildung 8.40b).

Um die Box bis auf die Fontgröße zu erhöhen, verwenden wir *boxheight={fontsize descender}* (siehe Abbildung 8.40c).

Im nächsten Schritt erweitern wir die Matchbox mit den *offset*-Unteroptionen nach links, rechts und unten, um den Abstand zwischen Text und Boxrändern zu vereinheitlichen. Außerdem zeichnen wir ein Rechteck um die Matchbox, indem wir mit *borderwidth* eine Randbreite definieren (siehe Abbildung 8.40d).

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/text_on_color`.

Abb. 8.40 Verzieren einer Textzeile mittels einer Matchbox und verschiedener Unteroptionen

Generierte Ausgabe	***Unteroptionen der Option matchbox für PDF_fit_textline( )***
a) Giant Wing Paper Plane	`boxheight={capheight none}`
b) Giant Wing Paper Plane	`boxheight={ascender descender}`
c) Giant Wing Paper Plane	`boxheight={fontsize descender}`
d) Giant Wing Paper Plane	`boxheight={fontsize descender} borderwidth=0.3 offsetleft=-2 offsetright=2 offsetbottom=-2`

8.4.2 Matchboxen in einem Textflow

Gestaltung von Abschnitten eines Textflows. In diesem Abschnitt werden wir Teile eines Textflows gestalten: Wir zeichnen die Wörter *very dangerous* so aus, als seien sie mit einem Marker angestrichen. Dazu schließen wir die Wörter in die Inline-Optionen *matchbox* und *matchbox=end* ein (siehe Abbildung 8.41).

Abb. 8.41 Textflow mit der Inline-Option matchbox

Generierte Ausgabe	***Text und Inline-Optionen für PDF_create_textflow( )***
It is very dangerous to fly the Giant Wing in a thunderstorm.	`It is <matchbox={fillcolor=red boxheight={ascender descender}}>very dangerous <matchbox=end> to fly the Giant Wing in a thunderstorm.`

Textflow-Matchbox mit Weblink. Im Folgenden statten wir einen Abschnitt des Textflows mit einem Weblink aus. Dazu erstellen wir im ersten Schritt den Textflow inklusive einer Matchbox namens *kraxi*, die den zu verknüpfenden Textabschnitt kennzeichnet. Im zweiten Schritt erzeugen wir die Aktion zum Öffnen eines URL. Dann legen wir eine Anmerkung vom Typ *Link* mit einem unsichtbaren Rahmen an. In deren Optionsliste verweisen wir auf die Matchbox *kraxi*, die als Verknüpfungsrechteck verwendet wird (die Rechteckkoordinaten in *PDF_create_annotation( )* werden dabei ignoriert).

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `text_output/weblink_in_text`.

```
/* Textflow mit Matchbox "kraxi" erstellen und platzieren */
String  tftext =
        "For more information about the Giant Wing Paper Plane see the Web site of " +
        "<underline=true matchbox={name=kraxi boxheight={fontsize descender}}>" +
        "Kraxi Systems, Inc.<matchbox=end underline=false>";

String  optlist = "font=" + normalfont + " fontsize=8 leading=110%";
tflow = p.create_textflow(tftext, optlist);
if (tflow == -1)
        throw new Exception("Error: " + p.get_errmsg());

result = p.fit_textflow(tflow, 0, 0, 50, 70, "fitmethod=auto");
if (!result.equals("_stop"))
        { /* ... */ }

/* Aktion von Typ "URI" erzeugen */
optlist = "url={http://www.kraxi.com}";
act = p.create_action("URI", optlist);

/* Anmerkung vom Typ "Link" auf Matchbox "kraxi" anlegen */
optlist = "action={activate " + act + "} linewidth=0 usematchbox={kraxi}";
p.create_annotation(0, 0, 0, 0, "Link", optlist);
```

Selbst wenn sich der Text *Kraxi Systems, Inc.* über mehrere Zeilen erstreckt, wird in einem einzigen Aufruf von *PDF_create_annotation( )* automatisch die passende Anzahl von Verknüpfungen erstellt. Abbildung 8.42 zeigt das Ergebnis.

Abb. 8.42
Hinzufügen von Weblinks zu Teilen eines Textflows

8.4.3 Matchboxen und Bilder

Bild mit Weblink. Wir möchten den von einem Bild bedeckten Bereich mit einem Weblink versehen. Dazu platzieren wir das Bild unter Angabe einer Matchbox. Der Code ähnelt dem Code in Abschnitt »Textflow-Matchbox mit Weblink«, Seite 263. Statt des Textflows platzieren wir jedoch das Bild mit folgender Optionsliste:

```
String  optlist = "boxsize={130 130} fitmethod=meet matchbox={name=kraxi}";
p.fit_image(image, 10, 10, optlist);
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/link_annotations`.

Bild mit Rahmen. In diesem Beispiel verwenden wir die Matchbox, um einen Rahmen um ein Bild zu zeichnen. Mit *fitmethod=meet* platzieren wir das Bild unter Beibehaltung seiner Proportionen vollständig in der angegebenen Box. Mit der Option *matchbox* so-

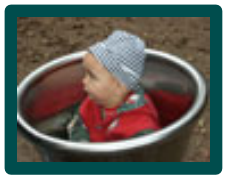
wie der Unteroption *borderwidth* definieren wir einen breiten Rand um das Bild. Mit der Unteroption *strokecolor* legen wir die Randfarbe fest, und die Unteroptionen *linecap* und *linejoin* dienen zur Abrundung der Ecken.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `images/frame_around_image`.

Die Matchbox wird prinzipiell vor dem Bild gezeichnet, was bedeutet, dass sie vom Bild teilweise verdeckt würde. Um dies zu vermeiden, vergrößern wir die Matchbox mit den *offset*-Unteroptionen um 50 Prozent der Randbreite, um den Rand über den vom Bild verdeckten Bereich hinausragen zu lassen. Alternativ dazu können wir den Rand entsprechend verbreitern. Abbildung 8.43 zeigt die Optionsliste für *PDF_fit_image( )* zum Zeichnen des Rahmens.

Abb. 8.43 Zeichnen eines Rahmens um ein Bild mittels Matchbox

Generierte Ausgabe	***Optionsliste für PDF_fit_image( )***

	`boxsize={60 60} position={center} fitmethod=meet`
	`matchbox={name=kraxi borderwidth=4`
	`offsetleft=-2 offsetright=2 offsetbottom=-2 offsettop=2`
	`linecap=round linejoin=round strokecolor L={rgb 0.0 0.3 0.3}}`

Ausrichten von Text an einem Bild. Das folgende Codefragment zeigt, wie sich vertikaler Text am rechten Rand eines Bildes ausrichten lässt. Das Bild wird mit *fitmethod=meet* unter Beibehaltung seiner Proportionen vollständig in die angegebene Box platziert. Die Koordinaten der Fitbox werden mit *PDF_info_matchbox( )* abgefragt, und eine vertikale Textzeile wird relativ zur rechten unteren Ecke *(x2, y2)* der Fitbox platziert. Außerdem wird die Matchbox mit einem Rand versehen (siehe Abbildung 8.44).

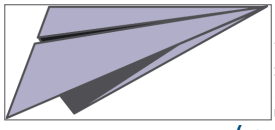
Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `images/align_text_at_image`.

```
/* Optionsliste zum Laden und Platzieren des Bildes */
String  optlist = "boxsize={300 200} position={center} fitmethod=meet " +
        "matchbox={name=giantwing borderwidth=3 strokecolor={rgb 0.85 0.83 0.85}}";

/* Bild laden und platzieren*/
...

/* Koordinaten der rechten unteren (zweiten) Matchbox-Ecke abfragen */
if ((int) p.info_matchbox("giantwing", 1, "exists") == 1)
{
        x1 = p.info_matchbox("giantwing", 1, "x2");
        y1 = p.info_matchbox("giantwing", 1, "y2");
}
/* Textzeile an abgefragter Ecke mit einem Abstand von 2 beginnen */
p.fit_textline("Foto: Kraxi", x2+2, y2+2, "font=" + font + " fontsize=8 orientate=west");
```

Abb. 8.44 Koordinaten zur Platzierung der Textzeile mittels Matchbox abfragen

Generierte Ausgabe	***Generierungsschritte***
	*Schritt 1: Bild mit Matchbox platzieren* *Schritt 2: Matchbox-Info für Koordinaten (x2, y2) abfragen* *Schritt 3: Textzeile an den abgefragten Koordinaten (x2, y2) mit Option* `orientate=west` *beginnen*

9 Interaktive Elemente

Cookbook *Codebeispiele zu interaktiven Elementen finden Sie in der Kategorie* `interactive` *des PDFlib Cookbook.*

9.1 Links, Lesezeichen und Anmerkungen

Dieser Abschnitt zeigt die Vorgehensweise beim Erstellen von interaktiven Elementen wie Lesezeichen, Formularfelder oder Anmerkungen. Abbildung 9.1 zeigt das Ergebnisdokument mit allen interaktiven Elementen, die wir in diesem Abschnitt erzeugen werden. Das Dokument enthält folgende Hypertext-Elemente:

- Im Dokumentfenster rechts oben befindet sich über dem Text *www.kraxi.com* ein unsichtbarer Weblink. Wenn Sie in diesen Bereich klicken, wird die zugehörige Webseite geöffnet.
- Unmittelbar darunter liegt ein grau hinterlegtes Formular-Textfeld, das automatisch (per JavaScript) mit dem aktuellen Datum gefüllt wird.
- Die rote Reißzwecke enthält eine Anmerkung mit Dateianhang. Durch Anklicken lässt sich die angehängte Datei öffnen.
- Unten links gibt es eine Formularfeld-Schaltfläche mit einem Druckersymbol. Wenn Sie auf diese Schaltfläche klicken, wird der Acrobat-Menübefehl *Datei, Drucken* ausgeführt.
- Im Navigationsfenster befindet sich das Lesezeichen »Our Paper Planes Catalog«. Wenn Sie auf dieses Lesezeichen klicken, wird die Seite eines anderen PDF-Dokuments angezeigt.

Im Folgenden wird gezeigt, wie Sie diese interaktiven Elemente mit PDFlib generieren.

Links zu einer Webseite. Wir erzeugen zunächst eine Verknüpfung zur Webseite *www.kraxi.com*. Dies geschieht in drei Schritten. Zuerst platzieren wir den Text, auf dem sich der Weblink befinden soll. Anhand der Option *matchbox* mit *name=kraxi* legen wir das Rechteck der Textfitbox fest, um später darauf zu verweisen.

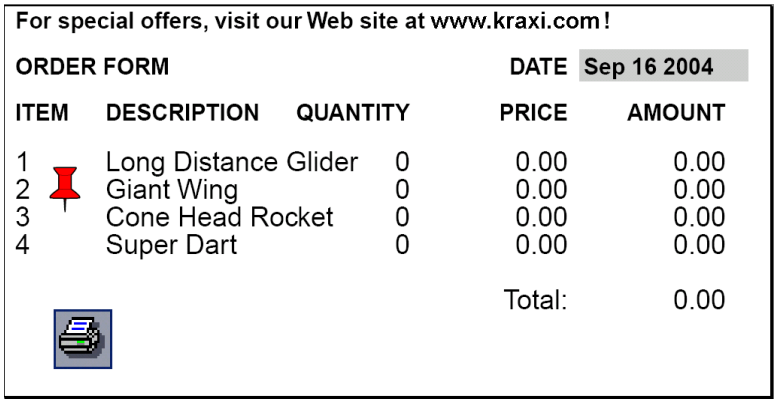


Abb. 9.1 Dokument mit interaktiven Elementen

Im zweiten Schritt legen wir eine Aktion vom Typ *URI* (in Acrobat: *Web-Verknüpfung öffnen)* an. Dabei wird ein Action-Handle erstellt, das wir im Folgenden einem oder auch verschiedenen interaktiven Elementen zuweisen können.

Im dritten Schritt generieren wir die eigentliche Verknüpfung. Eine Verknüpfung ist in PDF eine Anmerkung *(Annotation)* vom Typ *Link*. Dem Link übergeben wir in der Option *action* das Ereignis *activate*, bei dem die Aktion ausgeführt werden soll, sowie unser oben erstelltes Handle *act* für die auszuführende Aktion. Um den Link wird standardmäßig ein schmaler schwarzer Rahmen gezogen. Das mag am Anfang zur genauen Positionierung nützlich sein, wir blenden den Rand aber mit *linewidth=0* aus.

```
normalfont = p.load_font("Helvetica", "unicode", "");
p.begin_page_ext(pagewidth, pageheight, "topdown");

/* Textzeile "Kraxi Systems, Inc." mit Matchbox platzieren */
String  optlist =
        "font=" + normalfont + " fontsize=8 position={left top} " +
        "matchbox={name=kraxi} fillcolor={rgb 0 0 1} underline";

p.fit_textline("Kraxi Systems, Inc.", 2, 20, optlist);

/* Aktion des Typs URI erzeugen */
optlist = "url={http://www.kraxi.com}";
int act = p.create_action("URI", optlist);

/* Link über Matchbox "kraxi" anlegen */
optlist = "action={activate " + act + "} linewidth=0 usematchbox={kraxi}";
/* 0 Rechteck-Koordinaten werden durch Matchbox-Koordinaten ersetzt */
p.create_annotation(0, 0, 0, 0, "Link", optlist);

p.end_page_ext("");
```

Ein Beispiel zur Erstellung eines Weblinks über einem Bild oder Teilen eines Textflows finden Sie in Abschnitt 8.4, »Matchboxen«, Seite 262.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/link_annotations`.

Lesezeichen mit Sprung in eine andere Datei. Nun werden wir das Lesezeichen »Our Paper Planes Catalog« erstellen, das auf eine andere PDF-Datei verweist, und zwar auf den Papierfliegerkatalog *paper_planes_catalog.pdf*. Wir erzeugen zunächst eine Aktion vom Typ *GoToR*. In der Optionsliste zu dieser Aktion definieren wir mit der Option *filename* den Namen des Zieldokuments und mit der Option *destination*, dass ein bestimmter Seitenausschnitt deutlich vergrößert angezeigt wird. Genauer gesagt soll das Dokument auf der zweiten Seite *(page=2)* in fester Vergrößerung *(type=fixed)* mit der Seitenmitte im sichtbaren Bereich *(left=50 top=200)* und in einer Vergrößerung von 200% *(zoom 2)* angezeigt werden.

```
String  optlist =
        "filename=paper_planes_catalog.pdf " +
        "destination={page=2 type=fixed left=50 top=200 zoom=2}";

goto_action = p.create_action("GoToR", optlist);
```

Im nächsten Schritt erstellen wir das eigentliche Lesezeichen *(Bookmark)*. Dem Lesezeichen übergeben wir in der Option *action* das Ereignis *activate*, bei dem die Aktion ausgeführt werden soll, sowie unser oben erstelltes Handle *goto_action* für die auszuführende Aktion. Mit der Option *fontstyle=bold* legen wir Fettschrift fest und mit *textcolor=blue* färben wir das Lesezeichen blau ein. Als Funktionsparameter übergeben wir den anzuzeigenden Lesezeichentext »Our Paper Planes Catalog«.

```
String  optlist =
        "action={activate " + goto_action + "} fontstyle=bold textcolor=blue";

catalog_bookmark = p.create_bookmark("Our Paper Planes Catalog", optlist);
```

Beim Anklicken des Lesezeichens wird der definierte Seitenausschnitt des Zieldokuments angezeigt.

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/nested_bookmarks`.

Anmerkung mit Dateianhang. Als nächstes Beispiel erstellen wir eine Anmerkung mit einem Dateianhang. Dazu erzeugen wir eine Anmerkung vom Typ *FileAttachment*. Mit der Option *filename* übergeben wir die anzuhängende Datei und mit *mimetype=image/gif* den Typ der Datei (MIME ist eine verbreitete Konvention zur Typisierung von Dateien). Die Anmerkung erscheint als Reißzwecke *(iconname=pushpin)* in rot *(annotcolor=red)* und verfügt über einen Tooltip *(contents={Get the Kraxi Paper Plane!})*. Sie wird nicht gedruckt *(display=noprint)*.

```
String  optlist =
        "filename=kraxi_logo.gif mimetype=image/gif iconname=pushpin " +
        "annotcolor=red contents={Get the Kraxi Paper Plane!} display=noprint";

p.create_annotation(left_x, left_y, right_x, right_y, "FileAttachment", optlist);
```

Beachten Sie dabei, dass die Größe des mit *iconname* definierten Symbols nicht variabel ist. Es wird in seiner Standardgröße in die linke obere Ecke des übergebenen Rechtecks platziert.

9.2 Formularfelder und JavaScript

Schaltfläche zum Drucken. Das folgende Beispiel erstellt eine Schaltfläche zum Drucken des Dokuments. In der ersten Variante des Beispiels versehen wir die Schaltfläche mit der Beschriftung *Print*, in der zweiten Version dann mit einem Symbol. Dazu erzeugen wir zunächst eine Aktion vom Typ *Named* (in Acrobat: *Menübefehl ausführen)*. Außerdem müssen wir die Schriftart für die Beschriftung festlegen:

```
print_action = p.create_action("Named", "menuname=Print");
button_font = p.load_font("Helvetica-Bold", "unicode", "");
```

Der Formularschaltfläche übergeben wir in der Option *action* das Ereignis *up* (in Acrobat: *Maustaste loslassen)*, bei dem die Aktion ausgeführt werden soll, sowie unser oben erstelltes Handle *print_action* für die auszuführende Aktion. Mit *backgroundcolor {rgb=yellow}* legen wir als Hintergrundfarbe Gelb fest und mit *bordercolor {rgb=red}* definieren wir einen schwarzen Rand. Mit *caption=Print* beschriften wir die Schaltfläche mit dem Text *Print,* und mit *tooltip ={Print the document}* legen wir einen Hilfstext fest. Mit *font* schließlich legen wir den Font anhand des oben erzeugten Handles *button_font* fest. Die Größe der Beschriftung wird standardmäßig so angepasst, dass diese vollständig auf der Schaltfläche Platz hat. Danach wird das eigentliche Feld mit seinen Koordinaten, dem Namen *print_button*, dem Typ *pushbutton* sowie seinen Optionen erstellt.

```
String  optlist =
        "action {up " + print_action + "} backgroundcolor=yellow " +
        "bordercolor=black caption=Print tooltip={Print the document} font=" +
        button_font;

p.create_field(left_x, left_y, right_x, right_y, "print_button", "pushbutton", optlist);
```

Im Folgenden erweitern wir die erste Variante des Beispiels, indem wir den Text *Print* auf der Schaltfläche durch ein Druckersymbol ersetzen. Dazu müssen wir die entsprechende Bilddatei *print_icon.jpg* bereits vor dem Anlegen der Seite als Template laden. Mit der Option *icon* weisen wir der Schaltfläche das oben erstellte Template-Handle *print_icon* zu. Dann erzeugen wir wie oben die Formularschaltfläche:

```
print_icon = p.load_image("auto", "print_icon.jpg", "template");
if (print_icon == -1)
{
        /* Fehlerbehandlung */
        return;
}
p.begin_page_ext(pagewidth, pageheight, "");
...
String  optlist = "action={up " + print_action + "} icon=" + print_icon +
        " tooltip={Print the document} font=" + button_font;

p.create_field(left_x, left_y, right_x, right_y, "print_button", "pushbutton", optlist);
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/form_pushbutton`.

Einfaches Textfeld. Als nächstes steht die Erstellung des Textfelds an, das sich in unserem Dokument rechts oben befindet und zur Datumseingabe dienen soll. Dazu besor-

gen wir uns ein Font-Handle und legen dann das Formularfeld an. Es erhält den Namen *date*, ist vom Typ *textfield* und wird grau hinterlegt.

```
textfield_font = p.load_font("Helvetica-Bold", "unicode", "");
String  optlist = "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(left_x, left_y, right_x, right_y, "date", "textfield", optlist);
```

Die Fontgröße eines Textfelds ist standardmäßig auf *auto* gesetzt. Wenn Sie Text eingeben, wird dieser zunächst in der Feldgröße angezeigt. Erreicht er das Feldende, wird er automatisch verkleinert, sodass er immer vollständig im Feld sichtbar ist.

Cookbook *Vollständige Codebeispiele hierzu finden Sie in den Cookbook-Topics* `interactive/form_textfield_layout` *und* `interactive/form_textfield_height`.

Textfeld mit JavaScript. Um das obige Textfeld eleganter zu gestalten, soll es automatisch mit dem aktuellen Datum gefüllt werden, und zwar immer beim Öffnen der Seite. Dazu erzeugen wir im ersten Schritt eine Aktion vom Typ *JavaScript* (in Acrobat: *JavaScript ausführen)*. In der Optionsliste zu dieser Aktion definieren wir mit *script* ein JavaScript-Fragment, das das aktuelle Datum im Format Monat-Tag-Jahr in das Textfeld *date* einträgt.

```
String  optlist =
        "script={var d = util.printd('mmm dd yyyy', new Date()); " +
        "var date = this.getField('date'); date.value = d;}"

show_date = p.create_action("JavaScript", optlist);
```

Im zweiten Schritt legen wir die Seite an. In der Optionsliste zu dieser Seite definieren wir mit *action*, dass die oben definierte Aktion *show_date* durch das Ereignis *open* (in Acrobat: *Seite öffnen)* ausgelöst wird.

```
String optlist = "action={open " + show_date + "}";
p.begin_page_ext(pagewidth, pageheight, optlist);
```

Im letzten Schritt legen wir das Formular-Textfeld wie oben an. Das aktuelle Datum wird dann bei jedem Öffnen der Seite automatisch in das Feld mit dem Namen *date* eingetragen:

```
textfield_font = p.load_font("Helvetica-Bold", "winansi", "");
String  optlist = "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(left_x, left_y, right_x, right_y, "date", "textfield", optlist);
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/form_textfield_fill_with_js`.

Formatierungsoptionen für Textfelder. Der Inhalt eines Textfeldes kann in Acrobat anhand verschiedener Optionen, etwa für Zahlen, Prozentwerte oder Datumsangaben formatiert werden. Diese Art der Formatierung ist über benutzerdefinierten JavaScript-Code implementiert, der von Acrobat verwendet wird. Von PDFlib werden diese Formatierungsfunktionen nicht direkt unterstützt, da sie nicht in der PDF-Referenz festgelegt sind. Wir möchten Ihnen trotzdem einige nützliche Hinweise geben, damit Sie eigene Formatierungsoptionen für Textfelder erstellen können, indem Sie einfache JavaScript-Codefragmente mit der Option *action* von *PDF_create_field( )* übergeben.

Um ein Textfeld zu formatieren, werden ihm JavaScript-Codefragmente als *keystroke*- und *format*-Aktionen mitgegeben. Der JavaScript-Code ruft eine interne Acrobat-Funktion auf, deren Parameter die Formatierung im Einzelnen steuern.

Das folgende Beispiel erstellt die beiden Aktionen *keystroke* und *format* und ordnet sie einem Formularfeld zu. Der Feldinhalt wird dabei mit zwei Dezimalstellen und dem Währungssymbol EUR formatiert:

```
keystroke_action = p.create_action("JavaScript",
        "script={AFNumber_Keystroke(2, 0, 3, 0, \"EUR \", true); }");

format_action = p.create_action("JavaScript",
        "script={AFNumber_Format(2, 0, 0, 0, \"EUR \", true); }");

String  optlist = "font=" + font + " action={keystroke " + keystroke_action +
                      " format=" + format_action + "}";
p.create_field(50, 500, 250, 600, "price", "textfield", optlist);
```

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/form_textfield_input_format`.

Um die in Acrobat unterstützten Formate zu nutzen, müssen Sie im JavaScript-Code die entsprechenden Funktionen angeben. Tabelle 9.1 zeigt die JavaScript-Funktionsnamen für die Aktionen *keystroke* und *format* für alle unterstützten Formate; die Funktionsparameter werden in Tabelle 9.2 beschrieben. Diese Funktionen werden so wie im obigen Beispiel eingesetzt.

Tabelle 9.1 JavaScript-Formatierungsfunktionen für Textfelder

Format	***JavaScript-Funktionen zum Einsatz mit den Aktionen keystroke und format***
Zahl	`AFNumber_Keystroke(nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend)` `AFNumber_Format(nDec, sepStyle, negStyle, currStyle, strCurrency, bCurrencyPrepend)`
Prozent	`AFPercent_Keystroke(ndec, sepStyle), AFPercent_Format(ndec, sepStyle)`
Datum	`AFDate_KeystrokeEx(cFormat), AFDate_FormatEx(cFormat)`
Zeit	`AFTime_Keystroke(tFormat), AFTime_FormatEx(cFormat)`
Speziell	`AFSpecial_Keystroke(psf), AFSpecial_Format(psf)`

Tabelle 9.2 Parameter für die JavaScript-Formatierungsfunktionen

Parameters	***Mögliche Werte***	
nDec	*Anzahl der Dezimalstellen*	
sepStyle	*Art der Dezimaltrennzeichen:*	
	0	*1,234.56*
	1	*1234.56*
	2	*1.234,56*
	3	*1234,56*
negStyle	*Auszeichnung von negativen Zahlen:*	
	0	*Normal*
	1	*Text in rot*
	2	*Text in Klammern*
	3	*Beide*

Tabelle 9.2 Parameter für die JavaScript-Formatierungsfunktionen

Parameters	Mögliche Werte	
strCurrency	*Währungsstring, zum Beispiel* `\u20AC` *für das Eurozeichen*	
bCurrency-Prepend	**false**	*Währungssymbol nicht voranstellen*
	true	*Währungssymbol voranstellen*
cFormat	*Datumsformat-String. Er kann die folgenden Platzhalter sowie die für* `tFormat` *angeführten Zeitformate enthalten:*	
	d	*Tag des Monats*
	dd	*Tag des Monats mit führender Null*
	ddd	*Abkürzung für den Wochentag*
	m	*Monat als Zahl*
	mm	*Monat als Zahl mit führender Null*
	mmm	*Abkürzung für den Monatsnamen*
	mmmm	*vollständiger Monatsname*
	yyyy	*Jahr mit vier Ziffern*
	yy	*Jahr mit den beiden letzten Ziffern*
tFormat	*Zeitformat-String. Er kann die folgenden Platzhalter enthalten:*	
	h	*Stunde (0-12)*
	hh	*Stunde mit führender Null (0-12)*
	H	*Stunde (0-24)*
	HH	*Stunde mit führender Null (0-24)*
	M	*Minuten*
	MM	*Minuten mit führender Null*
	s	*Sekunden*
	ss	*Sekunden mit führender Null*
	t	*'a' oder 'p'*
	tt	*'am' oder 'pm'*
psf	*Beschreibt weitere Formate:*	
	0	*Zipcode*
	1	*Zipcode + 4*
	2	*Telefonnummer*
	3	*Sozialversicherungsnummer*

Validieren der Formularfeld-Eingabe. Das folgende Beispiel fügt einem Formularfeld Javascript hinzu, um prüfen zu können, ob die Eingabe des Benutzers für ein Textfeld dem gewünschte Format *mm/dd/yyyy* entspricht:

```
optlist =
"script={" +
        "// JavaScript code for date mask format MM/DD/YYYY\n" +
        "var re = /^[0-9]{2}\\/[0-9]{2}\\/[0-9]{4}$/\n" +
        "if (event.value !=\"\") {\n" +
        " if (re.test(event.value) == false) {\n" +
        " app.alert ({\n" +
        "    cTitle: \"Incorrect Format\",\n" +
        "    cMsg: \"Please enter date using mm/dd/yyyy format\"\n" +
        " });\n" +
        " }\n" +
        "}\n" +
```

```
"}";
validate_action = p.create_action("JavaScript", optlist);
textfield_font = p.load_font("Helvetica", "unicode", "");
optlist = "action={validate=" + validate_action + "} " +
          "backgroundcolor={gray 0.8} font=" + textfield_font;
p.create_field(llx, lly, urx, ury, "startdate", "textfield", optlist);
```

9.3 PDF mit Geodaten

Cookbook *Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic* `interactive/starter_geospatial`.

9.3.1 Georeferenziertes PDF in Acrobat

PDF 1.7ext3 erlaubt die Einbindung von Geodaten (Weltkoordinaten) in PDF-Seiteninhalte. PDF-Dokumente mit Geodaten können ab Acrobat 9 für verschiedene Zwecke eingesetzt werden (in Acrobat X/XI müssen Sie die Werkzeuggruppe *Analysieren* über die Schaltfläche im oberen Werkzeuge-Bereich aktivieren):

- Anzeige der Positionskoordinaten unter dem Mauszeiger aktivieren: *Werkzeuge, Analysieren, Werkzeug für Geodatenposition*. Sie können die Positionskoordinaten unter dem Mauszeiger kopieren, indem Sie rechts-klicken und *Koordinaten in Zwischenablage kopieren* auswählen;
- Kartenposition suchen: *Werkzeuge, Analysieren, Werkzeug für Geodatenposition*, rechtsklicken Sie und wählen Sie *Position suchen* und geben dann die gewünschten Koordinaten ein;
- Kartenposition markieren: *Werkzeuge, Analysieren, Werkzeug für Geodatenposition*, rechts-klicken Sie und wählen Sie *Position markieren;*
- Abstand, Umfang und Fläche auf geografischen Karten messen: *Werkzeuge, Analysieren, Messwerkzeug;*

Adobe Reader verfügt nur über die ersten beiden oben genannten Möglichkeiten. Verschiedene Einstellungen für Geodaten wie das bevorzugte Koordinatensystem für Positionsanzeigen können Sie über das Menü *Bearbeiten, Voreinstellungen, [Allgemein...], Messen (Geo)* vornehmen.

In PDFlib stehen Ihnen für den Umgang mit Geodaten folgende Funktionen und Optionen zur Verfügung:

- Mit der Option *viewports* von *PDF_begin/end_page_ext( )* können Sie einer Seite georeferenzierte Bereiche zuordnen. Mit *Viewports* lassen sich mit der Option *georeference* unterschiedliche Geodaten für verschiedene Bereiche auf einer Seite festlegen, zum Beispiel mehrere Karten für dieselbe Seite.
- Mit der Option *georeference* von *PDF_load_image( )* können Sie einem Bild ein erdgebundenes Koordinatensystem zuweisen.

9.3.2 Geografische und projizierte Koordinatensysteme

Ein geografisches Koordinatensystem beschreibt die Erde in geografischen Koordinaten, das heißt, in Winkeleinheiten für Längen-und Breitengrade. Ein projiziertes Koordinatensystem kann auf einem geografischen Koordinatensystem aufgesetzt werden und beschreibt die Transformation von geografischen Koordinatenpunkten in ein zweidimensionales (projiziertes) Koordinatensystem. Die resultierenden Koordinaten werden Hochwert und Rechtswert genannt; Angaben in Grad sind für projizierte Koordinatensysteme nicht mehr erforderlich. Während geografische Koordinatensysteme bei GPS und anderen globalen Anwendungen Verwendung finden, sind projizierte Koordinatensysteme für Kartografie und Anwendungen mit mehr oder weniger lokalem Charakter erforderlich.

Aus historischen und mathematischen Gründen ist weltweit eine Vielzahl von verschiedenen Koordinatensystemen in Gebrauch. Sowohl geografische als auch projizier-

te Koordinatensysteme lassen sich mit zwei etablierten Methoden, EPSG und WKT, beschreiben.

EPSG. EPSG ist eine Sammlung tausender Koordinatensysteme, die über numerische Codes referenziert werden. EPSG ist nach der aufgelösten *European Petroleum Survey Group* benannt und wird nun von der Internationalen Vereinigung der Öl- und Gaslieferanten betreut *(International Association of Oil and Gas Producers, OGP)*.

EPSG-Referenzcodes verweisen auf eines der Koordinatensysteme in der EPSG-Datenbank. Die vollständige EPSG-Datenbank können Sie hier herunterladen:

```
www.epsg.org
```

WKT (Well-Known Text). Das System WKT *(Well-Known Text)* ist deskriptiv und besteht aus einer Textspezifikation aller relevanten Parameter eines Koordinatensystems. WKT ist in dem Dokument *OpenGIS® Implementation Specification: Coordinate Transformation Services* spezifiziert, das als *Document 01-009* vom *Open Geospatial Consortium (OGC)* veröffentlicht wurde. Die Spezifikation können Sie hier herunterladen:

```
www.opengeospatial.org/standards/ct
```

WKT wurde auch als ISO 19125-1 standardisiert. Obwohl WKT und EPSG beide in Acrobat verwendet werden können und von PDFlib unterstützt werden, sind in Acrobat nicht alle möglichen EPSG-Codes implementiert. Insbesondere scheinen EPSG-Codes für geografische Koordinatensysteme nicht von Acrobat unterstützt zu werden. In diesem Fall empfehlen wir den Einsatz von WKT. Auf der folgenden Webseite finden Sie die Zuordnung von WKT zu einem bestimmten EPSG-Code:

```
www.spatialreference.org/ref/epsg
```

9.3.3 Beispiele für Koordinatensysteme

Das geografische Koordinatensystem von WGS84 *(World Geodetic System)* bildet die Basis für GPS und andere Anwendungen (zum Beispiel OpenStreetMap). Es kann folgendermaßen in der Unteroption *worldsystem* der Option *georeference* ausgedrückt werden:

```
worldsystem={type=geographic wkt={
GEOGCS["WGS 84",
    DATUM["WGS_1984", SPHEROID["WGS 84", 6378137, 298.257223563]],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.01745329251994328]]
}}
```

Das geografische Koordinatensystem von ETRS *(European Terrestrial Reference System)* ist fast identisch mit dem von WGS84. Es kann folgendermaßen angegeben werden:

```
worldsystem={type=geographic wkt={
GEOGCS["ETRS_1989",
    DATUM["ETRS_1989", SPHEROID["GRS_1980", 6378137.0, 298.257222101]],
    PRIMEM["Greenwich", 0.0],
    UNIT["Degree", 0.0174532925199433]]
}}
```

Hinweis *EPSG-Codes für die Systeme von WGS84 und ETRS werden hier nicht aufgeführt, da Acrobat EPSG-Codes wohl nur für projizierte, aber nicht für geografische Koordinatensysteme unterstützt (siehe unten).*

Beispiele für projizierte Koordinatensysteme. Eine Projektion basiert auf dem zugrundeliegenden geografischen Koordinatensystem. Im folgenden Beispiel legen wir ein projiziertes Koordinatensystem für den Einsatz mit GPS-Koordinaten an.

```
worldsystem={type=projected wkt={
    PROJCS["ETRS_1989_UTM_Zone_32N",
      GEOGCS["GCS_ETRS_1989",
        DATUM["D_ETRS_1989", SPHEROID["GRS_1980", 6378137.0, 298.257222101],
          TOWGS84[0, 0, 0, 0, 0, 0, 0]],
        PRIMEM["Greenwich", 0.0],
        UNIT["Degree", 0.0174532925199433]],
      PROJECTION["Transverse_Mercator"],
      PARAMETER["False_Easting", 500000.0],
      PARAMETER["False_Northing", 0.0],
      PARAMETER["Central_Meridian", 9.0],
      PARAMETER["Scale_Factor", 0.9996],
      PARAMETER["Latitude_Of_Origin", 0.0],
      UNIT["Meter", 1.0]]
}}
```

Der entsprechende EPSG-Code für dieses Koordinatensystem ist 25832. Als Alternative zu WKT kann das obige System auch folgendermaßen über dessen EPSG-Code angegeben werden:

```
worldsystem={type=projected epsg=25832}
```

9.3.4 Einschränkungen für georeferenziertes PDF in Acrobat

Bei der Verarbeitung von georeferenziertem PDF in Acrobat 9/X/XI sind uns folgende Einschränkungen aufgefallen:

- Acrobat scheint EPSG-Codes nur für projizierte, aber nicht für geografische Koordinatensysteme zu unterstützen.
  Lösung: Verwenden Sie statt des EPSG-Codes das entsprechende WKT.
- Geodaten lassen sich nicht an Form XObjects anhängen. Deshalb unterstützt PDFlib die Option *georeference* nicht für *PDF_open_pdi_page( ), PDF_begin_template_ext( )* und *PDF_load_graphics( ),* obwohl dies laut PDF-Referenz funktionieren sollte.
  Lösung für die Erstellung vektorbasierter Karten: Sie können die Geodaten mit der Option *viewports* von *PDF_begin_page_ext( )* zu der Seite hinzufügen.
- Übereinanderliegende Karten: Sie können mehrere bildbasierte Karten auf der selben Seite platzieren. Falls Sie Positionskoordinaten in einem Bereich mit sich überlappenden Karten anzeigen lassen möchten, verwendet Acrobat die Koordinaten der zuletzt platzierten Karte (also der Karte, die auch sichtbar ist). Wenn allerdings beide Image-Handles identisch sind (das heißt mit einem einzigen Aufruf von *PDF_load_image( )* erzeugt wurden, berücksichtigt Acrobat die unterschiedlichen Bildgeometrien nicht mehr: die Koordinaten des ersten Bildes werden fälschlicherweise in den Bereich des zweiten Bildes erweitert, was zu einer falschen Positionsanzeige führt.
  Lösung: Wenn Sie mehrere Instanzen der gleichen bildbasierten Karte auf derselben Seite benötigen, öffnen Sie das Bild mehrfach.

- Das Messwerkzeug für Flächen funktioniert zuverlässig nur bei projizierten Koordinatensystemen, aber nicht bei geografischen Koordinatensystemen.

10 Dokumentenaustausch

10.1 XMP-Metadaten

Als Alternative oder zusätzlich zu Dokument-Infofeldern unterstützt PDFlib das Format XMP *(Extensible Metadata Platform)* für die Angabe von Metadaten. XMP wurde als ISO 16684-1:2012 standardisiert. PDFlib unterstützt XMP auf mehrere Arten, die im Folgenden vorgestellt werden.

Cookbook *Ein einfaches Codebeispiel zu XMP finden Sie im Cookbook-Topic* `interchange/embed_xmp`.

Am häufigsten wird XMP verwendet, um Metadaten an das gesamte Dokument anzuhängen. Außer Metadaten auf Dokumentebene kann XMP für Seiten, Schriften, ICC-Profile, Bilder, Grafiken, Templates und importierte PDF-Seiten übergeben werden. Dies kann mit der Option *metadata* verschiedener Funktionen durchgeführt werden, zum Beispiel:

```
metadata={filename=info.xmp inputencoding=winansi}
```

Die Option *metadata* muss einen XMP-Stream beschreiben, der alle oder einen Teil der Metadaten enthält. Die vom Benutzer übergebenen XMP-Metadaten überprüft PDFlib auf die Einhaltung von XML- und XMP/RDF-Regeln. Bei PDF/A gelten für benutzerdefinierte XMP-Properties zusätzliche Regeln; siehe Abschnitt 11.3.8, »XMP-Dokumentmetadaten für PDF/A«, Seite 334.

Interne und reservierte XMP-Properties. PDFlib erzeugt intern verschiedene XMP-Properties, zum Beispiel *CreationDate*. Andere XMP-Properties sind erforderlich, um die Konformität zu verschiedenen PDF-Standards wie PDF/A und PDF/X zu signalisieren. Interne Properties sowie Properties zur Beschreibung der Standardkonformität können nicht mit XMP überschrieben werden, das vom Benutzer übergeben wird.

Automatische XMP-Synchronisierung für Dokument-Infofelder. Wenn die Option *autoxmp* von *PDF_begin/end_document( )* auf *true* gesetzt ist, synchronisiert PDFlib sowohl an *PDF_set_info( )* übergebene Dokument-Infofelder als auch verschiedene intern erzeugte Einträge wie *CreationDate* mit den entsprechenden Einträgen in den XMP-Metadaten auf Dokumentebene.

Dokument-Infofelder, die einer bekannten Property in einem der vordefinierten XMP-Schemas entsprechen, werden in das entsprechende Schema eingefügt. Unbekannte Infofelder werden in der Regel in das erweiterte PDF-Schema *(pdfx)* platziert, werden aber bei PDF/A ignoriert.

Klonen von XMP-Metadaten. Wenn mehrere oder alle Seiten eines PDF-Dokuments importiert werden, sollten Sie die XMP-Metadaten klonen, sofern in der Eingabe vorhanden. Mit dem folgenden Codefragment lassen sich XMP-Metadaten klonen:

```
if (p.pcos_get_string(indoc, "type:/Root/Metadata").equals("stream"))
{
        xmp = p.pcos_get_stream(indoc, "", "/Root/Metadata");
        p.create_pvf("/xmp/document.xmp", xmp, "");
        optlist += " metadata={filename=/xmp/document.xmp}";
```

```
    }

    p.end_document(optlist);
    p.delete_pvf("/xmp/document.xmp");
```

10.2 Web-optimiertes (linearisiertes) PDF

PDFlib ist in der Lage, PDF-Dokumente zu linearisieren (linearisiertes PDF wird auch *Schnelle Webanzeige* genannt). Bei der Linearisierung werden die Objekte in der PDF-Datei umgeordnet und Informationen hinzugefügt, die dem schnelleren Zugriff dienen.

Während nicht-linearisierte PDF-Dokumente vollständig zum Client übertragen werden müssen, kann ein Webserver linearisierte PDF-Dokumente seitenweise mit einem Verfahren names Byteserving transferieren. Damit kann Acrobat (als Browser-Plugin) ein PDF-Dokument auch teilweise abrufen. Die erste Dokumentseite wird Benutzern dann unverzüglich angezeigt, ohne dass sie die vollständige Übertragung des Dokuments vom Server abwarten müssen. Dies erhöht die Benutzerfreundlichkeit.

Beachten Sie, dass nicht PDFlib, sondern der Webserver die PDF-Daten an den Browser übergibt. PDFlib bereitet die PDF-Dateien lediglich zum Byteserving vor. Zum Byteserving von PDF sind folgende Voraussetzungen zu erfüllen:

- Das PDF-Dokument muss linearisiert werden. Dies lässt sich mit der Option *linearize* in *PDF_begin_document( )* wie folgt bewerkstelligen:

  ```
  p.begin_document(outfilename, "linearize");
  ```

  In Acrobat können Sie in den Dokumenteigenschaften nachsehen, ob eine Datei linearisiert ist *(Schnelle Webanzeige: Ja)*.
- Der Webserver muss Byteserving unterstützen. Das zugrunde liegende Byte-Range-Protokoll ist Bestandteil von HTTP 1.1 und somit in allen üblichen Webservern implementiert.
- Acrobat muss als Browser-Plugin installiert sein und in Acrobat müssen Sie seitenweises Herunterladen aktiviert haben *(Bearbeiten, Voreinstellungen..., [Allgemein...,] Internet, Schnelle Webanzeige zulassen)*. Diese Einstellung ist standardmäßig aktiviert.

Je größer eine PDF-Datei ist (in Seiten oder MB), desto mehr wird sie bei der Übertragung über das Web von der Linearisierung profitieren.

Linearisierung kleiner Dateien. Da Linearisierung für die Verbesserung der web-basierten Anzeige von großen PDF-Dokumenten gedacht ist, macht sie bei einseitigen Dokumenten nicht viel Sinn. Acrobat behandelt kleine linearisierte Dokumente nicht immer als linearisiert. Beispielsweise gelten in Acrobat ab Version 9 Dokumente kleiner als 4 KB als nicht linearisiert, unabhängig davon, ob sie tatsächlich linearisiert sind oder nicht.

Temporärer Speicher für die Linearisierung. PDFlib muss das Dokument erst vollständig erstellen, bevor es in einem nachfolgenden eigenen Verarbeitungsschritt linearisiert wird. Dafür benötigt PDFlib temporär zusätzlichen Speicher, der in etwa der Größe des generierten Dokuments (ohne Linearisierung) entspricht. Je nach Einstellung der Option *inmemory* für *PDF_begin_document( )* speichert PDFlib die für die Linearisierung erforderlichen Daten entweder im Hauptspeicher oder in einer temporären Datei auf der Festplatte.

10.3 Grundlagen von Tagged PDF

Tagged PDF ist eine Voraussetzung für die ISO-Standards PDF/UA, PDF/A-1a, PDF/A-2a und PDF/A-3a, für Section 508 in den USA, für BITV in Deutschland und viele andere Verordnungen. Tagged PDF erweitert PDF um Informationen zur Dokumentstruktur mit den folgenden Vorteilen:

- Barrierefreiheit *(Accessibility)*: Tagged PDF ist für Menschen mit einer Beeinträchtigung zugänglich, zum Beispiel über die Sprachausgabe von Acrobat oder spezielle Screenreader-Software (siehe Abbildung 10.1).
- Umfließen von Text: Seiteninhalte werden zur Anpassung an die aktuelle Fenster- oder Bildschirmgröße automatisch neu formatiert. Irrelevante Seiteninhalte (sogenannte Artefakte) werden dabei nicht angezeigt. Im Umfließen-Modus können Sie die Anzeige vergrößern, ohne die ganze Zeit Seiten umblättern zu müssen. Umfließen von Text ist wichtig für die Barrierefreiheit und verbessert die Seitenanzeige auf mobilen Geräten mit kleinen Bildschirmen.
- Zuverlässige Textextraktion und -konvertierung in andere Dokumentformate: die Konvertierung von Tagged PDF in andere Formate wie RTF, XML oder HTML führt zu genauerer Ausgabe.

PDF/UA erweitert Tagged PDF um zusätzliche Anforderungen für Dokument-Tags. Falls Sie zugängliche PDF-Dokumente erstellen wollen, sollten Sie die erweiterten Regeln für PDF/UA beachten; siehe Abschnitt 11.6, »PDF/UA für Barrierefreiheit«, Seite 356.
Wenn Sie nicht alle PDF/UA-Anforderungen erfüllen können (weil Sie zum Beispiel Dokumente aus bestehendem, nicht PDF/UA-konformem PDF montieren müssen) sollten Sie den PDF/UA-Modus deaktivieren und so viele PDF/UA-Regeln wie möglich einhalten.

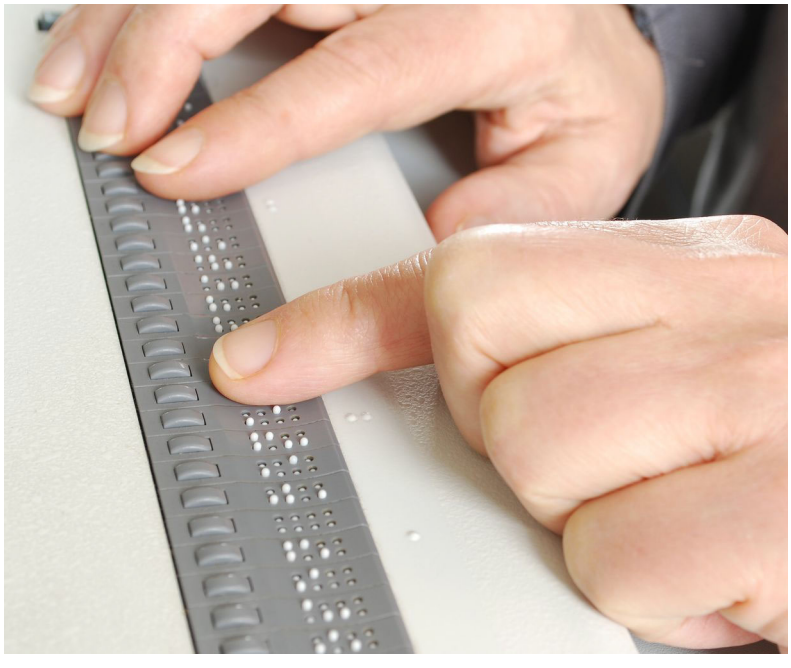


Abb. 10.1
Ein Screenreader erfasst Text auf dem Bildschirm und gibt ihn auf ein Braille-Gerät aus

Cookbook *Codebeispiele zur Erzeugung von Tagged PDF finden Sie in der Kategorie* `pdfua` *des PDFlib Cookbook. Alle Beispiele für Tagged PDF im Cookbook erzeugen PDF/UA-konforme Ausgabe.*

10.3.1 Der logische Strukturbaum (Strukturhierarchie)

Tagged PDF kann nur erzeugt werden, wenn der Client Informationen über die interne Dokumentstruktur liefert und sich bei der Generierung der PDF-Ausgabe an bestimmte Regeln hält. Zur Erzeugung von Tagged PDF muss die Dokumentoption *tagged* auf *true* gesetzt werden, und die Option *lang* wird empfohlen:

```
if (p.begin_document("tagged.pdf", "tagged=true lang=en") == -1) {
        throw new Exception("Error: " + p.get_errmsg());
```

Die logische Struktur in einem Tagged PDF wird von einer Hierarchie von Elementen beschrieben, der sogenannten Strukturhierarchie, auch logische Struktur oder Tagbaum genannt. Ausgehend von der obersten Ebene *(Stammelement*, oft auch als Element *Document* bezeichnet), besteht die Strukturhierarchie aus einer beliebigen Anzahl von Ebenen. Auf jeder dieser Ebenen kann ein Element null, ein oder mehrere Objekte der folgenden Typen enthalten:

- Weitere Strukturelemente: das Element *Document* kann zum Beispiel mehrere Elemente vom Typ *Article* enthalten und jedes Element *Article* wiederum beliebig viele Elemente vom Typ *P* (*Paragraph*, Absatz).
- Direkter Inhalt: Seiteninhalte wie markierter Text und Grafiken, aus importierten Bildern erzeugte XObjects usw. Diese Objekte stellen den zu einem Strukturelement gehörenden grafischen Inhalt dar.
- Interaktive Objekte, zum Beispiel Anmerkungen und Formularfelder.

Strukturhierarchie in Acrobat. In Acrobat X/XI können Sie die Tag-Namen und die Strukturhierarchie folgendermaßen anzeigen lassen:

- Wählen Sie *Anzeige, Ein-/Ausblenden, Navigationsfenster, Tags* (siehe Abbildung 10.2).

Erstellen des Strukturbaums. Strukturelemente lassen sich mit der Funktion *PDF_begin_item( )* erzeugen, die paarweise mit *PDF_end_item( )* benutzt werden muss. Beispielsweise erstellt das Codefragment unten eine Hierarchie aus einem Element Abschnitt *(Sect)*, welches die Elemente Überschrift *(H1)* und Absatz *(P)* enthält. Die Hierarchieebenen sind durch Einrückung dargestellt:

```
/* Erstellen eines Strukturelements vom Typ "Sect" (Abschnitt) */
id_sect = p.begin_item("Sect", "Title={Past and Future}");

        /* Erstellen eines Strukturelements vom Typ "H1" (Überschrift) */
        id_h1 = p.begin_item("H1", "Title={Company History}");
            p.fit_textline(...);
        p.end_item(id_h1);

        /* Erstellen eines Strukturelements vom Typ "P" (Absatz) */
        id_p = p.begin_item("P", "");
            p.fit_textline(...);
        p.end_item(id_p);

/* Beenden von "Sect" */
p.end_item(id_sect);
```

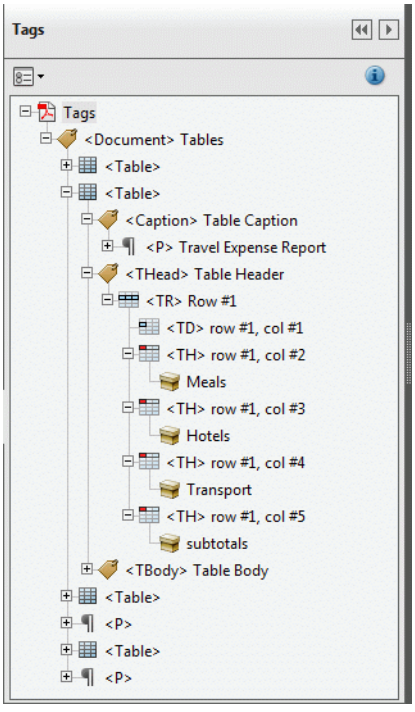


Abb. 10.2
Acrobat-Tagfenster mit dem logischen Strukturbaum eines Dokuments

Standardmäßig werden Strukturelemente als Unterelement des aktuell aktiven Elements eingefügt, und zwar hinter allen anderen eventuell vorhandenen Unterelementen. Sofern die Elemente in logischer Reihenfolge erstellt werden, ergibt sich daraus die korrekte Baumstruktur. Für weitere Informationen hierzu, siehe Abschnitt 10.4.4, »Erzeugung von Seiteninhalt in abweichender Reihenfolge«, Seite 308.

An die Strukturelemente können in Optionen ein oder mehrere Attribute übergeben werden, zum Beispiel die Option *lang* für die Angabe der natürlichen Sprache des Content-Elements oder *Alt* für Alternativtext von Bildern. Die Anzahl der verfügbaren Optionen hängt vom Typ des Strukturelements ab.

Vereinfachtes Tagging. In vielen Fällen kann der Inhalt eines Strukturelements mit einem einzigen PDFlib-Funktionsaufruf zur Platzierung erstellt werden, was zu der typischen Sequenz von *PDF_begin_item( )/PDF_fit_*( )/PDF_end_item( )* führt. Diese Sequenz kann verkürzt werden, indem das Platzieren von Inhalt und das Anbringen von Tags in einem einzigen Funktionsaufruf zusammengefasst werden. Dies wird mit der Option *tag* bewerkstelligt, die von den meisten Funktionen zum Platzieren von Seiteninhalt unterstützt wird. Das Codefragment oben kann unter Angabe der Option *tag* auf die Funktionen zum Platzieren von Text und Bild reduziert werden:

```
/* Erstellen eines Strukturelements vom Typ "Sect" (Abschnitt) */
id_sect = p.begin_item("Sect", "Title={Past and Future}");

        /* Erstellen eines Strukturelements vom Typ "H1" (Überschrift) */
        p.fit_textline(..., "tag={tagname=H1 Title={Company History}}");

        /* Erstellen eines Strukturelements vom Typ "P" (Absatz) */
        p.fit_textline(..., "tag={tagname=P}");
```

```
/* Beenden von "Sect" */
p.end_item(id_sect);
```

Im Einzelnen arbeitet die Option *tag* wie folgt:

- Für den erstellten Inhalt wird ein neues Strukturelement erzeugt und vor Ende des Aufrufs abgeschlossen. Die folgenden Fälle sind Ausnahmen zu dieser Regel:
  - Ein mit der Option *tag* von *PDF_begin_document( )* erzeugtes Strukturelement wird mit *PDF_end_document( )* beendet.
  - Wenn mit der Option *tag* von *PDF_begin_item( )* mehrere Tags übergeben werden, werden alle mit dem entsprechenden Aufruf von *PDF_end_item( )* beendet.
- *PDF_fit_table( ):* Bei *tagname=Table* oder bei einem Tag-Namen mit einer Rollenzuordnung zu *Table* in *PDF_fit_table( )* erzeugt PDFlib die erforderlichen Tabellen-Tags (siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300). Automatisch generierte Tags vom Typ *TH* und *TD* für Tabellenzellen können mit der Option *tag* von *PDF_add_table_cell( )* weiter spezifiziert werden.
- *PDF_fit_textflow( ):* die vollständige Textflow-Instanz bildet das neue Strukturelement.
- Das erzeugte Element ist ein Unterelement des aktuell aktiven Elements oder des Elements, das in der Option *parent* übergeben wurde.
- Gruppierungselemente können nur mit *PDF_begin_item( )* erstellt werden, jedoch nicht mit der Option *tag* in allen Funktionen außer *PDF_begin_document( ).*

In manchen Fällen mag es nötig oder gewünscht sein, verschachtelte Tags durch vereinfachtes Anbringen von Tags zu erstellen; für ein Beispiel hierzu siehe »Links und andere Arten von Anmerkungen«, Seite 303.

Da das vereinfachte Anbringen von Tags nicht die ID des erzeugten Strukturelements offenlegt, kann es nicht mit *PDF_activate_item( )* oder der Unteroption *parent* der Option *tag* verwendet werden, da diese eine ID für das Strukturelement erwarten.

10.3.2 Standard- und benutzerdefinierte Elementtypen

Standard-Elementtypen. PDF bietet eine Reihe von Standard-Elementtypen zur Unterstützung einer Vielzahl von Dokumentklassen. Diese Standard-Elementtypen werden von PDFlib vollständig unterstützt, siehe Tabelle 10.1. Die Tabelle soll Ihnen die Auswahl der passenden Tags erleichtern und enthält außerdem die Unterscheidung zwischen BLSE *(Block level structure elements)* und ILSE *(Inline level structure elements)*. Für weitere Informationen hierzu siehe Abschnitt »Strukturelemente vom Typ Block-Level und Inline-Level«, Seite 288.

Gruppierungselemente bilden Container für andere Strukturelemente. Sie können allerdings keinen direkten Seiteninhalt enthalten. Besteht das PDF aus einem komplexen Dokument, sollte als oberstes Element des Strukturbaums ein Element vom Typ *Document* verwendet werden. Besteht das PDF aus einem Dokumentfragment, sollte einer der Elementtypen *Part, Art, Sect* oder *Div* als oberstes Element des Strukturbaums verwendet werden. Das oberste Element (Stammelement) kann einfach in der Option *tag* von *PDF_begin_document( )* angegeben werden.

Pseudo-Elementtypen erzeugen kein Strukturelement, sondern versehen den Inhalt mit bestimmten Eigenschaften und werden meist zur Auszeichnung von Artefakten eingesetzt (siehe Abschnitt 10.3.3, »Artefakte«, Seite 291).

Tabelle 10.1 Standard-Elementtypen (Tags) bei Tagged PDF und zusätzliche Pseudo-Elementtypen von PDFlib

Typ	Beschreibung
Gruppierungselemente (Container)	
Document	*Vollständiges Dokument; Empfohlen als oberstes Element (Stammelement) des Strukturbaums*
Part	*(Teil) Grobe Einteilung eines Dokuments als Container für kleinere Inhaltseinheiten wie Textkörper (Article) oder Abschnittselemente (Section)*
Art	*(Artikel) Inhaltlich weitgehend in sich abgeschlossener Textkörper; Artikel sollten keine weiteren Artikel enthalten.*
Sect	*(Section, Abschnitt) Container-Element für die Gruppierung von inhaltlich zusammenhängenden Elementen; ein Abschnitt kann zum Beispiel eine Überschrift, verschiedene einleitende Absätze und weitere untergeordnete Abschnitte enthalten.*
Div	*(Division, Kapitel) Generisches Element auf Blockebene oder eine Gruppe solcher Elemente*
BlockQuote	*(Blockzitat) Text bestehend aus einem oder mehreren Absätzen, die von einem anderen Verfasser stammen als der unmittelbar ihn umgebende Haupttext*
Caption	*(Tabellen-, Bildunterschrift) Kurzer Text, der eine Tabelle oder Abbildung beschreibt*
TOC	*(Table of contents, Inhaltsverzeichnis) Strukturierte Liste mit Einträgen von Inhaltselementen (Elementtyp* `TOCI`*) und/oder weiteren untergeordneten Inhaltsverzeichnissen* (`TOC`)
TOCI	*(Table of contents item, Inhaltsverzeichnis-Eintrag) Listeneintrag in einem Inhaltsverzeichnis. Er sollte ein passendes Element vom Typ* `Link` *enthalten.*
Index	*(Index) Folge von Einträgen, die aus einem Stichwort und Referenzelementen bestehen, die auf das Vorkommen des Stichworts im Hauptkörper des Dokuments verweisen*
NonStruct	*(Nicht strukturrelevantes Element) Gruppierungselement ohne eigentliche Bedeutung für die Struktur, es bildet lediglich einen Container für andere Strukturelemente; nicht zu verwechseln mit dem Element vom Typ* `Div`*, da es weder interpretiert noch in andere Dokumentformate exportiert wird. Seine einzelnen Unterelemente werden dagegen normal verarbeitet.*
Private	*(Private element; anwendungsspezifisches Element) Gruppierungselement mit anwendungsspezifischem Inhalt ohne eigentliche Bedeutung für die Struktur. Weder das Element* `Private` *noch seine Unterelemente werden interpretiert oder in andere Dokumentformate exportiert.*
Überschriften- und Absatzelemente (BLSE)	
H	*(Heading, Überschrift; bei PDF/UA nur zusammen mit* `structuretype=strong`*) Bezeichnung für eine Unterteilung des Dokumentinhalts. Das Element* `H` *sollte das erste Element im übergeordneten Element sein. Elemente vom Typ* `H` *sind für hierarchische Verschachtelung bestimmt.*
H1...H6	*Überschriften einer bestimmten Ebene. Sie sollten bei Anwendungen verwendet werden, die Abschnitte nicht hierarchisch verschachteln und deshalb die Überschriften-Ebene nicht anhand der Verschachtelungstiefe bestimmen können.*
H7, H8...	*Zusätzliche Überschriften-Ebenen, falls* `H1...H6` *nicht ausreichen.* `H7` *usw. sind kein Bestandteil von ISO 32000-1, sondern wurden für PDF/UA-1 eingeführt.*
P	*(Paragraph, Absatz) Generisches Absatzelement; kleinere inhaltlich zusammenhängende Einheit von Text, der keine Überschrift ist.*
Label- und Listenelemente (BLSE)	
L	*(Liste) Folge von Einträgen ähnlicher Bedeutung oder Relevanz*
LI	*(List item, Listeneintrag) Einzelnes Element einer Liste*

Tabelle 10.1 Standard-Elementtypen (Tags) bei Tagged PDF und zusätzliche Pseudo-Elementtypen von PDFlib

Typ	Beschreibung
Lbl	*(Label) Name oder Nummer, die einen Eintrag kennzeichnet und von anderen Einträgen in derselben Liste oder einer Gruppe ähnlicher Einträge unterscheidet; in einer Aufzählung oder in einer nummerierten Liste beispielsweise enthält das Label das Aufzählungszeichen bzw. die Aufzählungsnummer mit dem zugehörigen Satzzeichen.*
LBody	*(List body, Listenkörper) Eigentlicher Inhalt eines einzelnen Listeneintrags*
Tabellenelemente (alle Tabellen-Tags lassen sich automatisch erzeugen, siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300)	
Table	*(BLSE) Zweidimensionale Anordnung von rechteckigen Zellen, eventuell mit komplexer Unterstruktur*
TR	*(Table row, Tabellenzeile) Zeile mit Überschriften oder Daten in einer Tabelle*
TH	*(Table header cell, Tabellen-Überschriftenzelle) Tabellenzelle mit Überschriftentext, der eine oder mehrere Zeilen oder Spalten der Tabelle beschreibt*
TD	*(Table data cell, Tabellen-Datenzelle) Tabellenzelle mit Daten, die Teil des Tabelleninhalts sind*
THead	*(Table header row group, PDF 1.5) Gruppe von Kopfzeilen in einer Tabelle*
TBody	*(Table body row group„ PDF 1.5) Gruppe von Tabellenzeilen, die den Tabellenkörper oder einen Teil davon bilden*
TFoot	*(Table footer row group, PDF 1.5) Gruppe von Tabellenzeilen, die den Tabellenfuß bilden*
Inline-Level-Elemente (ILSE, können mit der Option inline=false zu BLSE umgewandelt werden)	
Span	*(Nicht zu verwechseln mit Tabellenzellen, die mehrere Zeilen oder Spalten überspannen) Beliebiges Textsegment, dem keine bestimmten Eigenschaften zugeordnet sind*
Quote	*(Quotation, Zitat) Textsegment, das von einem anderen Verfasser stammt als der umgebende Text. Der zitierte Text steht direkt im Text innerhalb des umgebenden Absatzes; nicht zu verwechseln mit dem Blockzitat* `BlockQuote`*, bei dem es sich um einen oder mehrere ganze Absätze handelt.*
Note	*Zusatztext wie zum Beispiel eine Fuß- oder Endnote, auf die im Haupttext verwiesen wird. Diesem Element kann ein Element vom Typ* `Lbl` *untergeordnet sein.*
Reference	*Verweis auf Text an anderer Stelle im Dokument*
BibEntry	*(Bibliography entry, Eintrag für Quellenverzeichnis) Quellenangabe für ein Zitat. Diesem Element kann eine Element vom Typ* `Lbl` *untergeordnet sein.*
Code	*Computerprogrammtext*
Elemente für interaktive Funktionen (ILSE) (siehe Abschnitt 10.4.2, »Interaktive Elemente«, Seite 303)	
Link	*Verknüpfung zwischen einem Teil des ILSE-Inhalts und einer oder mehreren entsprechenden Link-Anmerkungen*
Annot	*(Annotation, Anmerkung; PDF 1.5) Verknüpfung zwischen einem Teil des ILSE-Inhalts und einer oder mehreren entsprechenden PDF-Anmerkungen. Es soll für alle Anmerkungen verwendet werden, sofern* `Link` *oder* `Form` *nicht besser passen.*
Form	*Interaktives Formularfeld*
Elemente zur Illustration (ILSE)	
Figure	*(Abbildung) Grafik oder grafische Darstellung*
Formula	*Mathematische Formel; dieser Elementtyp definiert ein Inhaltselement als Formel. Auch wenn die Formel als Bild dargestellt ist, muss das Element* `Formula` *verwendet werden (und nicht das Element* `Figure`*).*

Tabelle 10.1 Standard-Elementtypen (Tags) bei Tagged PDF und zusätzliche Pseudo-Elementtypen von PDFlib

Typ	*Beschreibung*
Elemente für japanisches Ruby und Warichu (ILSE; PDF 1.5)	
Ruby	*Seitliche Anmerkung in kleinerer Schriftgröße, die neben dem Haupttext steht und sich auf diesen bezieht. Das Element* `Ruby` *dient als Wrapper um die gesamte Ruby-Einheit.*
RB	*(Ruby base text, Ruby-Haupttext) Haupttext, auf den sich die Ruby-Anmerkung bezieht*
RT	*(Ruby annotation text, Ruby-Anmerkung) Seitliche Anmerkung in kleinerer Schriftgröße, die neben dem Haupttext platziert werden soll*
RP	*(Ruby punctuation, Ruby-Interpunktion) Interpunktion, die eine Ruby-Anmerkung umgibt, wenn diese nicht korrekt im Ruby-Stil formatiert werden kann und deshalb als normaler Kommentar formatiert wird oder wenn sie als Warichu formatiert ist*
Warichu	*(Warichu) Kommentar oder Anmerkung in kleinerer Schriftgröße, der auf zwei Zeilen formatiert und auf Höhe der zugehörigen Textzeile formatiert wird. Das Element wird direkt hinter den Haupttext platziert, auf den es sich bezieht (inline). Das Element* `Warichu` *dient als Wrapper um die gesamte Warichu-Einheit.*
WT	*(Warichu-Text) Warichu-Kommentar in kleinerer Schriftgröße, der auf zwei Zeilen formatiert und zwischen den umgebenden* `WP`*-Elementen platziert wird.*
WP	*(Warichu-Interpunktion) Interpunktion, die den Text vom Typ* `WT` *umgibt*
Pseudo-Elementtypen	
Artifact	*Artefakt, vom eigentlichen Seiteninhalt zu unterscheiden (siehe Abschnitt 10.3.3, »Artefakte«, Seite 291).*
ASpan	*(Accessibility-Span; in PDF ausgegeben als* `Span`*, nicht zu verwechseln mit dem Inline-Level-Element* `Span`*) Inhalte, die zu keinem Strukturelement gehören oder nur einen Teil eines Strukturelements darstellen, können mit Properties für Barrierefreiheit versehen werden. Das Pseudo-Element* `ASpan` *wird als* `Span` *mit einem Attribut wie* `Alt`, `ActualText`, `Lang` *oder* `E` *in die PDF-Ausgabe geschrieben.* `ASpan` *ist keinem Strukturelement zugeordnet.*
ReversedChars	*(Nicht empfohlen) Text in linksläufiger Schrift mit umgekehrten Zeichen*
Clip	*(Nicht empfohlen) Folge von markierten Beschneidungspfaden. Dies ist eine Folge von Beschneidungspfaden oder Text im Textdarstellungsmodus 7, jedoch ohne sichtbare Grafik oder PDF_save( ) / PDF_restore( ).*

Verschachtelungsregeln für Strukturelemente. Bei der Erstellung von Strukturelementen müssen verschiedenen Regeln eingehalten werden, siehe Tabelle 10.2. Die Regeln gelten für die angeführten Standard-Elementtypen sowie benutzerdefinierte Elementtypen, für die eine Rollenzuordnung auf den jeweiligen Standardtyp erstellt wird (siehe Abschnitt »Benutzerdefinierte Elementtypen und Rollenzuordnungen«, Seite 291). Zusätzliche Regeln gelten für PDF/UA-1 (siehe Abschnitt 11.6, »PDF/UA for Universal Accessibility«, Seite 330).

Die Verschachtelungsregeln für neue Strukturelemente lassen sich mit der Option *checktags* von *PDF_begin_document( )* deaktivieren oder lockern. Dies ist jedoch nicht empfehlenswert, da es zu einer ungültigen Strukturhierarchie führen kann. Diese Option ist zur Unterstützung bei der Migration von älteren Anwendungen gedacht. Einige Regeln in Tabelle 10.2 sind als »strenge Regeln« gekennzeichnet. Die Option *checktags=relaxed* verstärkt alle Regeln außer den strengen Regeln.

Strukturelemente vom Typ Block-Level und Inline-Level. Bei Block-Level-Elementen (BLSE) handelt es sich um auf einer Seite platzierte inhaltliche Segmente, wohingegen es sich bei Inline-Level-Elementen (ILSE) um kleinere inhaltliche Fragmente mit beson-

Tabelle 10.2 Verschachtelungsregeln für Tags bei PDF_begin_item( ) und der Option tag *verschiedener Funktionen*

Element	*Regel*
Direkter Inhalt	*Die folgenden Elemente können direkten Seiteninhalt enthalten, also Text, Bilder oder Vektorgrafik:* H, H1, ..., H6, H7, ... P Lbl, LBody TH, TD Span, Quote, Note, Reference, BibEntry, Code Link, Annot Figure, Formula RB, RT, RP, WT, WP Artifact, ASpan, ReversedChars, Clip *Alle anderen Elemente benötigen ein Strukturelement, in dem direkter Seiteninhalt aufgenommen werden kann. PDFlib löst beim Versuch, Seiteninhalte direkt in der Hierarchie hinzuzufügen, eine Exception aus.*
Gruppierungselemente	*Gruppierungselemente dürfen weder direkten Inhalt noch* ASpan *oder ILSEs als Unterelement enthalten; bevor Inhalt hinzugefügt werden kann, muss also erst ein BLSE erstellt werden:* Document, Part, Art, Sect, Div, BlockQuote, Caption, TOC, TOCI, Index, NonStruct, Private *Als Ausnahme zu dieser Regel dürfen die folgenden ILSEs als Unterelemente in Gruppierungselementen verwendet werden:* Figure, Formula, Form, Link, Annot
Block-Level-Elemente	*Die folgenden Block-Level-Elemente dürfen keinen direkten Seiteninhalt enthalten, sondern benötigen ein passendes Gruppierungselement oder ein zuvor erstelltes BLSE:* L, LI, Table, TR, THead, TFoot, TBody *Strenge Regel: das Element* P *darf keine Gruppierungselemente enthalten.*
Pseudo- und Inline-Elemente	*Pseudo-Elemente (also* Artifact, ASpan, ReversedChars, Clip*) und die folgenden ILSEs dürfen keine untergeordneten Elemente enthalten:* Code, BibEntry, Note, Quote, Reference, Span *Bei* inline=false *sind untergeordnete Elemente erlaubt.*
Tabellenelemente	*Elemente vom Typ* Table *können ein oder mehrere Elemente vom Typ* TR *enthalten oder optional das Element* THead *gefolgt von einem oder mehreren Elementen* TBody *oder* TFoot*. Außerdem dürfen Elemente vom Typ* Table *das Element* Caption *als erstes oder letztes untergeordnetes Element haben.* *Elemente vom Typ* TR *können die Elemente* TH *und* TD *enthalten.* *Elemente vom Typ* TH *und* TD *dürfen die Elemente* TR, TH, TD, THead, TBody *und* TFoot *nicht enthalten.* *Elemente vom Typ* THead, TBody *und* TFoot *dürfen nur* TR*-Elemente enthalten und dürfen nur* Table *als übergeordnetes Element haben.* TR *darf nur* Table, THead, TBody *oder* TFoot *als übergeordnetes Element haben.*
Listenelemente	*Elemente vom Typ* L *können optional das Element* Caption *und ein oder mehrere Elemente vom Typ* LI *enthalten.* *Elemente vom Typ* LI *können ein oder mehrere Elemente* Lbl *oder* LBody *oder beide enthalten.* LI *muss als übergeordnetes Element* L *haben.* LBody *darf nur* LI *als übergeordnetes Element haben.*
Inhaltsverzeichnis (TOC)	*Elemente vom Typ* TOC *können optional das Element* Caption *als erstes untergeordnetes Element sowie ein oder mehrere Elemente* TOCI *und* TOC *enthalten (auch kombiniert).* *Elemente vom Typ* TOCI *dürfen nur die Elemente* Lbl, Reference, NonStruct, P *und* TOC *enthalten.* TOCI *muss* TOC *als übergeordnetes Element haben.*
Label-Elemente	*Dem Element* Lbl *dürfen nur die Elemente* LI, TOCI, BibEntry *oder* Note *übergeordnet sein.*

Tabelle 10.2 Verschachtelungsregeln für Tags bei PDF_begin_item( ) und der Option tag *verschiedener Funktionen*

Element	*Regel*
interaktive Elemente: Links, Formularfelder und Anmerkungen	*Folgende Elementtypen sind weder bei der Option* tag *von PDF_create_field( ) und PDF_create_annotation( ) noch als übergeordnete Elemente von* Link, Annot *und* Form *erlaubt: Tabellenelemente, ILSEs, Ruby- und Warichu-Elemente, Pseudo-Elemente.* Annot *kann nicht verschachtelt werden.* Link *kann nicht verschachtelt werden.* Form*-Elemente dürfen nur das Element* OBJR *enthalten, welches durch PDF_create_field( ) automatisch erzeugt wird.*
japanisches Ruby und Warichu	Ruby *darf nur* RB, RT *und* RP *als untergeordnete Elemente enthalten.* *Warichu darf nur* WT *und* WP *als untergeordnete Elemente enthalten.*

derer Formatierung oder einem besonderen Verhalten handelt. BLSEs und ILSEs sind in Tabelle 10.1 jeweils gekennzeichnet. Die Unterscheidung ist für einige Tagging-Optionen relevant; für weitere Informationen siehe die PDFlib-Referenz oder Tabelle 10.3, in der weitere Unterschiede aufgeführt sind.

Mit der Option *inline* von *PDF_begin_item( )* oder der Option *tag* kann der Status der Elementtypen *BibEntry, Code, Note, Quote, Reference* und *Span* von regulär auf inline gesetzt werden oder umgekehrt. Ein Accessibility-Span sollte zum Beispiel regulär *(inline=false)* sein, wenn ein Absatz, der sich über mehrere Seiten erstreckt, verschiedene Sprachen enthält. Alternativ dazu könnte das Element beendet und auf der nächsten Seite ein neues Element begonnen werden. Inline-Elemente müssen auf der Seite, auf der sie begonnen wurden, auch geschlossen werden.

Tabelle 10.3 Reguläre und Inline-Elemente

	Reguläre Elemente	***Inline-Elemente***
betroffene Elemente	*Gruppierungselemente und BLSEs*	*ILSEs und Pseudo-Elemente*
Status regulär/inline änderbar	*nein*	*nur* Code, BibEntry, Note, Quote, Reference, Span
Teil des Dokumentstrukturbaums	*ja*	*nein, mit folgenden Ausnahmen:* Figure, Formula, Form, Link, Annot
kann sich über mehrere Seite erstrecken	*ja*	*nein*
von anderen Elementen unterbrechbar	*ja*	*nein*
kann mit PDF_activate_item( ) aktiviert werden	*ja*	*nein*
beliebig verschachtelbar	*ja*	*nur mit anderen Inline-Elementen*

Geltungsbereich und Seitengrenzen. Die meisten Strukturelemente können nur im Geltungsbereich *page* erzeugt werden. Gruppierungselemente können dagegen auch im Geltungsbereich *document* erzeugt werden.

Inline- und Pseudoelemente müssen geschlossen werden, bevor die Seite beendet oder suspendiert wird. Dagegen können andere Elementtypen über die Seite hinausreichen.

Benutzerdefinierte Elementtypen und Rollenzuordnungen. Zusätzlich zu den Standard-Elementtypen (siehe Tabelle 10.1) können Sie Namen für Elementtypen selbst definieren und diese verwenden. Sie werden üblicherweise zur Lokalisierung von Elementnamen (zum Beispiel Deutsch *Abbildung* für *Figure)* oder für anwendungsspezifische Elementnamen verwendet (zum Beispiel *Normal* für *P)*. Um die Wiederverwendbarkeit von Dokumenten mit benutzerdefinierten Namen von Elementtypen zu erhöhen, müssen diese einem passenden Standard-Elementtyp zugeordnet werden. Dies nennt man Rollenzuordnung. Sie können einen Standard-Elementtyp auch einem anderen Standard-Elementtyp zuordnen, um seine Semantik zu ändern. Benutzerdefinierte Elementtypen können keinen Inline- oder Pseudo-Elementen zugeordnet werden. Die Rollenzuordnung lässt sich mit der Dokumentoption *rolemap* durchführen, zum Beispiel:

```
p.begin_document("tagged.pdf",
        "tagged=true lang=en rolemap={ {Heading H1} {Subhead H2} {Paragraph P} }");
```

Rollenzuordnungen in Acrobat. In Acrobat X/XI können Sie Rollenzuordnungen folgendermaßen anzeigen und bearbeiten:

- Wählen Sie Menü *Anzeige, Ein-/Ausblenden, Navigationsfenster, Tags*, klicken Sie auf die Menüschaltfläche oben im Fenster *Tags* und wählen Sie *Rollenzuordnung bearbeiten.*

10.3.3 Artefakte

Relevanter Inhalt und Artefakte. Der Inhalt einer Seite setzt sich folgendermaßen zusammen:

- Der bedeutungstragende Inhalt eines Textes wird vom Verfasser des Dokuments erstellt. Der logische Strukturbaum beschreibt die Objekte, die den relevanten Inhalt des Dokuments ausmachen. Anmerkungen können ebenfalls enthalten sein.
- Grafik- oder Textobjekte, die für den relevanten Inhalt des Dokuments nicht von Bedeutung sind, wie zum Beispiel Seitenzahlen oder Formatierung, werden Artefakte genannt. Artefakte werden in den Strukturbaum nicht mit aufgenommen und von Screenreadern ignoriert.

Die Kennzeichnung von Artefakten ist sehr ratsam, da sie beim Umfließen von Text und zur Barrierefreiheit (Accessibility) hilfreich ist und auch bei PDF/UA verlangt wird. Typische Artefakte sind laufende Kopf-/Fußzeilen, Seitenzahlen, Hintergrundbilder und andere sich auf jeder Seite wiederholende Objekte.

Artefakte in Acrobat. In Acrobat X/XI können Sie Artefakte auf eine der folgenden Arten überprüfen:

- Wählen Sie *Werkzeuge, Ein-/Ausgabehilfe, TouchUp-Leserichtung,* um Inhaltselemente anzuzeigen oder zu bearbeiten. Artefakte werden im TouchUp-Leserichtungwerkzeug von Acrobat *Hintergrund* genannt. Anders als Strukturelemente werden sie nicht mit einem Rahmen oder Tag-Namen dargestellt, wenn das Werkzeug aktiviert ist.
- Wählen Sie *Anzeige, Ein-/Ausblenden, Navigationsfenster, Inhalt,* um Artefakte anzuzeigen. Im Fenster *Inhalt* wird der gesamte Inhalt jeder Seite mit den jeweiligen Strukturelementen und Artefakten angezeigt. Da Artefakte nicht vorgelesen werden, fehlt der entsprechende nummerierte Block auf der Seite. Durch Anklicken eines Artefakts in der Liste wird jedoch das entsprechende Inhaltselement auf der Seite hervorgehoben.

- Nach Artefakten suchen: wählen Sie *Anzeige, Ein-/Ausblenden, Navigationsfenster, Tags,* klicken Sie auf die Menüschaltfläche oben im Fenster *Tags*, wählen Sie *Suchen...* und dann *Außertextliche Elemente* aus der Liste. Da Artefakte nicht Teil der Dokumentstruktur sind, gibt es keinen entsprechenden Eintrag im Fenster *Tags* oder *Reihenfolge*.
- Wählen Sie *Anzeige, Sprachausgabe, Sprachausgabe aktivieren,* um die Strukturelemente auf der Seite vorlesen zu lassen. Artefakte werden nicht mit vorgelesen.

Auszeichnen von Inhalt als Artefakt. Artefakte lassen sich in PDFlib mit dem Tag *Artifact* von *PDF_begin_item( )* angeben (obwohl sie keine Tags im Sinne von Strukturelementen sind):

```
id = p.begin_item("Artifact", "");
```

Artefakte können auch beim vereinfachten Anbringen von Tags angegeben werden, also mit der Option *tag* verschiedener Funktionen:

```
p.fit_textline(text, x, y, "tag={tagname=Artifact}");
```

Klassifizierung von Artefakten. Artefakte sollten mit dem Pseudo-Tag *Artifact* als solche gekennzeichnet und mit einem der folgenden Schlüsselwörter der Option *artifacttype* klassifiziert werden:

- *Pagination:* Paginierungseigenschaften wie laufende Kopfzeile oder Seitenzahlen. *Pagination*-Artefakte können mit der Option *artifactsubtype* und einem der Schlüsselwörter *Header, Footer* oder *Watermark* genauer gekennzeichnet werden.
- *Layout:* typografische oder Designelemente wie Linien oder Tabellenschattierungen
- *Page:* Produktionshilfen wie Beschnittmarken oder Farbauszüge;
- *Background*: über die gesamte Breite oder Höhe einer Seite oder eines Strukturelements verlaufende Bilder oder farbige Bereiche.

Artefakte der Klasse *Pagination* und *Background* unterstützen die Option *Attached*, die angibt, an welcher Kante oder Kanten der Seite das Artefakt steht *(Top/Bottom/Left/Right)*. Dieses Attribut kann von Acrobat zur Verbesserung des Umfließens von Text verwendet werden.

Im folgenden Beispiel wird ein Artefakt vom Typ *pagination* mit dem Subtyp *Header* erzeugt:

```
id = p.begin_item("Artifact",
        "artifacttype=Pagination artifactsubtype=Header Attached={Top Left}");
```

Automatisches Tagging von Artefakten. Da jeglicher Seiteninhalt entweder als Strukturelement oder als Artefakt ausgezeichnet werden sollte, werden bei PDFlib bestimmte grafische Elemente automatisch mit Tags versehen. Mit *artifacttype=Layout* werden die folgenden Elemente zur Textdekoration im Modus Tagged PDF automatisch als *Artifact* ausgezeichnet:

- Blockdekoration: alle mit *PDF_fill_*block( )* erzeugten Elemente zur Textdekoration, das heißt, das Zeichnen und Füllen von Umrisslinien gemäß den Properties *backgroundcolor* und *bordercolor*.
- Matchbox-Dekoration: gemäß den Matchbox-Optionen *fillcolor*, *shading* und *strokecolor* erzeugte Matchbox-Rechtecke

- Tabellendekoration: das Zeichnen und Füllen von Umrisslinien gemäß den Optionen *fill, stroke, showborder, showgrid,* wenn automatisches Anbringen von Tabellen-Tags aktiviert ist (siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300)
- Textline-Artefakte: *leader, shadow, showbox*
- Textflow-Artefakte: *leader, shadow, showbox, showtabs*
- Textdekoration: *underline, overline, strikeout*

Wie alle Funktionen für Tagged PDF funktioniert das automatische Tagging von Artefakten nur im Geltungsbereich *page*.

10.3.4 Textverarbeitung

Angabe der Sprache. Bei Tagged PDF sollte die natürliche Sprache des Textes explizit angegeben werden; dadurch können einige Screenreader das Dokument in der passenden Sprache vorlesen. Die natürliche Sprache kann auf mehreren Ebenen angegeben werden:

- Mit der Option *lang* von *PDF_begin_document( )* sollte die Primärsprache angeben werden, also die natürliche Sprache des gesamten Dokuments. Bei Dokumenten in mehreren Zielsprachen, wie beispielsweise einem mehrsprachigen Vertrag, sollte die Option nicht gesetzt werden.
- Die Dokumentsprache kann bei einzelnen Elementen auf jeder Stufe der Strukturhierarchie mit der Option *lang* von *PDF_begin_item( )* oder der Option *tag* verschiedener Funktionen überschrieben werden.
- Hypertext-Strings können Unicode-Escapesequenzen zur Sprachkennzeichnung enthalten (siehe unten).

Für Inhalte, die als Text kodiert, aber nicht Teil einer natürlichen Sprache sind, wie zum Beispiel Programmcode, Musiknoten, Schriftmuster oder mathematische Formeln sollte ein leerer Sprachcode verwendet werden, zum Beispiel *lang={ }*.

Unicode-Sprachkennungen setzen sich aus den folgenden Sequenzen zusammen:

- dem Unicode-Wert U+001B (zwei Bytes)
- einem ISO-639-Sprachcode (zwei ASCII-Bytes), z.B. *en, ja, de*
- optional einem ISO-3166-Ländercode (zwei ASCII-Bytes), z.B. *US, JP*
- dem Unicode-Wert U+001B (zwei Bytes).

Beispiele in hexadezimaler Schreibweise:

```
001B656E5553001B          (=enUS)
001B7A68001B              (=zh)
001B6465001B              (=de)
```

In Unicode-fähigen Sprachen wie Java müssen die beiden ASCII-Zeichen zu einem einzigen Unicode-Wert zusammengesetzt werden:

```
\u001B\u6465\u001B        (=de)
```

In der C-Sprachbindung müssen die beiden ASCII-Zeichen zu einem einzigen Unicode-Wert zusammengesetzt werden. Mit *charref=true* kann die Sequenz folgendermaßen ausgedrückt werden:

```
&#x001B;&#x6465;&#x001B;  (=de)
```

Erzeugung von Tagged PDF mit Textflows. Textflows bieten leistungsfähige Features zur Textformatierung (siehe Abschnitt 8.2, »Mehrzeilige Textflows«, Seite 224). Da sich einzelne Textfragmente nicht mehr unter Clientkontrolle befinden, sondern automatisch von PDFlib formatiert werden, muss bei der Generierung von Tagged PDF mit Textflows besondere Sorgfalt angewandt werden:

- Der Inhalt einer Textflow-Fitbox kann vollständig in einem Strukturelement enthalten sein, allerdings können Textflows keine einzelnen Strukturelemente enthalten.
- Alle Teile eines Textflows (alle Aufrufe von *PDF_fit_textflow( )* mit einem bestimmten Textflow-Handle) sollten sich innerhalb desselben Strukturelements befinden.
- Da sich ein Textflow über mehrere Seiten erstrecken kann, die unterschiedliche Strukturelemente enthalten, sollte das übergeordnete Element mit Bedacht gewählt und nicht einfach die Option parent= -1 verwendet werden, die auf das falsche übergeordnete Element verweisen könnte.
- Wenn Sie das Matchbox-Feature nutzen wollen, um Links oder andere Anmerkungen in einem Textflow zu erstellen, ist es kaum möglich, die Kontrolle über die Position der Anmerkung im Strukturbaum zu behalten.

Wortgrenzen durch Leerzeichen markieren. Wörter sollten durch Leerzeichen (U+0020) voneinander getrennt werden. Mit der Option *autospace* lassen sich Leerzeichen automatisch nach jedem Aufruf einer der Textausgabe-Funktionen erzeugen.

Trennung. Die Trennung eines Wortes am Ende einer Zeile sollte durch ein weiches Trennzeichen (*soft hyphen* U+00AD) und nicht durch eine hartes Trenn- oder Minuszeichen (U+002D) dargestellt werden. Bei weichen Trennzeichen kann Acrobat die Wörter bei der Textsuche wieder korrekt zusammensetzen oder die Trennzeichen entfernen, falls der Zeilenumbruch bei einer anderen Formatierung keine Trennung mehr erfordert. Wenn der Text ein weiches Trennzeichen U+00AD enthält, verwendet PDFlib die Glyphe für U+00AD, sofern im Font vorhanden, ansonsten U+002D. Wenn im Font zwei verschiedene Glyphen für U+00AD und U+002D vorkommen, reicht es, im Text U+00AD zu verwenden, um die Anforderungen von Tagged PDF an die Trennung zu erfüllen.

Wenn im Font keine separate Glyphe für U+00AD vorhanden ist und stattdessen ein anderes Trennzeichen verwendet wird, muss das Trennzeichen mit dem Attribut *ActualText,* das U+00AD enthält, als *Span* oder *ASpan* ausgezeichnet werden. (Dies ist bei PDF 1.4 jedoch nicht möglich). Bei mehrzeiligem Textflow wird *ActualText* dem jeweiligen *hyphenchar* automatisch zugewiesen und *autospace* unterdrückt.

ActualText lässt sich bei Textzeilen folgendermaßen hinzufügen:

- Bei *PDF_fit_textline( )* sorgt die Option *tagtrailinghyphen* für das passende *ActualText* und unterdrückt *autospace*. Da die Option standardmäßig auf U+00AD gesetzt ist, wird der Text automatisch korrekt mit Tags versehen, wenn U+00AD als Trennzeichen verwendet wird.
- Wenn im Font keine separate Glyphe für U+00AD vorhanden ist, können Sie sie mit der folgenden Option zum Laden von Fonts durch das entsprechende weiche Trennzeichen aus einem geeigneten Fallback-Font ersetzen:

```
fallbackfonts={{fontname=AuxiliaryFont encoding=unicode embedding forcechars=x00AD}}
```

- Sie können ein passendes *ActualText* auch manuell zuweisen; dies ist lediglich erforderlich, wenn im Font keine separaten Glyphen für U+00AD und U+002D vor-

handen sind (für diese Vorgehensweise wird PDF ab Version 1.5 benötigt; verwenden Sie für PDF 1.4 *inline=false):*

```
p.set_option("charref=true");
p.fit_textline("-", x, y, "tag={tagname=ASpan ActualText=&#x00AD;}");
```

Eigenschaften von Type-3-Fonts. Für alle in einem Tagged PDF verwendeten Type-3-Fonts sollten die Optionen *familyname*, *stretch* und *weight* von *PDF_begin_font( )* mit geeigneten Werten übergeben werden.

10.3.5 Alternativtext, Ersatztext und Abkürzungsexpansion

Tagged PDF bietet Features zur Verbesserung der Zugänglichkeit von Bildern und Texten, die ohne zusätzliche Informationen nur schwer lesbar wären.

Alternativtext (Alt). Sie können Bildern, Formeln oder anderen nicht textuellen Elementen mit der Option *Alt* einen Alternativtext mitgeben. Der übergebene Wert liefert eine Beschreibung dieses und aller ihm untergeordneten Strukturelemente. Der Alternativtext sollte aus einem ganzen Wort oder Satz bestehen und mit einem Punkt oder Leerzeichen abgeschlossen werden, damit Screenreader ihn nicht mit dem nachfolgenden Text vermischen. Sie sollten im Alternativtext auf einleitende Wendungen wie *»Dieses Bild zeigt…«* verzichten.

Das dargestellte Firmenlogo von PDFlib könnte mit der Option *Alt* zum Beispiel folgendermaßen beschrieben werden:

```
p.fit_image(image, x, y, "tag={tagname=Figure Alt={PDFlib company logo}}")
```

Ersatztext (ActualText). Seitenelemente, die zwar als Text im PDF dargestellt werden, deren Bedeutung aber nicht direkt abgeleitet werden kann, kann mit der Option *AtualText* ein Ersatztext zugewiesen werden. Dazu gehören zum Beispiel Illustrationen mit Zierbuchstaben oder mehrzeiligen Anfangsbuchstaben. Der übergebene Wert dient als Ersatztext für dieses und alle ihm untergeordneten Strukturelemente. Der Ersatztext kann ein oder mehrere Zeichen enthalten (im Gegensatz zum Attribut *Alt,* das aus einem ganzen Wort oder Satz besteht). Der Ersatztext sollte dem entsprechen, was jemand beim Lesen des Inhalts sehen würde.

Der Symbolglyphe *flower,* für die es keinen Unicode-Wert gibt, könnten Sie zum Beispiel einen entsprechenden *ActualText* übergeben, der beschreibt, dass sie eigentlich als Aufzählungszeichen U+2022 verwendet wird:

```
p.fit_textline("&.flower;", x, y, "tag={tagname=ASpan ActualText={&#x2022;} } ...");
```

Verschachtelungsregeln für Alternativ- und Ersatztext. Für die Verwendung der Attribute *Alt* und *ActualText* gelten folgende Regeln:

- Wenn ein übergeordnetes Strukturelement bereits ein Attribut vom Typ *ActualText* oder *Alt* enthält, dürfen diese in den untergeordneten Elementen nicht mehr vorkommen, da sich das übergeordnete Attribut bereits auf alle untergeordneten Elemente bezieht.
- Ein Element mit einem Attribut *Alt* oder *ActualText* muss direkten Inhalt oder ein oder mehrere Elemente vom Typ *Link* enthalten; wenn das Attribut auf ein Element vom Typ *Link* angewendet wird, muss es direkten Inhalt oder einen oder mehrere *OBJR*-Verweise enthalten, die mit *PDF_create_annotation( )* erstellt wurden (siehe

»Links und andere Arten von Anmerkungen«, Seite 303). Sonst wäre es nicht möglich, die Seitenzahl zu ermitteln, auf der das Attribut gelesen werden muss. Diese Regel bezieht sich auf das Element selbst; untergeordnete Elemente mit direktem Inhalt sind in diesem Fall nicht ausreichend.

Mit dem Element *ASpan* lässt sich *Alt* oder *ActualText* auf einzelne Teile eines Strukturelements anwenden.

Alternativ- und Ersatztext in Acrobat. In Acrobat X/XI können Sie die Attribute *Alt* und *ActualText* eines Strukturelements folgendermaßen anzeigen:

- Wählen Sie das Menü *Anzeige, Ein-/Ausblenden, Navigationsfenster, Tags,* rechts-klicken Sie auf ein Strukturelement im Tagbaum und wählen Sie *Eigenschaften...*. Im Dialog *TouchUp-Eigenschaften* werden unter der Registermarke *Tag* die Werte für die Attribute *Actual Text* und *Alternate Text* angezeigt.

Abkürzungsexpansion (E). Für Abkürzungen und Akronyme lässt sich mit der Unteroption *E* der Option *tag* ein Erweiterungstext angeben. Dieser sollte aus einem ganzen Wort oder einem Satz bestehen. Selbst bei einer Abkürzung ohne Erweiterungstext können Sie das Attribut *E* übergeben, um die Sprachausgabe zu verbessern. Für die Abkürzung *IBM* kann zum Beispiel der Erweiterungstext *I B M* mit Leerzeichen zwischen den einzelnen Buchstaben zugewiesen werden.

Im folgenden Codefragment wird für die Abkürzung *Mr.* der Erweiterungstext *Mister* hinterlegt:

```
p.fit_textline("Mr.", x, y, "tag={tagname=ASpan E={Mister} } ...");
```

10.3.6 Druckreihenfolge und logische Lesereihenfolge

Es gibt zwei grundsätzlich verschiedene Konzepte für die Anordnung bzw. Reihenfolge des Inhalts in PDF. Abbildung 10.3 zeigt eine Beispielseite mit zwei Textspalten, unterbrochen von einer Tabelle und einer Zusammenfassung auf grauem Hintergrund, sowie von Kopf-und Fußzeilen.

Die Reihenfolge der API-Funktionsaufrufe von PDFlib bestimmt die Druckreihenfolge der PDF-Text- und Grafikoperatoren in der Seitenbeschreibung (in Acrobat: Leserichtung in Druckdatenstrom). Der Aufbau der Seiten durch die jeweilige Anwendung ist dabei technisch bedingt und muss nicht unbedingt der semantischen Anordnung der Seiteninhalte entsprechen. Die Navigationsfenster *Reihenfolge* und *Inhalt* zeigen die Seiteninhalte immer in Druckreihenfolge an.

Die logische Lesereihenfolge ist die Reihenfolge, in der ein Mensch den Text liest. Sie bestimmt die von einem Screenreader und der Sprachausgabe von Acrobat verwendete Reihenfolge. Die logische Lesereihenfolge wird durch die logische Baumstruktur bestimmt. Tagged PDF erfordert, dass alle Inhalte in semantisch richtiger Reihenfolge mit Tags versehen werden. Die Strukturhierarchie muss die Seiteninhalte in der Reihenfolge enthalten, in der sie vom Menschen gelesen werden. Das korrekte Anbringen von Tags sorgt dafür, dass Screenreader die Inhalte in logischer Reihenfolge ausgeben können. Da Artefakte nicht Teil der Baumstruktur sind, sind sie auch nicht Teil der logischen Lesereihenfolge.

Druckreihenfolge und logische Lesereihenfolge in Acrobat. Die logische Lesereihenfolge können Sie in Acrobat X/XI auf folgende Arten prüfen:

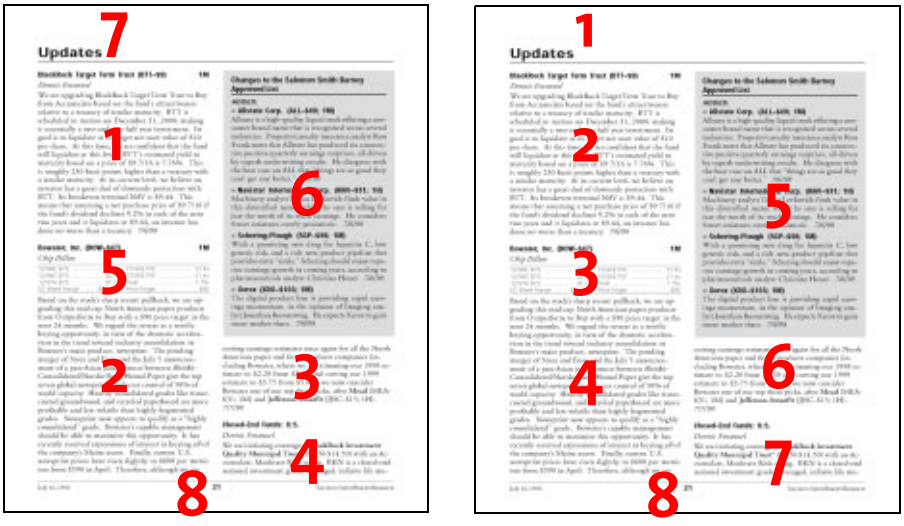


Abb. 10.3
Logische Lesereihenfolge (links) und Druckreihenfolge (rechts)

- Wählen Sie *Anzeige, Ein-/Ausblenden, Navigationsfenster, Tags* und prüfen Sie im Tagfenster die Reihenfolge der Elemente von oben nach unten. Diese sollte die gewünschte Lesereihenfolge korrekt widerspiegeln.
- Wählen Sie *Anzeige, Sprachausgabe, Sprachausgabe aktivieren...* ; Acrobat liest die Seiteninhalte in der im Dokument festgelegten Reihenfolge vor.

In den Navigationsfenstern *Reihenfolge* und *Inhalt* werden die Seiteninhalte immer in Druckreihenfolge dargestellt.

Erzeugen von Seiteninhalt in logischer Lesereihenfolge. Empfehlenswert ist die nahe liegende Methode, erst nacheinander alle Bestandteile eines Strukturelements zu erzeugen und dann mit dem nächsten Element fortzufahren. Technisch ausgedrückt sollte der Strukturbaum während eines einzigen »Depth-First«-Durchlaufs generiert werden. Dabei werden PDFlib-Funktionen zur Erzeugung der Seiteninhalte in der Reihenfolge aufgerufen, in der die Inhalte gelesen werden.

PDFlib unterstützt verschiedene Methoden zur Erzeugung von Inhalten in beliebiger Reihenfolge, wobei die Strukturhierarchie immer in logischer Reihenfolge aufgebaut wird. Für weitere Informationen siehe Abschnitt 10.4.4, »Erzeugung von Seiteninhalt in abweichender Reihenfolge«, Seite 308.

10.3.7 Verwendung von Tagged PDF in Acrobat

In diesem Abschnitt schildern wir unsere Beobachtungen beim Testen von Tagged PDF in Adobe Acrobat. Tabelle 10.4 listet fehlerhaftes und inkonsistentes Verhalten in verschiedenen Versionen von Acrobat auf, das nach folgenden Acrobat-Features geordnet ist:

- Umfließen: Acrobat ermöglicht in Tagged PDF das Umfließen von Text, das heißt, die Anpassung des Seiteninhalts an die aktuelle Fenstergröße.

- Prüfen der Barrierefreiheit: Mit dem Accessibility Checker von Acrobat kann die Qualität von Tagged PDF für die Verwendung mit assistiver Technologie wie Screenreadern geprüft werden.
- Exportieren in andere Dateiformate: Mit Tagged PDF werden PDF-Dokumente in Acrobat erheblich besser in andere Formate wie XML oder RTF exportiert.
- Sprachausgabe: Tagged PDF ermöglicht eine verbesserte Sprachausgabe in Acrobat.
- Validierung von Tagged PDF: Mit dem Preflight-Feature von Acrobat können PDF-Dokumente validiert werden.
- Suchfunktion im Tagfenster: Mit der Suchfunktion im Tagfenster von Acrobat kann man nach Artefakten und nach nicht markierten Inhalten suchen.

Tabelle 10.4 Probleme in Acrobat beim Umgang mit Tagged PDF

Beschreibung, Empfehlungen, Lösungen	***Acrobat-Version***		
	9	***X***	***XI***
Acrobat-Feature Umfließen			
Importierte PDF-Seiten werden im Umfließen-Modus nicht angezeigt.	*X*	X	X
Symbolfonts (nicht Unicode-basiert) können den Umfließen-Modus deaktivieren. Sie sollten den Symboltext in das Element `Figure` *einfügen.*	*X*	–	–
BLSEs können sowohl untergeordnete Strukturelemente als auch direkte Inhaltselemente enthalten. Elemente mit verschiedenen Typen von Unterelementen (also mit ganzen Seitensegmenten und Nicht-Inline-Elementen) sollten vermieden werden, da sie beim Umfließen unter Umständen fehlerhaft behandelt werden (in Acrobat XI wird das Umfließen in diesem Fall gesperrt). *Damit die Umfließen-Funktion (sowie Accessibility Checker und Sprachausgabe) in diesem Fall funktionieren, sollten die direkten Elemente vor das erste Unterelement platziert werden. Ein Strukturelement mit verschiedenen Unterelementtypen sollte nur ein Unterelement mit direktem Inhalt haben, und dieses sollte das erste untergeordnete Element sein.*	*X*	*X*	*X*
Bei Seiten, die mit der Option `topdown` *erzeugt wurden, kann es zu Problemen beim Umfließen kommen.*	*X*	*X*	*X*
Enthält ein aktiviertes Element nur Inhalte, aber keine untergeordneten Strukturelemente, scheitert das Umfließen möglicherweise. Dies ist insbesondere dann der Fall, wenn das Element auf einer anderen Seite aktiviert wurde. Das Problem lässt sich umgehen, indem das aktivierte Element nicht inline mit einem `Span`*-Tag umgeben wird.*	*X*	*X*	*X*
Das Umfließen von Text auf Seiten mit Formularfeldern (einschließlich digitaler Signaturen) ist nicht möglich, in diesem Fall wird eine Warnung ausgegeben.	*X*	*X*	*X*
Acrobat-Feature Barrierefreiheitsprüfung (Accessibility Checker)			
Das Attribut `Alt` *wird bei Tags vom Typ* `Figure` *ignoriert.*	*X*	*X*	–
Bei der Prüfung auf Barrierefreiheit wird das Attribut `Lang` *auf Dokumentebene erwartet, obwohl das Attribut* `Lang` *jeweils für die einzelnen Strukturelemente gesetzt ist.*	*n/a*	*X*	*X*
Bei der Prüfung auf Barrierefreiheit wird Alternativtext für Bilder ignoriert, die Teil einer importierten Seite sind.	*X*	*X*	*X*
Bei der Prüfung auf Barrierefreiheit wird für Formularfelder vom Typ Kontrollkästchen, die mit einem Häkchen versehen sind, »Zeichenkodierung – fehlgeschlagen« angezeigt, obwohl der Inhalt der Formularfelder bei der Sprachausgabe korrekt wiedergegeben wird.	*X*	*X*	*X*

Tabelle 10.4 Probleme in Acrobat beim Umgang mit Tagged PDF

Beschreibung, Empfehlungen, Lösungen	**Acrobat-Version**		
	9	**X**	**XI**
Acrobat-Feature Exportieren in andere Dateiformate			
Enthält eine importierte PDF-Seite das `Form`*-Tag, wird in Acrobat der mit der Option* `ActualText` *übergebene Text in andere Formate exportiert, der mit dem* `Alt`*-Tag übergebene Text dagegen ignoriert. Die Sprachausgabe berücksichtigt jedoch beide Optionen.*	*X*	*–*	*–*
Acrobat extrahiert den tatsächlichen Inhalt: die Optionen `Alt` *und* `ActualText` *sowie die Tags* `Private` *und* `NonStruct` *werden ignoriert.*	*–*	*X*	*–*
Der Inhalt des Tags `NonStruct` *wird nicht nach HTML 4.01 CSS 1.0 exportiert (aber beim Export nach HTML 3.2 verwendet).*	*X*	*–*	*–*
Für ILSEs (wie `Code`*,* `Quote` *oder* `Reference`*) sollte ein Alternativtext übergeben werden. Wird die Option* `Alt` *verwendet, liest die Sprachausgabe zwar den übergebenen Text, es wird aber der tatsächliche Inhalt in andere Formate exportiert. Wird die Option* `ActualText` *verwendet, wird der übergebene Text sowohl bei der Sprachausgabe als auch beim Export verwendet.*	*X*	*X*	*X*
Acrobat-Feature Sprachausgabe			
Wenn mit Tags versehene Seiten mit PDI platziert werden und nur Artefakte enthalten, wird bei der Sprachausgabe trotzdem der Inhalt der platzierten Seiten vorgelesen.	*X*	*X*	*X*
Die Attribute `Alt` *und* `ActualText` *werden bei Tags vom Typ* `Figure` *ignoriert.*	*X*	*X*	*–*
Acrobat-Feature PDF-Validierung mit Preflight			
Das PDF-Analyseprofil »PDF-Syntaxfehler ermitteln« von Acrobat gibt fälschlicherweise »Inkonsistente Zuordnung im ParentTree« aus, wenn eine mit Tags versehene Seite mit PDI platziert wurde.	*X*	*X*	*–*
Weitere Acrobat-Funktionen			
Die Suchfunktion im Acrobat-Tagfenster gibt Artefakte fälschlicherweise als nicht markierten Inhalt aus, z.B. Tabellendekoration. Im Inhaltsfenster von Acrobat werden diese Elemente dagegen korrekt als »Container <Artifact>« aufgeführt.	*X*	*X*	*X*
Attribute von Strukturelementen können nach dem Speichern eines Dokuments mit Acrobat verlorengehen. Auch Attribute für die Barrierefreiheit können auf diese Weise verlorengehen, z.B. das Attribut `Summary` *des Elements* `Table`*.*	*X*	*X*	*X*

10.4 Fortgeschrittene Themen bei Tagged PDF

10.4.1 Automatisches Erstellen von Tabellen-Tags

Mit *PDF_fit_table( )* lassen sich für die erzeugten Tabellen automatisch geeignete Tags erstellen, die auf Informationen basieren, die an die Aufrufe von *PDF_add_table_cell( )* für den Tabelleninhalt übergeben wurden. Die Option *tag* von *PDF_fit_table( )* mit *tagname=Table* aktiviert das automatische Erstellen von Tabellen-Tags, wie in Tabelle 10.5 beschrieben.

Tabelle 10.5 Die Option `tag` *von PDF_fit_table( ) und das automatische Erstellen von Tabellen.Tags*

Tag-Option von PDF_fit_table( )	***Ergebnis***
tagname=Table	*Aktiviert das automatische Erstellen von Tabellen-Tags. Anstelle von* `tagname=Table` *kann auch ein benutzerdefiniertes Tag mit einer Rollenzuordnung zu* `Table` *verwendet werden. Wir empfehlen, die Unteroption* `Summary` *anzugeben.*
tagname=Artifact	*Der gesamte Tabelleninhalt einschließlich der Titelzeile wird als Artefakt ausgezeichnet; das Attribut* `BBox` *wird automatisch hinzugefügt.*
alle anderen Tag-Namen	*Es wird keine* `Table`*-Struktur erstellt, jedoch werden die Zelleninhalte dem in der Option* `tag` *übergebenen Element als Unterelemente hinzugefügt. Pseudo- und Inline-Elemente sind bei PDF_fit_table( ) nicht erlaubt.*
Option tag nicht angegeben	*Tabellen-Tags werden nicht automatisch erstellt. Wenn die Option* `tag` *bei einzelnen Aufrufen von PDF_add_table_cell( ) mit übergeben wird (als Unteroption für eine der Optionen* `fit*`*), wird das entsprechende Strukturelement für den Zelleninhalt ohne Tabellenstruktur erstellt. Dies ist nützlich, wenn eine Tabelle lediglich zu Layoutzwecken verwendet wird.*

Hinweis *Das automatische Erstellen von Tabellen-Tags kann nur wirksam eingesetzt werden, wenn der Tabellenformatierer von PDFlib verwendet wird. Obwohl das korrekte Anbringen von Tabellen-Tags auch bei der manuellen Erstellung von Tabellen (das heißt ohne den PDFlib-Tabellenformatierer) funktioniert, verlangt dieses Vorgehen jedoch genaue Kenntnisse über die Tabellenstruktur in der Client-Anwendung. Außer der entsprechenden Information über die Zeilen-/Spaltenstruktur wird auch Information über Überschriftenzellen und verbundene Zeilen bzw. Spalten benötigt. Auch leere Zellen müssen mit Tags versehen werden.*

Anzeigen von Tabellen-Tags in Acrobat. Die Struktur von Tabellenelementen können Sie bei Acrobat X/XI folgendermaßen anzeigen lassen:

- Wählen Sie *Werkzeuge, Ein-/Ausgabehilfe, TouchUp-Leserichtung*. Tabellenelemente werden markiert und mit einer kleinen Nummer am oberen linken Rand der Tabelle versehen. Eine Tabellenzusammenfassung wird, falls vorhanden, neben dieser Nummer angezeigt.
- Wählen Sie die Nummer (oder bei Acrobat XI den Namen des Strukturtyps) am oberen linken Rand der Tabelle, und klicken Sie im Dialog *TouchUp-Leserichtung* auf *Tabellen-Editor*. Die Tabellenstruktur wird durch horizontale und vertikale Linien dargestellt. Bei Acrobat X werden je nach Zellentyp für *TH/TD* Icons angezeigt, bei Acrobat XI dagegen nicht (siehe Abbildung 10.4).
- Rechts-klicken Sie auf eine Zelle und wählen Sie *Eigenschaften von Tabellenzellen*, um den Zellentyp (Überschriftenzelle *TH* bzw. Datenzelle *TD)*, die Attribute *scope* (in Acro-

Abb. 10.4
Der Tabellen-Editor von Acrobat zeigt Überschriften- (TH) und Datenzellen (TD).

bat: *Umfang)*, *rowspan* (in Acrobat: *Zeilenbereich)* und *colpan* (in Acrobat: *Spaltenbereich)* und die Werte *header/ID* abzufragen.

Beachten Sie folgende Einschränkungen beim Umgang mit dem Tabellen-Editor von Acrobat X/XI:

- Die Linien für die Tabellenzellen werden manchmal an der falschen Position angezeigt.
- Wenn in der Tabelle ein benutzerdefiniertes Strukturelement mit einer Rollenzuordnung zu *Table* vorkommt und nicht das Standardelement *Table* selbst, wird der Tabellen-Editor in Acrobat nicht aktiviert.
- Elemente vom Typ *Caption* in einer Tabelle werden im Tabellen-Editor nicht angezeigt.
- Eine Tabellenzelle mit vertikalem Text (etwa eine Textzeile mit *orientate=east* oder *west)* sowie die Zelle direkt rechts daneben werden im Tabellen-Editor nicht angezeigt, obwohl sie im logischen Strukturbaum vorhanden sind und ihr Inhalt auf der Seite sichtbar ist.

Automatisch erstellte Tabellen-Tags und -attribute. Tabellen-Tags werden im Geltungsbereich *page* auf folgende Weise automatisch erstellt und unterliegen folgenden Regeln:

- Für jede Tabelleninstanz wird ein separates Element *Table* erstellt. Wenn eine Tabelle zum Beispiel in zwei oder mehr Instanzen aufgeteilt ist, werden mehrere Elemente vom Typ *Table* erstellt. Wenn die Unteroption *Summary* wie empfohlen in der Option *tag* von *PDF_fit_table( )* übergeben wurde, wird dem Element *Table* das Attribut *Summary* hinzugefügt.
- Wenn die Option *caption* bei *PDF_fit_table( )* angegeben wurde, wird das Gruppierungselement *Caption* erstellt. *Caption* darf keinen direkten Inhalt haben. Sie müssen deshalb die Unteroption *tag* der Option *caption* übergeben, um ein Strukturelement als Unterelement von *Caption* festzulegen. In diesem Element kann der eigentliche Inhalt des Tabellentitels enthalten sein.
- Für jede Tabellenzeile wird ein Element *TR* angelegt. Zeilen, die in der Option *header* von *PDF_fit_table( )* angegeben wurden, werden in *THead* eingebettet. Zeilen, die in

der Option *footer* angegeben wurden, werden in *TFoot* eingebettet. Alle anderen Zeilen werden von *TBody* eingefasst, sofern Kopf-/Fußzeilen vorhanden sind.

- Jede Tabellenzelle wird in ein Element *TH* (Überschriftenzelle) oder *TD* (Datenzelle) gemäß der Unteroption *tagname* der Option *tag* von *PDF_add_table_cell( )* eingebettet. Falls diese Option nicht übergeben wird, wird der Zellentyp folgendermaßen ermittelt:
  - Das Attribut *Scope* für eine Zelle bewirkt *TH* (selbst bei *tagname=TD)*.
  - Wenn eine andere Zelle die Option *Headers* mit der ID der aktuellen Zelle trägt, wird die Zielzelle zu *TH* gemacht (selbst bei *tagname=TD*).
  - Wenn die Zelle Teil einer Tabellenkopfzeile gemäß der Option *header* von *PDF_fit_table( )* ist, wird sie in das Element *TH* eingebettet und *Scope=Column* wird hinzugefügt.
- Ein leeres Dummy-Element *TD* wird für jede Zelle erstellt, für die kein Inhalt mit *PDF_add_table_cell( )* angelegt wurde.
- Die Elemente *TH* und *TD* erhalten die entsprechenden Attribute *RowSpan* und *ColSpan* gemäß der Option *rowspan* und *colspan* von *PDF_add_table_cell( )*. Die Unteroptionen *RowSpan* und *ColSpan* der Option *tag* können nicht verwendet werden.
- Den Elementen *Table, TH* und *TD* werden die passenden Attribute *Width* und *Height* zugewiesen; *Table*-Elementen wird das Attribut *BBox* zugewiesen.
- Der Option *tag* von *PDF_add_table_cell( )* können weitere Attribute für Tabellenzellen als Unteroptionen mitgegeben werden. Folgende Optionen sind nicht erlaubt: *RowSpan, ColSpan*, *Height, index*, *parent, Width.*
- Tabellenzeilen und -spalten werden in Zick-Zack-Reihenfolge, beginnend mit der oberen linken Zelle (das heißt Spalte 1, Zeile 1) bis zur unteren rechten Zelle ausgegeben, unabhängig von der Reihenfolge der Aufrufe von *PDF_add_table_cell( )*.
- Dekorative Tabellenelemente werden automatisch als *Artifact* mit *artifacttype= Layout* ausgezeichnet, das heißt Rahmen und Schattierung von Tabellenzellen, -reihen, -spalten oder der gesamten Tabelle, Rahmen und Füllung von Matchboxen, *showborder*-Regeln sowie Hilfen zu Visualisierung, die über *debugshow*, *showcells* und *showgrid* gesteuert werden.

Cookbook *Codebeispiele zur automatischen Erstellung von Tabellen-Tags finden Sie in den Topics* `tagged_table` *und* `tagged_invoice` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

Hinzufügen von Tags und Attributen zu Tabellenzellen. Vereinfachtes Anbringen von Tags kann bei Tabellenüberschriften, -zellen oder Inhalten von Tabellenzellen angewendet werden. Die Angabe der Option *tag* von *PDF_add_table_cell( )* ist in folgenden Fällen nützlich:

- Tabellenzellen können mit *tagname=TH* zu Überschriftenzellen (statt zu *TD*-Datenzellen) gemacht werden, wenn sie nicht in Überschriftenzeilen enthalten sind oder kein *Scope*-Attribut haben.
- Um eine richtige Tag-Struktur aufzubauen, wie sie für Links benötigt wird; für weitere Informationen siehe »Tags für Links in Tabellenzellen«, Seite 305.

Folgende Einschränkungen gelten für die Unteroptionen der Option *tag* von *PDF_add_table_cell( ):*

- Folgende Optionen können nicht verwendet werden: *ColSpan, Height, index, parent, RowSpan, Width.*

- Die Option *tagname* kann nur den Wert *TH* oder *TD* haben, um den Zellentyp anzugeben. Untergeordnete Tags können jedoch durch Verschachtelung der Option *tag* erstellt werden.
- Da die Option *id* innerhalb eines Dokuments eindeutig sein muss, ist sie für Zellen, die sich in mehreren Tabelleninstanzen wiederholen, nicht erlaubt, zum Beispiel in Kopf- oder Fußzeilen von Tabellen, die mehrere Instanzen erfordern.

Hinzufügen von Tags und Attributen zu Tabelleninhalten. Sie können die Option *tag* als Unteroption an folgende Optionen oder entsprechende Unteroptionen der Option *caption* von *PDF_add_table_cell( )* übergeben:

```
fitannotation, fitfield, fitgraphics, fitimage, fitpath,
fitpdipage, fittextflow, fittextline
```

Dies ist in folgenden Fällen nützlich:

- Um die Unterstruktur für den Inhalt von Tabellenzellen zu erstellen. Die Option *tag* erstellt ein Unterelement der Elemente *TH* oder *TD* der Zelle. Die folgenden Werte für *tagname* sind bei der Option *tag* nicht erlaubt, wenn das automatische Erstellen von Tabellen-Tags aktiviert ist, das heißt, dass verschachtelte Tabellen nicht unterstützt werden:

  ```
  Table, TR, TH, TD, THead, TBody, TFoot
  ```

- Die unten aufgeführten Tabellenattribute können nicht automatisch erstellt werden, sind aber für die Barrierefreiheit erforderlich. Sie müssen diese explizit angeben:
  - Die Option *Headers* muss bei *TD*-Zellen angegeben werden, die sich auf eine oder mehrere *TH*-Zellen in der Tabelle beziehen (das heißt, sie beziehen sich auf Kopfzellen, die nicht Teil der Option *header* von *PDF_fit_table( )* sind).
  - Die Optionen *Id* und *Scope=Row* müssen bei *TH*-Zellen angegeben werden, die in einer Option *Headers* referenziert werden *(Scope=Column* wird bei *TH*-Kopfspalten automatisch erzeugt).
- Eine Tabellenüberschrift kann beliebigen Inhalt haben, der wiederum mit Tags versehen werden kann. Die folgende Optionsliste erzeugt ein *Caption*-Element mit einer einzelnen Textzeile innerhalb eines verschachtelten *P*-Elements:

  ```
  caption={ fittextline={tag={tagname=P title={Travel Expense Report}} ... } ... }
  ```

10.4.2 Interaktive Elemente

Links, Anmerkungen und Formularfelder müssen ebenfalls zugänglich gemacht werden. Die entsprechenden interaktiven Elemente müssen immer im Strukturbaum an der richtigen Stelle dargestellt werden. Sie können nicht als Artefakte ausgezeichnet werden.

Cookbook *Codebeispiele zum Anbringen von Tags für Links und Formularfelder finden Sie im Topic* `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

Links und andere Arten von Anmerkungen. Die folgenden Voraussetzungen müssen erfüllt sein, damit Anmerkungen für Screenreader zugänglich werden (siehe Abbildung 10.5):

- Ein *Link*- oder *Annot*-Strukturelement muss als Container für die nächsten beiden Elemente in logischer Lesereihenfolge erstellt werden. Die Optionen *Alt* oder *ActualText* für Alternativtext können angegeben werden. Das Attribut *Alt* des *Link*-Elements sollte den Zweck des Links beschreiben. Beachten Sie, dass Pseudo-Elemente, Gruppierungselemente, Tabellenelemente, ILSEs, Ruby- und Warichu-Elemente nicht als übergeordnete Elemente von *Link* und *Annot* erlaubt sind.
- Sichtbarer Inhalt des interaktiven Elements, zum Beispiel Text oder ein Bild, sollte innerhalb des Container-Elements platziert werden. Wenn kein Seiteninhalt vorhanden ist, wie etwa für eine Textanmerkung, kann dieses Element entfallen. Sie sollten die Option *matchbox* der verschiedenen Funktionen zum Erstellen von Inhalt verwenden, um die Geometrie-Information für das nächste Element vorzubereiten.
- Für interaktive Anmerkungen oder Formularfelder müssen ein oder mehrere Strukturelemente vom Typ *OBJR (object reference)* gesetzt sein. OBJR-Elemente werden automatisch durch den Aufruf von *PDF_create_annotation( )* erzeugt. Ein *OBJR*-Element wird als untergeordnetes Element des gerade aktiven Elements eingefügt, es sei denn, in der Unteroption *parent* der Option *tag* ist ein anders übergeordnetes Element angegeben. Die Option *contents* von *PDF_create_annotation( )* sollte für Link-Anmerkungen angegeben werden; dies ist für PDF/UA erforderlich. Andere Anmerkungstypen sollten mit der Option *contents* von *PDF_create_annotation( )* oder der Tag-Option *ActualText* erstellt werden. Mit der Option *usematchbox* lässt sich die Geometrie des im zweiten Schritt erstellten, sichtbaren Inhalts übergeben.

Die Erstellung der Elemente in Schritt zwei und drei kann vertauscht werden. Während das *OBJR*-Element automatisch erstellt wird, müssen Sie für die Erstellung der anderen benötigten Elemente selbst sorgen. Die obigen Anforderungen für Anmerkungen gelten auch für Anmerkungen in Tabellenzellen, die mit der Option *fitannotation* von *PDF_add_table_cell( )* erstellt wurden.

Mit dem folgenden Codefragment lässt sich ein interaktiver Link mit den drei erforderlichen Elementen erstellen; Abbildung 10.5 zeigt die daraus resultierende Tag-Struktur:

```
/* Erstellen des übergerodneten Link-Elements */
id_link = p.begin_item("Link", "Title={Kraxi on the Web} Alt={Kraxi on the Web}");

/* Erstellen des sichtbaren Inhalts für den Link */
p.fit_textline("Click here to go to the Kraxi website", x, y,
        "matchbox={name={kraxi.com}} fontsize=14 font=" + font);

/* Erzeugen der URI-Aktion */
action = p.create_action("URI", "url={http://www.kraxi.com}");

/* Erstellen der Linkanmerkung der Matchbox "kraxi.com" */
p.create_annotation(0, 0, 0, 0, "Link",
        "action={activate=" + action + "} "
        "linewidth=0 usematchbox={kraxi.com} contents={Link to Kraxi Inc. Web site}");

p.end_item(id_link);
```

Hinweis *Die für interaktive Elemente erforderliche Tag-Sequenz lässt sich mit Textflow und Matchboxen nicht erzeugen, da Tags nicht innerhalb eines Textflows erstellt werden können und das Erstellen dieser Tags nach Platzierung des Textflows die logische Lesereihenfolge zerstören würde.*

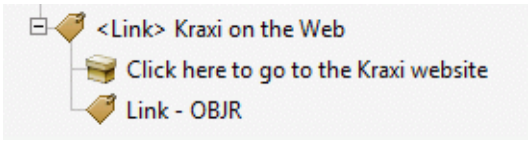


Abb. 10.5
Darstellung eines zugänglichen Links im Strukturbaum

Tags für Links in Tabellenzellen. Links in einer Tabellenzelle benötigen die oben beschriebene Tag-Struktur. Die Strukturelemente *TH/TD, Link, OBJR* sowie möglichen Inhalt korrekt zu verschachteln, kann etwas kompliziert sein. Deshalb übergeben Sie am besten die Option *tag* an *PDF_add_table_cell( )* und nutzen die dabei mögliche Verschachtelung von Tags. Die Optionsliste unten befüllt eine Tabellenzelle mit einer Textzeile und einer Linkanmerkung. Das umschließende *TD*-Element wird in der äußeren *tag*-Option übergeben (da *TD* automatisch durch den Tabellenformatierer erstellt wird, kann die äußere Unteroption *tagname* weggelassen werden). Das Element *Link* wird mit der inneren *tag*-Option übergeben. Abschließend wird das erforderliche *OBJR*- Element mit der Option *fitannotation* automatisch erzeugt, die äquivalent zu *PDF_create_annotation( )* ist:

```
fittextline={font=... fontsize=25 fillcolor=blue}
annotationtype=Link fitannotation={contents={Kraxi home page} action={activate ...}}
tag={tagname=TD tag={tagname=Link}}
```

Formularfelder. Formularfelder benötigen zwei Strukturelemente, um sie zugänglich zu machen:

- Ein *Form*-Strukturelement muss als Container für das nächste Element in logischer Lesereihenfolge erstellt werden. Die Optionen *Alt* oder *ActualText* für Alternativtext können angegeben werden. Beachten Sie, dass Pseudo-Elemente, Tabellenelemente, ILSEs, Ruby- und Warichu-Elemente nicht als übergeordnete Elemente von *Form* erlaubt sind. Bei der Erstellung von Optionsfeldern (radio buttons) ist für die Optionsfeldgruppe in *PDF_create_fieldgroup( )* kein *Form*-Element erforderlich, sondern nur für die einzelnen Optionsfelder in *PDF_create_field( )*.
- Ein Strukturelement vom Typ *OBJR* (*object reference*). OBJR-Elemente werden automatisch durch den Aufruf von *PDF_create_field( )* erzeugt. Ein *OBJR*-Element wird als untergeordnetes Element des gerade aktiven Elements eingefügt, es sei denn, in der Unteroption *parent* der Option *tag* ist ein anderes übergeordnetes Element angegeben. Die Option *tooltip* von *PDF_create_field( )* sollte für die bessere Zugänglichkeit von Formularfeldern angegeben werden; dies ist für PDF/UA erforderlich.

Während das *OBJR*-Element automatisch mit *PDF_create_field( )* erstellt wird, müssen Sie für die Erstellung des *Form*-Elements selbst sorgen. Beide Elemente können vereinfacht über einen einzigen Aufruf von *PDF_create_field( )* mit Tags versehen werden, das heißt, durch Angabe der Option *tag* mit *tagname=Form:*

```
p.create_field(p, llx, lly, urx, ury, "firstname", "textfield",
        "bordercolor={gray 0} font=" + font + "tag={tagname=Form} tooltip={First name}");
```

Die obigen Anforderungen für Formularfelder gelten auch für Formularfelder in Tabellenzellen, die mit der Option *fitfield* von *PDF_add_table_cell( )* erstellt wurden.

Strukturierte Lesezeichen. Einem Lesezeichen kann zusätzlich ein Strukturelement zugeordnet werden. Solche Lesezeichen werden *Strukturierte Lesezeichen* genannt, und Acrobat bietet dafür zusätzliche Funktionen an. Beim Rechtsklick auf ein strukturiertes Lesezeichen in Acrobat werden die Funktionen *Seite(n) löschen* und *Seite(n) extrahieren* angeboten, die für die Seite(n) gelten, auf denen sich das Strukturelement befindet. Strukturierte Lesezeichen erzeugen eine Verbindung zwischen einem Lesezeichen und einem Strukturelement. Diese Verbindung kann auf zwei Arten hergestellt werden:

- Erzeugen Sie ein Lesezeichen mit *PDF_create_bookmark( )* und übergeben Sie sein Handle an die Option *bookmark* von *PDF_begin_item( )* oder an die Option *tag* der jeweiligen Funktionen:

  ```
  bm = p.create_bookmark("Section 1", "");
  id = p.begin_item("H1", "Title={Section 1} bookmark=" + bm);
  p.fit_textline(text, x, y, "");
  p.end_item(id);
  ```

  Dieses Vorgehen kann auch zusammen mit dem vereinfachten Anbringen von Tags verwendet werden:

  ```
  bm = p.create_bookmark("Section 1", "");
  p.fit_textline(text, x, y,
       "tag={tagname=H1 Title={Section 1} bookmark=" + bm + "}");
  ```

  Der Nachteil an diesem Vorgehen ist, dass der Lesezeichentext bereits vorhanden sein muss, bevor das Strukturelement und sein Inhalt erzeugt werden. Das kann besonders ungünstig sein, wenn die referenzierten Strukturelemente weiter oben im Strukturbaum liegen.
- Erzeugen Sie ein Strukturelement mit *PDF_begin_item( )* und übergeben Sie seinen Handle an die Option *item* von *PDF_create_bookmark( )*. Anstelle des Handles kann das Schlüsselwort *current* als Abkürzung verwendet werden, welches das beim Aufruf von *PDF_create_bookmark( )* gerade aktive Strukturelement referenziert:

  ```
  id = p.begin_item("H1", "Title={Section 1} ");
  bm = p.create_bookmark("Section 1", "item=current");
  p.fit_textline(text, x, y, "");
  p.end_item(id);
  ```

  Dieses Vorgehen hat den Vorteil, dass der Lesezeichentext erst vorhanden sein muss, wenn das Strukturelement und sein Inhalt erzeugt werden. Das vereinfachte Anbringen von Tags kann jedoch in diesem Fall nicht verwendet werden.

Strukturierte Lesezeichen können nur bereits geöffnete Strukturelemente und keine Pseudo- oder Inline-Elemente referenzieren. Im Clientcode muss sichergestellt sein, dass das Ziel des Lesezeichens dem Strukturelement entspricht (bei der Aktivierung des Lesezeichens in Acrobat würde sonst nicht das Element, sondern eine andere Position im Dokument angesprungen). Wenn sich das zugehörige Strukturelement über mehr als eine Seite erstreckt, sollte das Lesezeichen auf die erste Seite des entsprechenden Bereichs zeigen.

10.4.3 Listen

Mit Listen lassen sich verwandte Elemente gruppieren. Sie werden durch folgende Strukturelemente dargestellt (siehe Abbildung 10.6):

- Das *L*-Element enthält alle folgenden Strukturelemente. Mit der Option *List-Numbering* kann das Nummerierungssystem für die *Lbl*-Elemente festgelegt werden. Selbst ohne *Lbl*-Elemente ist die Option *ListNumbering* für Screenreader nützlich.
- Ein optionales Element *Caption*. Als Gruppierungselement kann *Caption* keinen direkten Inhalt, sondern nur weitere Strukturelemente enthalten (zum Beispiel *P)*.
- Ein oder mehrere Listenelemente *(LI)* mit folgendem Inhalt:
  - Ein optionales Label *(Lbl)* mit einem Aufzählungszeichen, einer Nummer o.ä.
  - Ein Element *LBody* mit dem eigentlichen Inhalt der Listenelemente. *LBody* kann entweder direkten Inhalt oder andere Strukturelemente einschließlich einer verschachtelten Liste enthalten.

Mit dem folgenden Codefragment wird eine Liste mit einem Einleitungssatz *(caption)* und drei Listenelementen erstellt. Jedes Listenelement beginnt mit einem als Label markierten Aufzählungszeichen U+2022. Abbildung 10.6 zeigt die Tag-Struktur, die sich daraus ergibt:

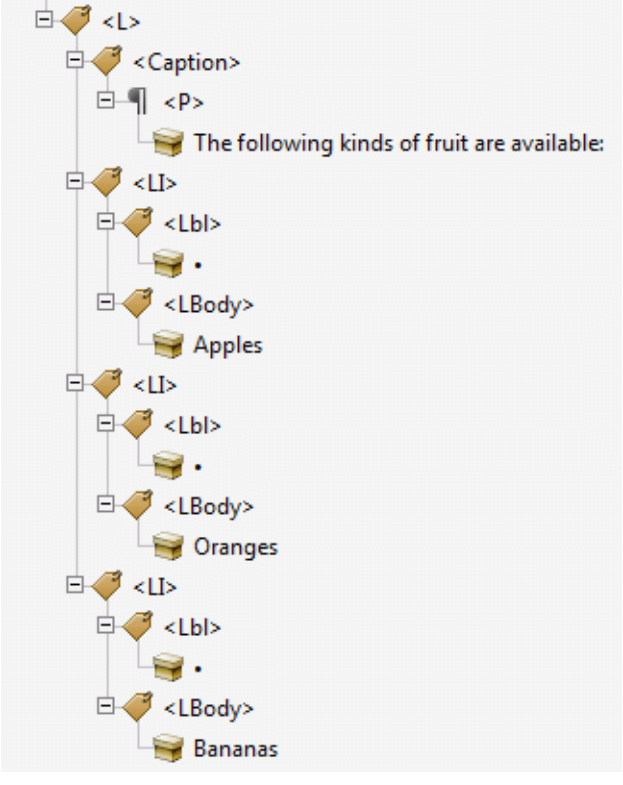


Abb. 10.6
Darstellung einer zugänglichen Liste im Strukturbaums

```
id_list = p.begin_item("L", "ListNumbering=Disc");

id_caption = p.begin_item("Caption", "");
    p.fit_textline("The following kinds of fruit are available:",
        x1, y, "tag={tagname=P}");
    y -= leading;
p.end_item(id_caption);

id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Apples", x2, y, "tag={tagname=LBody}"); y -= leading;
p.end_item(id_listitem);

id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Oranges", x2, y, "tag={tagname=LBody}"); y -= leading;
p.end_item(id_listitem);

id_listitem = p.begin_item("LI", "");
    p.fit_textline("&#x2022;", x1, y, "tag={tagname=Lbl}");
    p.fit_textline("Bananas", x2, y, "tag={tagname=LBody}"); y -= leading;
p.end_item(id_listitem);

p.end_item(id_list);
```

Cookbook *Ein Codebeispiel zur Erzeugung eines Tagged PDF finden Sie im Topic* `tagged_list` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

10.4.4 Erzeugung von Seiteninhalt in abweichender Reihenfolge

Wie bereits in Abschnitt 10.3.6, »Druckreihenfolge und logische Lesereihenfolge«, Seite 296 erwähnt, ist es von entscheidender Bedeutung, die Elemente in der Strukturhierarchie in logischer Lesereihenfolge zu erstellen. PDFlib bietet verschiedene Funktionen, um die logische Lesereihenfolge in der Strukturhierarchie zu erhalten, falls die Anwendung Seiteninhalte in einer anderen Reihenfolge erstellt (zum Beispiel immer von oben nach unten ohne die Beziehung der Spalten zueinander zu berücksichtigen):

- Erstellen von untergeordneten Strukturelementen in abweichender Reihenfolge mit der Option *index*. Damit lässt sich die Position ändern, an der ein neues Strukturelement innerhalb seines übergeordneten Elements eingefügt wird.
- Erstellen von Strukturelementen in abweichender Reihenfolge mit der Option *parent*. Damit lässt sich das übergeordnete Element ändern, in das ein neues Strukturelement eingefügt wird.
- Vor- und Zurückspringen in der Strukturhierarchie mit der Funktion *PDF_activate_item( )*. Damit lassen sich weitere Strukturelemente oder direkter Seiteninhalt zu einigen Elementen in der Strukturhierarchie hinzufügen.

Diese Möglichkeiten werden im Folgenden ausführlicher dargestellt.

Cookbook *Codebeispiele zum Auszeichnen von Elementen in abweichender Reihenfolge finden Sie in den Topics* `tag_out_of_order` *und* `tag_parallel_columns` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

Erzeugung von untergeordneten Elementen in abweichender Reihenfolge. Um untergeordnete Elemente innerhalb eines Strukturelements in abweichender Reihenfolge zu erzeugen, können Sie eine Position in der Baumstruktur mit der Option *index* von *PDF_*

begin_item () oder den Unteroptionen der *tag*-Option der verschiedenen Funktionen angeben. Das folgende Codefragment gibt Textfragmente in umgekehrter Reihenfolge aus und korrigiert die Reihenfolge im Strukturbaum durch Einsetzen jedes neuen Textfragments als jeweils erstes untergeordnetes Element *(index=0)* des übergeordneten Elements. Da jedes neue Element als jeweils erstes untergeordnetes Element des übergeordneten eingefügt wird, entsteht die logische Reihenfolge umkehrt zur Reihenfolge ihrer Erstellung:

```
p.fit_textline("three", x, y, "tag={tagname=P index=0}");
y += leading;
p.fit_textline("two",   x, y, "tag={tagname=P index=0}");
y += leading;
p.fit_textline("one",   x, y, "tag={tagname=P index=0}");
```

Mit *PDF_get_option( )* und den Schlüsselwörtern *activeitemindex* oder *activeitemkidcount* können Sie den Index des aktiven Tags innerhalb seines übergeordneten Elements abfragen und später an diese Stelle im Strukturbaum zurückkehren. Mit dem folgenden Codefragment wird ein neues Element hinter dem Element am gespeicherten Index eingefügt:

```
nextindex = p.get_option("activeitemindex") + 1;

...Erzeugung weiterer Elemente auf gleicher Ebene...

p.fit_textline(text, x, y, "tag={tagname=P index=" + nextindex + "}");
```

Erzeugung von Strukturelementen in abweichender Reihenfolge. Mit der Option *parent* können Sie untergeordnete Elemente an einer anderen als der aktuellen Position im Strukturbaum erzeugen. Die Option muss sich auf ein noch nicht geschlossenes Strukturelement beziehen. Elemente, die mit vereinfachtem Anbringen von Tags erstellt wurden, können hier nicht als Ziel verwendet werden, da sie innerhalb desselben Funktionsaufrufs erzeugt und geschlossen werden. Mit *PDF_get_option( )* und dem Schlüsselwort *activeitemid* können Sie die Id des aktiven Tags innerhalb seines übergeordneten Elements abfragen und später an diese Stelle im Strukturbaum zurückkehren:

```
parent_id = p.get_option("activeitemid");
...
p.fit_textline(text, x, y, "tag={tagname=P parent=" + parent_id + "}");
```

Die Optionen *parent* und *index* können kombiniert werden. Mit *PDF_suspend/resume_page( )* kann man eine Seite unterbrechen, auf einer anderen Seite fortfahren und dann wieder an die ursprüngliche Stelle zurückkehren, um dort weiteren Inhalt einzufügen.

Aktivierung von Elementen für komplexe Layouts. Um die Erstellung von Strukturinformationen bei komplexen nicht-linearen Seitenlayouts zu erleichtern, unterstützt PDFlib die sogenannte Elementaktivierung. Damit lässt sich ein bereits erzeugtes Strukturelement nachträglich aktivieren. Dies ist in Situationen sinnvoll, wo Entwickler den Überblick über mehrere Strukturbäume behalten müssen, die sich jeweils über eine oder mehrere Seiten erstrecken. Von Vorteil ist dieses Verfahren zum Beispiel in folgenden Fällen:

- Die Seite besteht aus mehreren Spalten.
- Der Haupttext ist unterbrochen, z.B. durch Kurzübersichten oder Einschübe.

- Tabellen und Abbildungen sind zwischen den Spalten platziert.

Mit der Funktion *PDF_activate_item( )* können Sie zwischen den verschiedenen Zweigen des Strukturbaums hin- und herschalten. Für die logische Lesereihenfolge muss die Clientanwendung den Seiteninhalt auch dann in logischer Anordnung aufbauen, wenn die visuelle Reihenfolge einfacher zu erzeugen wäre. Bei der Elementaktivierung kann der Inhalt im Gegensatz dazu in visueller oder einer anderen für die Anwendung passenden Reihenfolge aufgebaut werden. Dasselbe Verfahren kommt zum Einsatz, wenn sich der Inhalt über mehrere Seiten erstreckt.

Um Fehler in Acrobat zu umgehen, sollte deshalb direkt nach dem Aufruf von *PDF_activate_item( )* kein direkter Seiteninhalt, sondern nur andere Strukturelemente hinzugefügt werden.

Abfrage des aktiven Strukturelements. Um die Optionen *parent* und *index* oder *PDF_activate_item( )* zu verwenden, muss das gerade aktive Strukturelement mit seinen Unterelementen bekannt sein. Diese Statusinformation kann von der Anwendung verwaltet werden oder über PDFlib abgefragt werden. Die Funktion *PDF_get_option( )* mit den Schlüsselwörtern *activeitemid, activeitemindex, activeitemkidcount, activeitemname, activeitemstandardname* gibt Id, Index, Anzahl der untergeordneten Elemente, Namen und Standardnamen (bei Rollenzuordnungen) des aktuellen Elements aus.

10.4.5 Import von Tagged PDF mit PDI

Cookbook *Codebeispiele für das Importieren von Seiten aus Tagged PDF finden Sie in den Topics* `clone_pdfua` *und* `merge_and_stamp_pdfua` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

Im Tagged-PDF-Modus werden Seiten eines Tagged PDF zusammen mit ihren Strukturelement-Tags importiert. Wir kürzen eine auf diese Weise mit *usetags=true* importierte Seite als »Seite mit Tags« ab. Diesen Status können Sie mit dem Schlüsselwort *tagged* von *PDF_info_pdi_page( )* abfragen. Das Importieren von Seiten mit Tags wird im Folgenden beschrieben.

Öffnen eines PDF-Dokuments mit Tags. *PDF_open_pdi_document( )* prüft, ob das importierte Dokument kompatibel ist mit dem Modus von PDF/A-1a/2a/3a oder PDF/UA und liest den Strukturbaum des importierten Dokuments ein.

Wenn die Option *usetags* auf *false* gesetzt ist, wird die Strukturinformation des Dokuments ignoriert, und es können keine Tags aus dem Dokument importiert werden.

Hinweis *Attributklassen und Klassenzuordnungen werden nicht importiert.*

Klonen der Eingabesprache des Dokuments. Wenn die meisten oder alle Seiten eines PDF-Dokuments mit PDI importiert werden, empfehlen wir, die Angabe der Dokumentsprache zu klonen, sofern sie im Eingabedokument vorhanden ist (siehe Abschnitt »Angabe der Sprache«, Seite 293). Die Dokumentsprache lässt sich mit pCOS und dem folgenden Codefragment klonen:

```
if (p.pcos_get_string(indoc, "type:/Root/Lang").equals("string"))
{
        inputlang = p.pcos_get_string(indoc, "/Root/Lang");
        optlist += " lang=" + inputlang;
}
```

```
p.begin_document(filename, optlist);
```

Öffnen einer Seite mit Tags. *PDF_open_pdi_page( )* wählt die Strukturelemente des importierten Seiteninhalts aus und filtert die auf der Seite vorhandenen Tags aus. Tags für Anmerkungen werden beispielsweise entfernt, da PDI keine interaktiven Elemente importiert. Schließlich werden ein oder mehrere Elemente auf der importierten Seite ausgewählt, die die Spitze des importierten Unterbaumes bilden. Wenn die Option *usetags* auf *false* gesetzt ist, wird die Strukturinformation der Seite ignoriert.

Einträge in der importierten Rollenzuordnung des Dokuments werden in die Rollenzuordnung des Ausgabedokuments kopiert, wenn das entsprechende Element auf der Seite verwendet wird. Widersprüchliche Rollenzuordnungen werden ignoriert, das heißt, wenn ein benutzerdefiniertes Tag bereits eine Rollenzuordnung auf ein anderes Standardtag in der generierten Rollenzuordnung des Dokuments oder einem zuvor importierten Dokument hat. Seiten mit widersprüchlichen Rollenzuordnungen werden im PDF/UA-Modus allerdings abgelehnt, das heißt, der Aufruf von *PDF_open_pdi_page( )* schlägt fehl.

Import von Dokumenten mit ungültiger Tag-Struktur. In PDFlib sind strenge Prüfungen für die Verschachtelungsregeln von Tags gemäß ISO 32000-1 implementiert; für weitere Informationen siehe Abschnitt »Verschachtelungsregeln für Strukturelemente«, Seite 288. Diese Prüfungen können Sie auch auf importierte Dokumente und die aus importierten Seiten erzeugte Tag-Struktur anwenden. Die Verschachtelungsregeln werden auf importierte Seiten standardmäßig nicht angewandt. Sie können diese mit der Option *checktags* von *PDF_open_pdi_document( )* aktivieren. Bei *checktags=strict* werden alle Verschachtelungsregeln für Tags bei *PDF_open_pdi_page( )* überprüft. Wenn die Strukturhierarchie der importierten Seiten die Verschachtelungsregeln für Strukturelemente verletzt, schlägt der Aufruf von *PDF_open_pdi_page( )* fehl und *PDF_get_errmsg( )* gibt eine Fehlermeldung ähnlich der folgenden aus:

```
Grouping element type 'Document' cannot contain direct content
(but only other structure elements)
```

Da viele mit Tags versehene PDF-Dokumente aus der Praxis die Verschachtelungsregeln für Tags verletzen, können Sie diese Probleme in importierten Dokumenten mit einer der folgenden Methoden beheben:

- Durch Einfügen eines zusätzlichen Tags als oberstes Element der importierten Strukturhierarchie (zum Beispiel mit der Option *tags* von *PDF_fit_pdi_page( )*) lassen sich häufige Probleme mit importierten Seiten beheben, die gleich unter dem obersten Element *(root)* direkten Inhalt enthalten.
- Andere Probleme können durch das Einfügen eines zusätzlichen Tags dagegen nicht behoben werden, zum Beispiel unvollständige Strukturelemente für Tabellen oder Aufzählungen. Sie sollten, wenn möglich, das Eingabedokument korrigieren. Wenn dies keine angemessene Lösung ist, können Sie in *PDF_open_pdi_document( )* *checktags* auf *none* setzen, um Seiten aus Tagged PDF mit nicht konformer Tag-Struktur zu importieren.
- Wenn die importierte Struktur selbst korrekt ist, aber mit den generierten neuen Tags des Ausgabedokuments zu Konflikten führt, sollten Sie versuchen, die neuen Tags entsprechend anzupassen. Wenn dies nicht möglich ist, können Sie *checktags*

auf *none* setzen, damit Konflikte in der generierten Tag-Struktur ignoriert werden. Dies ist im PDF/UA-Modus allerdings nicht erlaubt.

Da nicht konforme Eingabe, die mit der jeweiligen Variante von *checktags=none* verarbeitet wird, zu nicht konformer PDF-Ausgabe führen kann, ist diese Einstellung nicht zu empfehlen.

Abfrage und Prüfung der Tags einer importierten Seite. Manchmal ist es schwierig, die importierten Strukturelemente richtig in die neu generierte Strukturhierarchie zu integrieren. Um diesen Prozess zu erleichtern, können Sie bei einer geöffneten Seite eines importierten Tagged PDF verschiedene Eigenschaften mit *PDF_info_pdi_page( )* abfragen:

- Das Schlüsselwort *fittingpossible* gibt an, ob die Seite im aktuellen Kontext platziert werden kann. Bei Angabe der Option *tag* können Sie prüfen, ob die Seite mit einem zusätzlichen Top-Level-Tag platziert werden kann. Nur die Unteroption *tagname* der Option *tag* wird ausgewertet; andere Unteroptionen sollten Sie nicht angeben. Dieses Schlüsselwort sollten Sie verwenden, wenn Sie bei der Strukturinformation einer importierten Seite unsicher sind und das Auslösen einer Exception bei *PDF_fit_pdi_page( )* wegen ungeeigneter importierter Strukturelemente vermeiden wollen. Wenn eine Seite beim Test mit *fittingpossible* abgelehnt wird, können Sie versuchen, ein weiteres Tag über die *tag*-Option einzufügen.
- Das Schlüsselwort *topleveltagcount* gibt die Anzahl der Top-Level-Strukturelemente aus, da es mehr als eine Top-Level-Tag geben kann. Beachten Sie, dass *topleveltagcount* in seltene Fällen gleich 0 sein kann, wenn der Seiteninhalt durch kein Stukturelement abgedeckt ist. Diese Seiten werden wie Seiten ohne Tags behandelt. Wie anderer direkter Inhalt können sie nicht als Unterelement eines Gruppierungselements platziert werden, sondern erfordern ein zusätzliches, die Seite einschließendes Tag.
- Das Schlüsselwort *topleveltag* gibt die Top-Level-Strukturelemente der importierten Seite an. Dies kann nützlich sein, um zu bestimmen, ob zusätzliche Strukturelemente über der importierten Seitenstruktur eingefügt werden können oder müssen.
- Das Schlüsselwort *lang* gibt den Wert für das Attribut *lang* aller importierten Top-Level-Strukturelemente aus. Damit lässt sich bestimmen, ob ein *lang*-Attribut bei übergeordneten Strukturelementen erforderlich ist.

Platzieren einer Seite mit Tags. *PDF_fit_pdi_page( )* platziert die importierte Seite auf einer neuen Seite und integriert ihre Strukturhierarchie in die generierte Dokumentstruktur, wobei das aktive Element als übergeordnetes Element für den importierten Strukturbaum verwendet wird. Mit der *tag*-Option von *PDF_fit_pdi_page( )* lässt sich ein zusätzliches Strukturelement erzeugen, das als neues übergeordnetes Element für die importierte Strukturhierarchie dient.

Eventuell in der importierten Hierarchie vorhandene Attribute *Alt* oder *ActualText* werden entfernt, wenn sie nicht mit den Attributen übergeordneter Elemente der generierten Dokumentstruktur vereinbar sind (siehe Abschnitt »Verschachtelungsregeln für Alternativ- und Ersatztext«, Seite 295).

Wenn die Seite mit der Option *usetags=true* von *PDF_open_pdi_document( )* und *PDF_open_pdi_page( )* geöffnet wird, kann sie höchstens einmal im Ausgabedokument platziert werden, da die importierte Struktur nur an einer einzigen Stelle in der Strukturhierarchie des Ausgabedokuments vorkommen darf. Wenn Seiteninhalt und -struktur

mehrfach vorkommen, kann die Seite auch mehrfach geöffnet werden (das heißt, es werden unterschiedliche Seiten-Handles platziert).

Wir empfehlen das folgende Vorgehen bei importierten Seiten mit unbekannter Dokumentstruktur:

- Wenn das Schlüsselwort *topleveltagcount* von *PDF_info_pdi_page( )* die Anzahl der Tags mit 0 angibt, ist die Seite entweder leer oder enthält nur Artefakte. Bei einigen Anwendungen werden solche Seiten übersprungen, da ihr Inhalt keine Relevanz hat. Wenn die Seite trotzdem importiert wird, sollten Sie ein zusätzliches *Artifact*-Tag angeben, um einen Fehler in Acrobat zu umgehen.
- Wenn das Schlüsselwort *fittingpossible* von *PDF_info_pdi_page( )* den Wert 1 für diese Seite ausgibt, kann sie mit *PDF_fit_pdi_page( )* platziert werden. Dieser Test ist bei Seiten, die als Artefakte platziert werden, nicht erforderlich.
- Ansonsten kann ein zusätzliches Tag der importierten Seitenstruktur übergeordnet werden. Die Wahl dieses Tags hängt von der Anwendung ab, besonders vom Typ des Strukturelements, bei dem die importierte Seite im Strukturbaum eingefügt wird.
- Wenn das Schlüsselwort *fittingpossible* von *PDF_info_pdi_page( )* die Seite mit dem zusätzlichen Tag immer noch zurückweist, kann die Anwendung entweder ein anderes Tag ausprobieren oder die Seite ablehnen.

Mit dem folgenden Codefragment wird die oben skizzierte Strategie implementiert. Ein weiteres *P*-Element wird eingefügt, wenn die Seite nicht direkt platziert werden kann:

```
fittingpossible = true;
additionaltag = "";

topleveltagcount = (int) p.info_pdi_page(page, "topleveltagcount", "");

if (topleveltagcount == 0)
{
        /* Die Seite enthält keine Strukturelemente,
         * das heißt, sie ist leeer oder enthält nur Artefakte.
         * Das Tag "Artifact" wird hinzugefügt, um einen Acrobat-Fehler zu umgehen.
         */
        additionaltag = "tag={tagname=Artifact} ";
}
else
/*
* Versuch, die Seite ohne ein weiteres Tag zu platzieren;
* wenn dies fehlschlägt, wird ein weiteres Tag eingefügt.
*/
if (p.info_pdi_page(page, "fittingpossible", "") == 0)
{
        additionaltag = "tag={tagname=P} ";

        if (p.info_pdi_page(page, "fittingpossible", additionaltag) == 0)
        {
            fittingpossible = false;
        }
}

if (fittingpossible)
{
        p.fit_pdi_page(page, 0, 0, "adjustpage " + additionaltag);
}
else
```

```
{
        System.err.println("Skipping page: " + p.get_errmsg());
}
```

Anwendungsfälle. Die folgenden Anwendungsfälle für importierte Seiten im Tagged-PDF-Modus können unterschieden werden; Tabelle 10.6 gibt eine Übersicht über Optionen und Voraussetzungen für diese Anwendungsfälle:

- Tags übernehmen: eine Seite wird aus einem mit Tags versehenen PDF-Dokument importiert und ihre Tags werden in die Ausgabe kopiert. Die Strukturelemente der Seite werden Teil der neuen Strukturhierarchie. Optional kann ein zusätzliches Tag oberhalb der importierten Hierarchie platziert werden.
- Tags entfernen: eine Seite wird aus einem PDF-Dokument mit oder ohne Tags importiert und die gesamte Seite wird mit einem neuen Tag ausgezeichnet. Vorhandene Strukturelemente werden entfernt und der Inhalt der importierten Seite besteht aus einem einzigen Tag ohne innere Struktur.
- Als Artefakt platzieren: eine Seite wird aus einem PDF-Dokument mit oder ohne Tags importiert und die gesamte Seite wird als Artefakt platziert, zum Beispiel bei Hintergrundbildern. Vorhandene Strukturelemente werden entfernt.

Tabelle 10.6 Import von PDF-Seiten mit und ohne Tags im Tagged-PDF-Modus

Anwendungsfall	***importierte Seite***	***Optionen***	***Voraussetzungen und Kommentare***
Tags übernehmen	*mit Tags*	`usetags=true` *in PDF_open_pdi_document( ) und PDF_open_pdi_page( )*	► *übergeordnetes Element[1] darf kein Inline- oder Pseudoelement sein* ► *importierte Seite kann nur einmal platziert werden* ► *Verschachtelungsregeln für Top-Level-Element(e) müssen eingehalten werden*
Tags entfernen	*mit oder ohne Tags[2]*	`usetags=false` *in PDF_open_pdi_document( ) oder PDF_open_pdi_page( )*	► *Inhalt der importierten Seite wird Teil des gerade aktiven Elements* ► *nur übergeordnete Nicht-Gruppierungselemente erlaubt*
als Artefakt platzieren	*mit oder ohne Tags*	`tag={tagname=Artifact}` *in PDF_fit_pdi_page( )*	► *Inhalt der importierten Seite wird Artefakt*

1. Aktives Tag, das mit PDF_begin_item( ) oder der Option `tag` *von PDF_fit_pdi_page( ) festgelegt wurde*
2. Diese Situation kann auch bei Seiten eines Tagged PDF auftreten, auf denen sich kein Strukturelement auf den Seiteninhalt bezieht.

10.4.6 PDFlib-Techniken für WCAG 2.0

Die vom W3C veröffentlichten PDF-Techniken für WCAG 2.0 finden Sie auf folgender Seite:

`www.w3.org/TR/WCAG20-TECHS/pdf.html`

Diese W3C-Empfehlung beschreibt die PDF-Features, die zur Erreichung der WCAG 2.0-Konformität erforderlich sind und erklärt, wie diese mit Adobe Acrobat und Autorenwerkzeugen wie Microsoft Word und OpenOffice erzeugt werden können. In diesem Abschnitt erläutern wir, wie man die WCAG-Ziele mit PDFlib erreicht. Nummerierung und Zielbeschreibungen sind wörtlich der oben angeführten W3C-Empfehlung entnommen.

Beachten Sie, dass die PDF-Techniken für WCAG 2.0 nicht alle technischen Anforderungen für barrierefreie PDF-Dokumente vollständig abdecken. Diese Lücke wird durch

den PDF/UA-Standard, der technische Spezifikationen für barrierefreies PDF enthält, geschlossen. Für weitere Informationen zu diesem Standard siehe Abschnitt 11.6, »PDF/UA für Barrierefreiheit«, Seite 356.

Wenn die Option *tag* erwähnt wird, kann auch ein alternativer Aufruf zu *PDF_begin_item( )* gewählt werden, in manchen Fällen auch umgekehrt (jedoch nicht in allen Fällen, da mit der *tag*-Option keine Gruppierungselemente erzeugt werden können). Gegebenenfalls wird auf entsprechenden Beispielcode im PDFlib Cookbook hingewiesen.

PDF1: Alternativtext für Bilder mit dem Eintrag Alt in PDF-Dokumente einfügen. Dieses Ziel lässt sich mit der Unteroption *Alt* der Option *tag* von *PDF_fit_image( ), PDF_fit_graphics( )* oder *PDF_fit_pdi_page( )* erreichen:

```
p.fit_image(image, x, y, "tag={tagname=Figure Alt={Young girl sitting at a table}} ...");
```

Für weitere Informationen siehe Abschnitt 10.3.5, »Alternativtext, Ersatztext und Abkürzungsexpansion«, Seite 295.

Cookbook Codebeispiele finden Sie im Topic `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF2: Erzeugen von Lesezeichen in PDF-Dokumenten. Dieses Ziel lässt sich mit *PDF_create_bookmark( )* erreichen:

```
bm = p.create_bookmark(text, optlist);
```

Optional können Sie strukturierte Lesezeichen erstellen (siehe Abschnitt »Strukturierte Lesezeichen«, Seite 306):

```
id = p.begin_item("H1", "Title={Section 1} ");
b = p.create_bookmark("Section 1", "item=current");
p.fit_textline(text, x, y, "");
p.end_item(id);
```

Cookbook Codebeispiele finden Sie im Topic `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF3: Sicherstellen korrekter Tab- und Lesereihenfolge in PDF-Dokumenten. Für weitere Informationen zu diesem Thema siehe Abschnitt 10.3.6, »Druckreihenfolge und logische Lesereihenfolge«, Seite 296. Dieses Ziel lässt sich meist durch Erzeugung von Seiteninhalt in logischer Lesereihenfolge erreichen. Für Informationen zu einer davon abweichenden Reihenfolge siehe Abschnitt 10.4.4, »Erzeugung von Seiteninhalt in abweichender Reihenfolge«, Seite 308. Außerdem muss die Option *taborder=structure* bei *PDF_begin_document( )* verwendet werden.

PDF4: Verbergen dekorativer Bilder mit dem Artifact-Tag in PDF-Dokumenten. Dieses Ziel lässt sich mit der Unteroption *tagname=Artifact* der Option *tag* von *PDF_fit_image( ), PDF_fit_graphics( )* oder *PDF_fit_pdi_page( )* erreichen. Mit der gleichen Methode lassen sich dekorativer Text oder Vektorgrafik verbergen, die mit *PDF_fit_textline( ), PDF_fit_graphics( )* und anderen Funktionen erzeugt wurden:

```
p.fit_image(image, x, y, "tag={tagname=Artifact} ...");
```

Für weitere Informationen siehe Abschnitt 10.3.3, »Artefakte«, Seite 291.

Cookbook Codebeispiele für die Erstellung von Textartefakten finden Sie im Topic `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF5: Hinweis auf erforderliche Formularschaltflächen in PDF-Formularen. Dieses Ziel lässt mit der Option *required* von *PDF_create_field( )* und *PDF_create_fieldgroup( )* erreichen. Im Tooltip für das Feld sollte darauf hingewiesen werden, dass das Feld ausgefüllt werden muss:

```
optlist = "required tooltip={Name (required)} font=" + font;
p.create_field(llx, lly, urx, ury, "date", "textfield", optlist);
```

Cookbook Codebeispiele für die Erzeugung von Formularfeldern finden Sie in der Kategorie `Interactive Elements` *des PDFlib Cookbook. Codebeispiele für die Erzeugung von Formularfeldern in Tagged PDF finden Sie in der Kategorie* `pdfua`.

PDF6: Einsatz von Tabellenelementen für die Auszeichnung von Tabellen in PDF-Dokumenten. Dieses Ziel lässt sich am einfachsten durch Erzeugen von Tabellenausgabe mit *PDF_fit_table( )* und dem automatischen Erstellen von Tabellen-Tags erreichen. Alle notwendigen Tabellen-Tags und -attribute werden automatisch von PDFlib erzeugt, sofern die Optionen *headers* und *caption* von *PDF_fit_table( )* mit geeigneten Werten übergeben werden. Für weitere Informationen siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300.

Beim manuellen Anbringen von Tabellen-Tags (nicht empfohlen) müssen Sie Folgendes beachten:

- Die Tabelle muss in einem Element *Table* enthalten sein.
- Ein geeignetes Element *Caption* muss erzeugt werden.
- Ein Element *TR* muss jede Tabellenzeile umschließen.
- Die Elemente *TH* und *TD* müssen auf jede Tabellenzelle korrekt angewendet werden.
- Zellen, die mehrere Spalten oder Zeilen überspannen, müssen die passenden *RowSpan*- und/oder *ColSpan*-Attribute zugewiesen werden.
- Für Tabellenzellen ohne Inhalt müssen leere *TD*-Elemente erzeugt werden.

Cookbook Codebeispiele zur automatischen Erstellung von Tabellen-Tags und Formatierung von Tabellen finden Sie im Topic `tagged_table` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF7: Anwenden von OCR auf ein gescanntes PDF-Dokument zur Erzeugung von Ersatztext. OCR *(Optical character recognition)* wird von PDFlib nicht unterstützt. Wenn Ihnen jedoch die entsprechenden OCR-Ergebnisse für eine gescannte Seite vorliegen, können Sie den Text mit *textrendering=3* (unsichtbarer Text) ausgeben, um das gescannte Bild mit dem zugehörigen Text zu kombinieren.

Cookbook Codebeispiele finden Sie im Topic `tagged_scan_with_ocr_text` *in der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF8: Definition von Abkürzungen mit dem Eintrag E für ein Strukturelement. Dieses Ziel lässt sich mit der Option *E* von *PDF_begin_item( )* oder der Unteroption der Option *tag* von *PDF_fit_textline( )* und anderen Funktionen erreichen:

```
p.begin_item("tag={tagname=P E={January} }");
```

Wenn der Text kein eigenständiges Strukturelement ist, kann das Pseudo-Tag *ASpan* verwendet werden, um einem beliebigen Fragment eines Strukturelements die Property *E* mitzugeben:

```
p.fit_textline("Jan.", x, y, "tag={tagname=ASpan E={January} } ...");
```

Für weitere Informationen siehe Abschnitt 10.3.5, »Alternativtext, Ersatztext und Abkürzungsexpansion«, Seite 295.

PDF9: Angabe einer Überschrift mit den Heading-Tags in PDF-Dokumenten. Dieses Ziel lässt sich mit der Unteroption *tagname=H1* (und den verwandten Tags *H, H2, H3*...) der *tag*-Option von *PDF_fit_textline( )* und anderen Funktionen erreichen:

```
p.fit_textline("Introduction", x, y, "tag={tagname=H1} ...");
```

Cookbook *Codebeispiele finden Sie im Topic* `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF10: Angabe von Labels für interaktive Formularfelder in PDF-Dokumenten. Dieses Ziel lässt sich mit der Option *tooltip* von *PDF_create_field( )* oder *PDF_create_fieldgroup( )* erreichen:

```
optlist = "tooltip={Enter your name} font=" + font;
p.create_field(llx, lly, urx, ury, "date", "textfield", optlist);
```

Cookbook *Codebeispiele finden Sie im Topic* `accessible_form_fields` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF11: Angabe von Links und Linktext mit dem Strukturelement /Link in PDF-Dokumenten. Dieses Ziel lässt sich durch Angabe des Elements *Link* mit verschachteltem *OBJR* und Inhaltselementen erreichen. Für weitere Informationen siehe Abschnitt »Links und andere Arten von Anmerkungen«, Seite 303.

Cookbook *Codebeispiele finden Sie im Topic* `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF12: Angabe von Namen, Rolle und Werten für Formularfelder in PDF-Dokumenten. Dieses Ziel lässt sich mit den Parametern und Optionen von *PDF_create_field( )* und *PDF_create_fieldgroup( )* folgendermaßen erreichen:

- Die Rolle des Feldes wird im Parameter *type* definiert.
- Der Feldname wird in der Option *tooltip* angegeben.
- Der Ausgangswert wird in der Option *currentvalue* hinterlegt; der Wert nach dem Zurücksetzen des Formularfeldes wird in der Option *defaultvalue* hinterlegt.
- Der Stil von Optionsfeldern und Kontrollkästchen wird in der Option *buttonstyle* angegeben.

Cookbook *Codebeispiele finden Sie im Topic* `accessible_form_fields` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF13: Angabe von Ersatztext mit dem Eintrag /Alt für Links in PDF-Dokumenten. Dieses Ziel lässt sich durch Angabe der Option *Alt* für das Element *Link* erreichen:

```
id_link = p.begin_item("Link", "Alt={Kraxi on the Web}");
```

Für weitere Informationen siehe Abschnitt 10.3.5, »Alternativtext, Ersatztext und Abkürzungsexpansion«, Seite 295.

PDF14: Angabe laufender Kopf- und Fußzeilen in PDF-Dokumenten. Dieses Ziel lässt sich durch Platzieren von Dokumenttiteln oder Kapitelüberschriften, Seitenzahlen, Verfassernamen, Datum oder ähnlichen Informationen an einer konsistenten Stelle auf der Seite erreichen. Meist stehen solche Angaben am oberen oder unteren Rand der Seite.

Kopf- und Fußzeilen können als *pagination*-Artefakte ausgezeichnet werden:

```
optlist =   "tag={tagname=Artifact artifacttype=Pagination " +
            "artifactsubtype=Header Attached={Top Left} } ...";
p.fit_textline("Page 5", x, y, optlist);
```

Für weitere Informationen siehe Abschnitt 10.3.3, »Artefakte«, Seite 291.

PDF15: Angabe von Absenden-Schaltflächen mit der Formularaktion »Absenden« in PDF-Formularen. Dieses Ziel lässt sich durch Erzeugung eines Formularfeldes mit *type =pushbutton* von *PDF_create_field( )* erreichen:

```
submit = p.create_action("SubmitForm",
        "exportmethod=html url={http://www.kraxi.com/get.php}");

optlist =   "tooltip={Submit form} action={up=" + submit + "} " +
            "fontsize=8 font=" + font;
p.create_field(llx, lly, urx, ury, "Submit", "pushbutton", optlist);
```

Cookbook Codebeispiele finden Sie im Topic `accessible_form_fields` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF16: Angabe der Standardsprache mit dem Eintrag /Lang im Dokumentkatalog eines PDF-Dokuments. Dieses Ziel lässt sich mit der Option *lang* von *PDF_begin_document( )* erreichen:

```
p.begin_document(filename, "lang=en");
```

Die Sprache kann auch für einzelne Strukturelemente festgelegt werden; siehe »PDF19: Angabe der Sprache für einen Absatz oder Satz mit dem Eintrag Lang in PDF-Dokumenten«, Seite 319.

Cookbook Codebeispiele finden Sie im Topic `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF17: Festlegen konsistenter Seitennummerierung für PDF-Dokumente. Dieses Ziel lässt sich mit der Option *label* von *PDF_begin/end_page_ext( )* zur Angabe logischer Seitenzahlen oder -namen erreichen, die der Seitennummerierung auf jeder Seite entsprechen. Dies ist notwendig, wenn die Seitenzahl auf der Dokumentseite von der PDF-Seitenzahl abweicht. Beispielsweise werden für die Seitenzahlen im Inhaltsverzeichnis oft römische Ziffern verwendet:

```
p.begin_page_ext(595, 842, "label={style=R}");
```

Dies muss auf jeder ersten Seite eines Abschnitts *(section)* im Dokument durchgeführt werden, auf der sich das Format der Seitenzahlen ändert. Beispielsweise könnten die Seiten des Haupttextes mit dezimalen Ziffern nummeriert sein, beginnend bei 10:

```
p.begin_page_ext(595, 842, "label={style=D start=10}");
```

Für weitere Informationen zur Option *label* siehe die PDFlib-Referenz.

PDF18: Angabe des Dokumenttitels mit dem Eintrag Title im Document Information Dictionary eines PDF-Dokuments. Dieses Ziel lässt sich durch Angabe von *Title* bei *PDF_set_info( )* oder der entsprechenden XMP-Property *dc:title* in der Option *metadata* von *PDF_begin/end_document( )* erreichen:

```
p.set_info("Title", "Phone bill for May 2013");
```

Damit der Dokumenttitel und nicht der Dateiname in der Titelleiste angezeigt wird, verwenden Sie die Unteroption *displaydoctitle* der Option *viewerpreferences* von *PDF_begin/end_document( ):*

```
p.begin_document(filename, "viewerpreferences={displaydoctitle}");
```

Cookbook Codebeispiele finden Sie im Topic `starter_pdfua1` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF19: Angabe der Sprache für einen Absatz oder Satz mit dem Eintrag Lang in PDF-Dokumenten. Dieses Ziel lässt sich mit der Option *lang* von *PDF_begin_item( )* oder der Unteroption der Option *tag* von *PDF_fit_textline( )* und anderen Funktionen erreichen:

```
p.begin_item("tag={tagname=P lang=de }");
```

Wenn der Text kein eigenständiges Strukturelement ist, kann das Pseudo-Tag *ASpan* verwendet werden, um einem beliebigen Fragment eines Strukturelements die Property *lang* mitzugeben:

```
p.fit_textline("Widerrufsrecht", x, y, "tag={tagname=ASpan Lang=de} ...");
```

Für weitere Informationen siehe Abschnitt »Angabe der Sprache«, Seite 293.

PDF20: Einsatz des Tabellen-Editors von Adobe Acrobat Pro zur Korrektur von Tabellen-Tags. Wenn das PDFlib-Feature Automatisches Anbringen von Tabellen-Tags verwendet wird, werden alle Tabellen mit korrekten Tags versehen, siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300. Bei komplexen Tabellen-Layouts können zusätzliche Tabellenattribute erforderlich sein.

Beim manuellen Anbringen von Tabellen-Tags und -attributen liegt die Verantwortung für das korrekte Anbringen bei den Benutzern.

Cookbook Codebeispiele zur automatischen Erstellung von Tabellen-Tags und Formatierung von Tabellen finden Sie im Topic `tagged_table` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF21: Verwendung von List-Tags für Listen in PDF-Dokumenten. Dieses Ziel lässt sich durch korrektes Auszeichnen von Listen mit den Elementen *L*, *Caption*, *LI*, *Lbl* und *LBody* erreichen, siehe Abschnitt 10.4.3, »Listen«, Seite 307.

Cookbook Codebeispiele finden Sie im Topic `tagged_list` *der Kategorie* `pdfua` *des PDFlib Cookbook.*

PDF22: Hinweis auf Benutzereingabe, die dem erforderlichen Format oder den Werten in PDF-Formularen nicht entspricht. Dieses Ziel lässt sich mit JavaScript-Code erreichen, der zur Validierung der Benutzereingabe an die Formularfelder angehängt wird:

```
optlist = "script={ ... }";
validate_action = p.create_action("JavaScript", optlist);
textfield_font = p.load_font("Helvetica", "pdfdoc", "nosubsetting");

optlist = "action={validate=" + validate_action + "} " +
          "backgroundcolor={gray 0.8} font=" + textfield_font +
          "tag={tagname=Form} tooltip={Starting date}";
p.create_field(llx, lly, urx, ury, "startdate", "textfield", optlist);
```

Für oben nicht dargestellte JavaScript-Codebeispiele siehe Abschnitt »Validieren der Formularfeld-Eingabe«, Seite 273.

Cookbook *Codebeispiele für die Erzeugung von Formularfeldern finden Sie in der Kategorie* `Interactive Elements` *des PDFlib Cookbook. Codebeispiele für die Erzeugung von Formularfeldern in Tagged PDF finden Sie in der Kategorie* `pdfua`.

PDF23: Bereitstellen interaktiver Formularfelder in PDF-Dokumenten. Dieses Ziel kann durch Erzeugung der erforderlichen Formularfelder mit *PDF_create_field( )* und der Option *tooltip* erreicht werden. Unter PDF22 finden Sie ein Codebeispiel hierzu; andere Arten von Formularfeldern lassen sich auf ähnliche Weise erzeugen.

Cookbook *Codebeispiele für die Erzeugung von Formularfeldern finden Sie in der Kategorie* `Interactive Elements` *des PDFlib Cookbook. Codebeispiele für die Erzeugung von Formularfeldern in Tagged PDF finden Sie in der Kategorie* `pdfua`.

11 PDF-Versionen und PDF-Standards

11.1 Acrobat und PDF-Versionen

PDFlib generiert nach Wahl des Anwenders Ausgabe, die zu folgenden PDF-Versionen kompatibel ist:

- PDF 1.4 (Acrobat 5)
- PDF 1.5 (Acrobat 6)
- PDF 1.6 (Acrobat 7)
- PDF 1.7 (Acrobat 8), technisch identisch mit ISO 32000-1
- PDF 1.7 Adobe extension level 3 (Acrobat 9)
- PDF 1.7 Adobe extension level 8 (Acrobat X und XI)
- PDF 2.0 gemäß ISO 32000-2

Die Version der PDF-Ausgabe lässt sich mit der Option *compatibility* in *PDF_begin_document( )* steuern. Der jeweilige PDF-Kompatibilitätsmodus lässt keine PDFlib-Funktionen einer höheren Stufe zu (siehe Tabelle 11.1). Der Versuch, solche Funktionen zu verwenden, löst eine Exception aus.

Import verschiedener PDF-Versionen mit PDI. In allen Kompatibilitätsmodi können mit PDI nur PDF-Dokumente mit derselben oder einer niedrigeren Kompatibilitätsstufe importiert werden. Soll ein PDF mit einer aktuelleren Kompatibilitätsstufe importiert werden, müssen Sie die Option *compatibility* entsprechend setzen (siehe Abschnitt 7.3.3, »Dokument- und seitenbezogene Prüfungen«, Seite 204). Als Ausnahme zu dieser Regel lassen sich Dokumente gemäß PDF 1.7 extension level 3 (Acrobat 9) und PDF 1.7 extension level 8 (Acrobat X/XI) auch in Dokumente mit PDF-Version 1.7 importieren.

Ändern der PDF-Version eines Dokuments. Wenn Sie Ausgabe in einer bestimmten PDF-Version erstellen und dazu PDF in einer höheren Version importieren möchten, müssen Sie die Dokumente vor dem PDI-Import in die niedrigere PDF-Version konvertieren. Sie können in Acrobat mit den Menübefehlen *Datei, Speichern unter..., Andere, Optimiertes PDF* (Acrobat XI) oder *Datei, Speichern unter..., Optimiertes PDF* (Acrobat X) oder *Erweitert, PDF Optimierung, Kompatibel mit* (Acrobat 9) die PDF-Version folgendermaßen ändern:

- Acrobat 9: PDF 1.3 - PDF 1.7 extension level 3
- Acrobat X und XI: PDF 1.3 - PDF 1.7 extension level 8

Tabelle 11.1 PDFlib-Funktionen, die eine bestimmte PDF-Kompatibilitätsstufe benötigen

Funktion	*PDFlib-API-Funktionen und -Optionen*
Funktionen, die PDF 2.0 = ISO 32000-2 oder einen bestimmten PDF-Standard benötigen	
Document Part Hierarchy	*PDF_begin/end_dpart( ): Eine Document Part Hierarchy erfordert PDF 2.0 oder PDF/VT.*
Beziehung von Dateianhängen	*Option* `relationship` *für PDF_load_asset( ) mit* `type=Attachment`*, für PDF_add_portfolio_file( ) und als Unteroption für die Option* `attachments` *von PDF_begin/end_document( ) und die Option* `attachment` *von PDF_create_annotation( ): Das Festlegen von Beziehungen erfordert PDF 2.0 oder PDF/A-3.*

Tabelle 11.1 PDFlib-Funktionen, die eine bestimmte PDF-Kompatibilitätsstufe benötigen

Funktion	***PDFlib-API-Funktionen und -Optionen***
Dateiassoziation	*Option* `associatedfiles` *für PDF_end_document( ), PDF_begin/end_page_ext( ), PDF_begin/end_dpart( ), PDF_begin_template_ext( ), PDF_load_image( ), PDF_open_pdi_page( ), PDF_load_graphics( ): Dateiassoziationen erfordern PDF 2.0 oder PDF/A-3.*
Funktionen, die PDF 1.7 extension level 8 (Acrobat X/XI) oder höher benötigen	
AES-Verschlüsselung mit 256-Bit-Schlüsseln	*PDF_begin_document( ): AES-Verschlüsselung mit 256-Bit-Schlüsseln und ein stärkerer Verschlüsselungsalgorithmus als in extension level 3 wird automatisch mit* `compatibility=1.7ext8` *verwendet, wenn die Option* `masterpassword`*,* `userpassword`*,* `attachmentpassword` *oder* `permissions` *übergeben wird*
direkte Verwendung von Ebenen für PDF/X-4:2010 (ohne Ebenen-Varianten)	*PDF_set_layer_dependency( ): Option* `createorderlist`
Funktionen, die PDF 1.7 extension level 3 (Acrobat 9) oder höher benötigen	
Multimedia (Flash)	*PDF_load_asset( )* *PDF_create_annotation( ): Option* `type=RichMedia` *PDF_create_action( ): Option* `type=RichMediaExecute` *PDF_begin_document( ): Option* `portfolio`*, Unteroptionen* `navigator` *und* `initialview=custom`
PDF mit Geodaten	*PDF_begin_document( ): Option* `viewports` *PDF_load_image( ): Option* `georeference`
PDF-Portfolios mit Dateiordnern	*PDF_add_portfolio_folder( )*
AES-Verschlüsselung mit 256-Bit-Schlüsseln	*PDF_begin_document( ): AES-Verschlüsselung mit 256-Bit-Schlüsseln gemäß PDF 1.7 wird automatisch mit* `compatibility=1.7ext3` *verwendet, wenn die Option* `masterpassword`*,* `userpassword`*,* `attachmentpassword` *oder* `permissions` *übergeben wird, um die Schwäche im AES-256-Verschlüsselungsalgorithmus gemäß PDF 1.7ext3 zu vermeiden*
Ebenen-Varianten	*PDF_set_layer_dependency( ): Abhängigkeitstyp* `Variant` *(Diese Funktion benötigt kein PDF 1.7ext3, funktioniert aber nur in Acrobat 9)*
Referenziertes PDF	*Option* `reference` *in PDF_open_pdi_page( ) und PDF_begin_template_ext( )* *(Diese Funktion benötigt kein PDF 1.7ext3, funktioniert aber nur ab Acrobat 9)*
Einbettung von 3D-Modellen im PRC-Format	*PDF_load_3ddata( ): Option* `type=PRC`
Barcode-Felder	*PDF_create_field( ) und PDF_create_fieldgroup( ): Option* `barcode`
Funktionen, die PDF 1.7 = ISO 32000-1 (Acrobat 8) oder höher benötigen	
PDF-Portfolios	*PDF_begin_document( ): Option* `portfolio` *PDF_add_portfolio_file( )*
Unicode-Dateinamen für Anhänge	*PDF_begin/end_document( ): Option* `attachments`*, Unteroption* `filename`
Funktionen, die PDF 1.6 (Acrobat 7) oder höher benötigen	
Benutzereinheiten	*PDF_begin/end_document( ): Option* `userunit`
Skalierung beim Drucken	*PDF_begin/end_document( ): Unteroption* `printscaling` *für Option* `viewerpreferences`
Modus zum Öffnen des Dokuments	*PDF_begin/end_document( ): Option* `openmode=attachments`

Tabelle 11.1 PDFlib-Funktionen, die eine bestimmte PDF-Kompatibilitätsstufe benötigen

Funktion	***PDFlib-API-Funktionen und -Optionen***
AES-Verschlüsselung mit 256-Bit-Schlüsseln	*PDF_begin_document( ): AES-Verschlüsselung wird automatisch mit* `compatibility=1.6` *oder* `1.7` *verwendet, wenn die Option* `masterpassword`, `userpassword`, `attachmentpassword` *oder* `permissions` *übergeben wird*
Nur Dateianlagen verschlüsseln	*PDF_begin/end_document( ): Option* `attachmentpassword`
Beschreibung von Dateianlagen	*PDF_begin/end_document( ): Unteroption* `description` *für Option* `attachments`
Einbettung von 3D-Modellen im U3D-Format	*PDF_load_3ddata( ), PDF_create_3dview( )* *PDF_create_annotation( ): Option* `type=3D` *PDF_create_action( ): Option* `type=GoTo3DView`
Funktionen, die PDF 1.5 (Acrobat 6) oder höher benötigen	
verschiedene Feldoptionen	*PDF_create_field( ) und PDF_create_fieldgroup( )*
Seitenlayout	*PDF_begin/end_document( ): Option* `pagelayout=twopageleft/right`
verschiedene Optionen für Anmerkungen	*PDF_create_annotation( )*
erweiterte Berechtigungen	`permissions=plainmetadata` *in PDF_begin_document( ), siehe Tabelle 3.4*
einige CMaps für CJK-Fonts	*PDF_load_font( ), siehe Tabelle 4.5*
Tagged PDF	*verschiedene Optionen für PDF_begin_item( );* *PDF_begin/end_page_ext( ): Option* `taborder`
Ebenen	*PDF_define_layer( ), PDF_begin_layer( ), PDF_end_layer( ), PDF_layer_dependency( )*
JPEG-2000-Rasterbilder	`imagetype=jpeg2000` *in PDF_load_image( )*
komprimierte Object-Streams	*Komprimierte Object-Streams werden automatisch erzeugt mit* `compatibility=1.5` *oder höher, sofern nicht* `objectstreams=none` *in PDF_begin_document( ) gesetzt wurde.*

11.2 Der PDF-Standard ISO 32000

ISO 32000-1. PDF 1.7 wurde als ISO 32000-1 standardisiert. Der technische Inhalt dieses internationalen Standards entspricht der PDF 1.7-Referenz von Adobe, dem Dateiformat von Acrobat 8. Mit PDFlib erzeugte PDF-Dokumente sind konform zu ISO 32000-1. Allerdings gibt es ein paar veraltete PDF-Funktionen, die nicht Teil von ISO 32000-1, aber in PDFlib mit *compatibility=1.7* oder darunter verfügbar sind. Diese Funktionen wurden aufgrund von Anforderungen aus der Praxis aufgenommen. Wenn die Konformität zu ISO 32000-1 erforderlich ist, dürfen die Funktionen in Tabelle 11.2 nicht verwendet werden.

Tabelle 11.2 PDF 1.7-Funktionen, die nicht Bestandteil von ISO 32000-1 sind

Funktion	PDFlib-API-Funktionen und -Optionen
Document Part Hierarchy	*PDF_begin/end_dpart( ): Eine Document Part Hierarchy erfordert PDF 2.0 oder PDF/VT.*
Dateiassoziation	*Option* `associatedfiles` *für PDF_end_document( ), PDF_begin/end_page_ext( ), PDF_begin/end_dpart( ), PDF_begin_template_ext( ), PDF_load_image( ), PDF_open_pdi_page( ), PDF_load_graphics( ): Dateiassoziationen erfordern PDF 2.0 oder PDF/A-3.*
angehängter Suchindex	*PDF_begin_document( ): Option* `search`

ISO 32000-2. Zum Zeitpunkt der Erstellung des vorliegenden Dokuments ist die nächste Version des ISO-Standards, ISO 32000-2, in Vorbereitung. Dieser Standard spezifiziert PDF 2.0 und umfasst Funktionen aus folgenden Gruppen:

- Acrobat-9-Funktionen, die von PDFlib mit der Dokumentoption *compatibility=pdf1.7ext3* unterstützt werden, z.B. PDF mit Geodaten, hierarchische Portfolios und AES-256-Verschlüsselung; für weitere Informationen siehe Tabelle 11.1.
- Acrobat-X-Funktionen, die von PDFlib mit *compatibility=pdf1.7ext8* unterstützt werden, vor allem AES-256-Verschlüsselung mit einem stärkeren Verschlüsselungsalgorithmus.
- Mit den PDF/A-3- und PDF/VT-Standards eingeführte Funktionen, die kein Bestandteil von ISO 32000-1 sind.

Wenn die Konformität zu ISO 32000-2 erforderlich ist, darf die Funktion in Tabelle 11.3 nicht verwendet werden.

Tabelle 11.3 PDF-Funktionen, die nicht mehr Bestandteil von ISO 32000-2 sind

Funktion	PDFlib-API-Funktionen und -Optionen
PostScript XObjects	*PDF_begin_template_ext( ): Option* `postscript`

11.3 PDF/A zur Archivierung

11.3.1 Die PDF/A-Standards

Das im Standard ISO 19005 definierte PDF/A-Format wurde entwickelt, um eine konsistente und stabile Teilmenge von PDF zu definieren, die sich langfristig archivieren lässt und zum zuverlässigen Datenaustausch in Unternehmen und Behörden verwendet werden kann.

PDF/A Competence Center in der PDF Association. PDFlib GmbH ist Gründungsmitglied der PDF Association, die das PDF/A Competence Center als eine ihrer Aktivitäten verantwortet. Ziel des Verbands ist die Förderung des Informations- und Erfahrungsaustauschs auf dem Gebiet der Langzeitarchivierung gemäß ISO 19005. Der Verband führt Veranstaltungen und Seminare durch und steht als kompetente Anlaufstelle und Austauschplattform rund um PDF/A zur Verfügung. Weitere Informationen finden Sie auf der Webseite der PDF Association unter *www.pdfa.org*.

PDF/A-1a:2005 und PDF/A-1b:2005 gemäß ISO 19005-1. PDF/A-1 basiert auf PDF 1.4 und legt Einschränkungen für die Verwendung von Farben, Fonts, Anmerkungen und andere Elementen fest. Von PDF/A-1 gibt es zwei Varianten:

- Konformität zu ISO 19005-1 Level B (PDF/A-1b) gewährleistet, dass das Erscheinungsbild eines Dokuments langfristig erhalten werden kann. Einfach ausgedrückt bedeutet das, dass das Dokument genauso aussieht, wenn es in Zukunft verarbeitet wird.
- Konformität zu ISO 19005-1 Level A (PDF/A-1a) basiert auf Level B, fordert aber zusätzlich einige aus der Variante »Tagged PDF« bekannte Eigenschaften. Gefordert sind Strukturinformationen und zuverlässige Textinterpretation, um die logische Dokumentstruktur und natürliche Lesereihenfolge zu erhalten. PDF/A-1a gewährleistet nicht nur, dass Dokumente bei späterer Verarbeitung unverändert aussehen, sondern auch, dass ihr Inhalt (Semantik) zuverlässig interpretierbar und auch für körperlich beeinträchtigte Benutzer zugänglich ist.

PDFlib unterstützt PDF/A-1 auf Basis der folgenden Dokumente:

- PDF/A-1-Standard (ISO 19005-1:2005)
- Technical Corrigendum 1 (ISO 19005-1:2005/Cor.1:2007)
- Technical Corrigendum 2 (ISO 19005-1:2005/Cor.2:2011)
- Alle relevanten, vom PDF/A Competence Center veröffentlichten TechNotes

Wenn im Folgenden von PDF/A-1 (ohne Konformitätsstufe) gesprochen wird, so sind beide Konformitätsstufen PDF/A-1a und PDF/A-1b gemeint.

PDF/A-2a, PDF/A-2b und PDF/A-2u gemäß ISO 19005-2. Die Varianten des PDF/A-2-Standards basieren auf ISO 32000-1 (das heißt PDF 1.7); sie unterstützen also mehr Funktionen als PDF/A-1. Anders als PDF/A-1 erlaubt der neuere PDF/A-2-Standard Transparenz, Ebenen, JPEG-2000-Kompression, PDF/A-Dateianhänge, PDF-Pakete und andere PDF-Funktionen. PDF/A-2 definiert die folgenden Varianten:

- Konformität zu ISO 19005-2 Level B (PDF/A-2b) gewährleistet das Erscheinungsbild eines Dokuments.

- Konformität zu ISO 19005-2 Level A (PDF/A-2a) umfasst zusätzlich zuverlässige Unicode-Textinterpretation und Tagged PDF mit Strukturinformationen. Die Tags gewährleisten die Barrierefreiheit von PDF/A-2a-Dokumenten.
- Konformität zu ISO 19005-2 Level U (PDF/A-2u) liegt zwischen PDF/A-2a und PDF/A-2b, weil zwar zuverlässige Unicode-Textinterpretation erforderlich ist, aber keine Strukturinformationen. PDF/A-2u gewährleistet, dass Dokumente bei späterer Verarbeitung unverändert aussehen und dass der Text extrahiert und durchsucht werden kann.

PDFlib unterstützt PDF/A-2 auf Basis des folgenden Dokuments:
- PDF/A-2-Standard (ISO 19005-2:2011)

Wenn im Folgenden von PDF/A-2 (ohne Konformitätsstufe) gesprochen wird, so sind die Konformitätsstufen PDF/A-2a, PDF/A-2b und PDF/A-2u gemeint.

PDF/A-3a, PDF/A-3b und PDF/A-3u, definiert in ISO 19005-3. PDF/A-3 ähnelt PDF/A-2, mit den folgenden Unterschieden:
- Während bei PDF/A-2 nur PDF/A-1- oder PDF/A-2-konforme Dateianhänge erlaubt sind, erlaubt PDF/A-3 beliebige Arten von Dateianhängen.
- Dateianhänge beziehen sich auf das gesamte Dokument, eine Seite oder ein anderes Element des Dokuments. Die Beziehung zwischen Dateianhang und dem jeweiligen Teil des Dokuments muss explizit angegeben werden, zum Beispiel das Quelldokument, alternative oder ergänzende Daten.

PDFlib unterstützt PDF/A-3 auf Basis des folgenden Dokuments:
- PDF/A-3-Standard (ISO 19005-3:2012)

Wenn im Folgenden von PDF/A-3 (ohne Konformitätsstufe) gesprochen wird, so sind die Konformitätsstufen PDF/A-3a, PDF/A-3b und PDF/A-3u gemeint.

11.3.2 Allgemeine Anforderungen

Cookbook *Codebeispiele für die Erzeugung von PDF/A finden Sie in der Kategorie* `pdfa` *des PDFlib Cookbook.*

Wenn sich der PDFlib-Client an die in diesem Kapitel dokumentierten Regeln hält, ist standardkonforme PDF/A-Ausgabe gewährleistet. Entdeckt PDFlib eine Regelverletzung bei der PDF/A-Erstellung, so wird eine Exception ausgelöst, die von der Anwendung abgefangen werden muss. In diesem Fall wird keine PDF-Ausgabe erzeugt.

Tabelle 11.4 zeigt die allgemeinen Anforderungen, die zur Erzeugung von PDF/A-konformer Ausgabe erfüllt sein müssen.

Kombinierte PDF/A- und PDF/UA-Dokumente. Ein PDF/A-Dokument kann gleichzeitig konform zu PDF/UA-1 sein. Tatsächlich empfehlen wir, wenn Sie PDF/A-1a/2a/3a erstellen möchten, die PDF/UA-Anforderungen zu erfüllen, um die Zugänglichkeit der erstellten Dokumente zu verbessern. Für weitere Informationen siehe Abschnitt »Erzeugung kombinierter PDF/UA- und PDF/A-Dokumente«, Seite 356.

Kombinierte PDF/A- und PDF/X-Dokumente. Ein PDF/A-Dokument kann gleichzeitig konform zu PDF/X-1a:2003, PDF/X-3:2003 oder PDF/X-4 sein (aber nicht zu PDF/X-4p oder PDF/X-5). Um ein Dokument mit diesen beiden Eigenschaften zu erzeugen, müs-

Tabelle 11.4 Allgemeine Anforderungen für die Konformität zu PDF/A Level A, B und U

Kriterium	Für PDF/A-Konformität erforderliche PDFlib-Funktionen und -Optionen (alle Konformitätsstufen)
PDF/A-Konformitätsstufe und PDF-Kompatibilität	*PDF_begin_document( ): Die Option* `pdfa` *muss auf die erforderliche PDF/A-Konformitätsstufe gesetzt werden, z.B.* `pdfa=PDF/A-2b`*.* *(PDF/A-1) Operationen, die PDF 1.5 oder höher benötigen, sind nicht zulässig (z.B. Ebenen).* *(PDF/A-2/3) Operationen, die PDF 1.7ext3 oder höher benötigen, sind nicht zulässig (z.B. PDF-Portfolios).*
Fonts	*Die Fontoption* `embedding` *muss auf* `true` *gesetzt sein. Die Optionen* `unicodemap=false` *und* `dropcorewidths=true` *sind nicht erlaubt.* *Auch die PDF-Standardfonts müssen eingebettet werden. Die einzige Ausnahme hierzu besteht für Fonts, die ausschließlich für unsichtbaren Text verwendet werden (sinnvoll bei OCR-Ergebnissen). Dies kann mit der Option* `optimizeinvisible` *gesteuert werden.*
Textausgabe	*(PDF/A-2/3) Die Fontoption* `replacementchar` *wird auf den Wert* `error` *gezwungen, um Glyphen vom Typ* `.notdef` *in der Ausgabe zu vermeiden.*
Seitengrößen	*PDF_begin/end_page_ext( ): Bei PDF/A gibt es keine strikte Beschränkung der Seitengröße. Die empfohlene Seitengröße (Breite und Höhe sowie alle Boxeinträge) liegt jedoch bei PDF/A-1 zwischen 3 und 14400 Punkt (508 cm) und bei PDF/A-2/3 zwischen 3 und 14400 Benutzereinheiten.*
Ebenen	*PDF/A-1: PDF_define_layer( ) und PDF_set_layer_dependency( ) sind nicht zulässig.* *PDF/A-2/3: Ebenen dürfen verwendet werden, aber einige Optionen von PDF_define_layer( ) und PDF_set_layer_dependency( ) sind unzulässig.*
Sicherheit	*In PDF_begin_document( ) sind die Optionen* `userpassword`*,* `masterpassword` *und* `permissions` *unzulässig.*
Externer Inhalt	*PDF_begin_template_ext( ), PDF_load_graphics( ) und PDF_open_pdi_page( ): Die Option* `reference` *ist nicht zulässig.* *PDF_load_asset( ): Die Option* `external` *ist nicht zulässig.*
Dateigröße	*Die Dateigröße des generierten PDF-Dokuments darf 2 GB nicht überschreiten, und die Anzahl der PDF-Objekte muss kleiner als 8 388 607 sein. Für weitere Informationen siehe Abschnitt 3.1.5, »Maximalgröße von PDF-Dokumenten und andere Grenzwerte«, Seite 71.*
PDF-Import	*PDF_open_pdi_document( ) ist beschränkt, außer wenn* `infomode=true`*; siehe Abschnitt 11.3.7, »Import von PDF/A-Dokumenten mit PDI«, Seite 332.*

sen Sie in den Optionen *pdfa* und *pdfx* von *PDF_begin_document( )* geeignete Werte übergeben, zum Beispiel:

```
ret = p.begin_document("combo.pdf", "pdfa=PDF/A-2b pdfx=PDF/X-4");
```

11.3.3 Anforderungen für Farbe und Rasterbilder

PDF/A garantiert die farbgetreue Wiedergabe durch die erforderlichen geräteunabhängigen Farbspezifikationen. Farbräume können aus folgenden Quellen stammen:

- Bilder, die direkt mit *PDF_load_image( )* und *PDF_fill_imageblock ( )* geladen werden und indirekt mit *PDF_load_graphics( )*
- Explizite Farbfestlegungen mit *PDF_set_graphics_option( )* oder *PDF_setcolor( )*
- Farbspezifikationen über Optionslisten, zum Beispiel in Textflows
- Mischfarbräume für Transparenzgruppen: *PDF_begin/end_page_ext( )*, *PDF_begin_template_ext( )* und *PDF_open_pdi_page( ):* Option *transparencygroup* mit Unteroption *colorspace*
- Bei interaktiven Elementen kann eine Rahmenfarbe definiert werden

Tabelle 11.5 zeigt die PDF/A-Anforderungen für die Farbverarbeitung, die in allen oben aufgeführten Operationen erfüllt werden müssen.

Tabelle 11.5 Anforderungen für Farbe und Rasterbilder für die Konformität zu PDF/A Level A, B und U

Kriterium	Für PDF/A-Konformität erforderliche PDFlib-Funktionen und -Optionen (alle Konformitätsstufen)
Druckausgabe-bedingung (output intent)	*Unmittelbar nach PDF_begin_document( ) muss entweder PDF_load_iccprofile( ) mit* `usage=outputintent` *oder PDF_process_pdi( ) mit* `action=copyoutputintent` *aufgerufen werden, sofern im Dokument einer der geräteabhängigen Farbräume für Graustufen, RGB oder CMYK verwendet wird und kein geeigneter Default-Farbraum für die Seite eingestellt wurde.*
Graustufen	*Graustufen-Farben können nur verwendet werden, wenn eine Ausgabebedingung für Graustufen, RGB oder CMYK vorhanden ist oder die Option* `defaultgray` *von PDF_begin_page_ext( ) gesetzt wurde.*
RGB-Farbe	*RGB-Farbe kann nur verwendet werden, wenn eine Ausgabebedingung für RGB vorhanden ist oder die Option* `defaultrgb` *von PDF_begin_page_ext( ) gesetzt wurde.*
CMYK-Farbe	*CMYK-Farbe kann nur verwendet werden, wenn eine Ausgabebedingung für CMYK vorhanden ist oder die Option* `defaultcmyk` *von PDF_begin_page_ext( ) gesetzt wurde.*
Transparenz und Überdrucken	*(PDF/A-1) Transparenz ist unzulässig; dies betrifft die folgenden API-Funktionen:* ► *PDF_load_image( ): Die Option* `masked` *ist unzulässig, sofern sich die Transparenzmaske nicht auf ein 1-Bit-Bild bezieht.* ► *PDF_load_image( ): Bilder mit impliziter Transparenz (Alphakanal) sind unzulässig; sie müssen mit der Option* `ignoremask` *geladen werden.* ► *PDF_load_graphics( ): SVG-Grafik mit Transparenzelementen ist unzulässig.* ► *PDF_create_gstate( ): Die Optionen* `opacityfill` *und* `opacitystroke` *sind unzulässig, sofern sie nicht den Wert 1 haben;* `blendmode` *darf nur mit dem Wert* `Normal` *verwendet werden.* ► *PDF_create_annotation( ): Die Option* `opacity` *ist unzulässig.* *(PDF/A-2/3) Transparenz ist erlaubt, aber die folgende Regel muss eingehalten werden für PDF_create_gstate( ):* `overprintmode=1` *ist nicht zulässig, wenn der aktuelle Farbraum ein ICC-basierter CMYK-Farbraum ist und* `overprintfill` *oder* `overprintstroke` *auf* `true` *gesetzt ist.*
Transparenzgruppen	*PDF_begin/end_page_ext( ), PDF_begin_template_ext( ), PDF_open_pdi_page( ) und PDF_load_graphics( ): Die Option* `transparencygroup` *ist folgendermaßen eingeschränkt:* *(PDF/A-1) Die Option* `transparencygroup` *ist unzulässig.* *(PDF/A-2/3) Die Unteroption* `colorspace` *der Option* `transparencygroup` *muss die obigen Anforderungen für Graustufen, RGB- und CMYK-Farbe erfüllen.*
Rasterbilder und Templates	*PDF_load_image( ): Die Optionen* `OPI-1.3`, `OPI-2.0` *und* `interpolate=true` *sind unzulässig.* *PDF_begin_template_ext( ): Die Optionen* `OPI-1.3`, `OPI-2.0` *und* `postscript` *sind unzulässig.* *(PDF/A-2/3) JPEG-2000-Bilder müssen bestimmte Bedingungen erfüllen, siehe Abschnitt »JPEG-2000-Bilder«, Seite 186.*

Druckausgabebedingungen. Die Druckausgabebedingung definiert das beabsichtigte Zielgerät, was für die originalgetreue Farbdarstellung entscheidend ist. Bei PDF/X ist eine Druckausgabebedingung zwingend erforderlich. PDF/A dagegen erlaubt die Angabe einer ICC-basierten Ausgabebedingung, setzt sie aber nicht zwingend voraus. Eine Druckausgabebedingung ist nur erforderlich, wenn im Dokument geräteabhängige Farben wie RGB verwendet werden. Wenn nur geräteunabhängige Farben wie zum Beispiel ICC-basierte Farben verwendet werden, ist keine Druckausgabebedingung erforderlich. Während PDF/X nur ICC-Profile für Drucker als Ausgabebedingung erlaubt, sind bei PDF/A auch Monitorprofile zulässig. Damit lässt sich das weit verbreitete sRGB-Profil als

Ausgabebedingung verwenden. Die Ausgabebedingung kann mit einem ICC-Profil folgendermaßen festlegt werden:

```
icc = p.load_iccprofile("sRGB", "usage=outputintent");
```

Statt ein ICC-Profil zu laden, kann die Druckausgabebedingung auch aus einem importierten PDF/A-Dokument kopiert werden; siehe Abschnitt »Kopieren der PDF/A-Druckausgabebedingung eines importierten Dokuments«, Seite 334. Die Druckausgabebedingung des generierten Dokuments muss genau einmal gesetzt werden, und zwar am besten direkt nach dem Aufruf von *PDF_begin_document( )*.

Strategien zum Einsatz von Farbe in PDF/A. Tabelle 11.6 gibt eine Übersicht, die bei der Planung von PDF/A-Anwendungen hilfreich sein kann. Der einfachste und in vielen Situationen angemessene Ansatz besteht in der Verwendung des *sRGB*-Profils als Druckausgabebedingung, da dieses Graustufen und RGB-Farben unterstützt. Außerdem ist *sRGB* bereits in PDFlib eingebaut und benötigt daher keine externen Profildaten oder -konfiguration.

Um schwarzen Text ohne ein Profil für die Druckausgabebedingung auszugeben, kann der Farbraum CIELab verwendet werden. Der *Lab*-Farbwert *(0, 0, 0)* definiert reines Schwarz in geräteunabhängiger Art und ist (im Gegensatz zu DeviceGray) auch ohne Druckausgabebedingung konform zu PDF/A. PDFlib stellt die aktuelle Farbe am Anfang jeder Seite automatisch auf schwarz. Abhängig davon, ob eine ICC-Druckausgabebedingung definiert wurde oder nicht, wird dazu der Farbraum DeviceGray oder Lab verwendet. Um die *Lab*-Farbe schwarz manuell einzustellen, verwenden Sie folgenden Aufruf:

```
p.set_graphics_option("fillcolor={lab 0 0 0}");
```

Tabelle 11.6 Strategien zum Einsatz von Farbe für die Konformität zu PDF/A Level A, B und U

	Farbräume, die im Dokument verwendbar sind					
Druckausgabebedingung	**CIELab**	**ICCBased**	**Pantone, HKS**	**Grayscale[1]**	**RGB[1]**	**CMYK[1]**
keine	ja	ja	ja	–	–	–
Graustufen-Profil	ja	ja	ja	ja	–	–
RGB-Profil, z.B. sRGB	ja	ja	ja	ja	ja	–
CMYK-Profil	ja	ja	ja	ja	–	ja

1. Geräteunabhängiger Farbraum ohne ICC-Profil

Zusätzlich zu den in Tabelle 11.6 aufgeführten Farbräumen können benutzerdefinierte Schmuckfarben mit entsprechendem alternativen Farbraum verwendet werden. Da PDFlib CIELab als alternativen Farbraum für die integrierten HKS- und Pantone-Schmuckfarben verwendet, können diese bei PDF/A immer verwendet werden. Bei benutzerdefinierten Farben muss der alternative Farbraum so gewählt werden, dass er zur PDF/A-Druckausgabebedingung konform ist.

11.3.4 Anforderungen für interaktive Funktionen

Tabelle 11.7 führt alle Operationen auf, die bei der Erzeugung PDF/A-konformer Ausgabe eingeschränkt sind. Der Aufruf einer im PDF/A-Modus unzulässigen Funktion löst eine Exception aus.

Tabelle 11.7 Anforderungen für interaktive Funktionen bei allen PDF/A-Konformitätsstufen

Kriterium	Für PDF/A-Konformität erforderliche PDFlib-Funktionen und -Optionen (alle Konformitätsstufen)
Anmerkungen	*(PDF/A-1) Für PDF_create_annotation( ) gelten folgenden Einschränkungen:* ► *Anmerkungen mit* `type=FileAttachment` *und* `Movie` *sind nicht zulässig.* ► *Die Optionen* `zoom` *und* `rotate` *für Textanmerkungen dürfen nicht auf* `true` *gesetzt werden.* ► *Die Optionen* `annotcolor` *und* `interiorcolor` *dürfen nur verwendet werden, wenn eine Druckausgabebedingung für RGB festgelegt wurde. Die Option* `fillcolor` *darf nur benutzt werden, wenn eine Druckausgabebedingung für RGB oder CMYK definiert wurde. Außerdem muss ein entsprechender RGB- oder CMYK-Farbraum verwendet werden.* ► *Die Option* `opacity` *ist nicht zulässig.* *(PDF/A-2/3) PDF_create_annotation( ): Nur* `type=Link` *ist erlaubt.*
Dateianhänge	*PDF/A-1: PDF_begin/end_document( ): Die Option* `attachments` *ist nicht zulässig.* *PDF/A-2: PDF_begin/end_document( ): Die Option* `attachments` *muss sich auf PDF/A-1- oder PDF/A-2-Dokumente beziehen.* *PDF/A-3: Beliebige Arten von Dateien können mit der Option* `associatedfiles` *angehängt werden, aber die Option* `attachments` *ist nicht zulässig. Die folgenden Bedingungen müssen erfüllt sein:* ► *Anhänge können über die Option* `associatedfiles` *von PDF_end_document( ), PDF_begin/end_page_ext( ), PDF_begin/end_dpart( ), PDF_begin_template_ext( ), PDF_load_image( ), PDF_open_pdi_page( ), PDF_load_graphics( ) unterschiedlichen Teilen des Dokuments zugeordnet sein. Jeder Anhang muss mit genau einem Teil des Dokuments verbunden werden, das heißt, jedes mit PDF_load_asset( ) erzeugte Handle darf nur an genau eine Option* `associatedfiles` *übergeben werden.* ► *Die Unteroptionen* `mimetype` *und* `relationship` *sind erforderlich.* ► *Die Unteroption* `description` *wird empfohlen.* ► *Die Unteroption* `external=true` *ist nicht zulässig.*
Formularfelder	*PDF_create_field( ) und PDF_create_fieldgroup( ) sind nicht zulässig.*
Aktionen und JavaScript	*PDF_create_action( ): Aktionen mit* `type=Hide`, `Launch`, `Movie`, `ResetForm`, `ImportData`, `JavaScript` *sind nicht zulässig; bei* `type=name` *sind nur* `NextPage`, `PrevPage`, `FirstPage` *und* `LastPage` *erlaubt.* *PDF_begin/end_document( ) und PDF_begin/end_page_ext( ): die Option* `action` *ist unzulässig.*

11.3.5 Zusätzliche Anforderungen für PDF/A Level U

Die meisten Standardanforderungen für PDF/A-2u und PDF/A-3u werden automatisch von PDFlib erfüllt. Bei der Generierung von Level-U-konformen Dokumenten ist lediglich eine einzige Operation eingeschränkt, siehe Tabelle 11.8. Wenn Ihre Anwendung also bereits PDF/A-2b oder PDF/A-3b erzeugt und die Anforderung aus Tabelle 11.8 erfüllt ist, sind die generierten Dokumente ebenfalls konform zu PDF/A-2u bzw. PDF/A-3u.

Tabelle 11.8 Zusätzliche PDF/A-Anforderung für die Konformität zu Level U

Kriterium	Für PDF/A-2u/3u-Konformität erforderliche PDFlib-Funktionen und -Optionen
Fonts	*Die Fontoption* `unicodemap=false` *ist nicht zulässig.*

11.3.6 Zusätzliche Anforderungen für PDF/A Level A

Für PDF/A-1a, PDF/A-2a und PDF/A-3a müssen alle Bedingungen erfüllt werden, die auch bei der Erstellung von Tagged PDF gelten, siehe Abschnitt 10.3, »Grundlagen von Tagged PDF«, Seite 282. Tabelle 11.9 führt erforderliche und empfohlene Operationen für die Er-

zeugung von Ausgabe gemäß Level A auf. Außer den allgemeinen Regeln für Tagged PDF sollten unbedingt die PDF/UA-Anforderungen erfüllt werden, um die Barrierefreiheit der generierten Dokumente zu verbessern; für weitere Informationen siehe Abschnitt 11.6, »PDF/UA für Barrierefreiheit«, Seite 356.

Die Benutzer sind für die Erstellung passender Strukturinformationen verantwortlich. Ist in einem Dokument der gesamte Text in einem einzigen Strukturelement abgelegt, so ist das zwar technisch korrektes PDF/A, verfehlt aber das Ziel der getreuen semantischen Wiedergabe.

Tabelle 11.9 Zusätzliche Anforderungen für die Konformität zu PDF/A Level A

Kriterium	**Anforderungen an PDFlib-Funktionen und -Optionen für PDF/A-1a/2a/3a-Konformität**
Fonts	*Die Fontoption* `unicodemap=false` *ist nicht zulässig.*
Tagged PDF	*Es müssen alle Bedingungen für Tagged PDF erfüllt werden (siehe Abschnitt 10.3, »Grundlagen von Tagged PDF«, Seite 282). Die Strukturhierarchie des Dokuments sollte die logische Struktur des Dokuments so genau wie möglich wiedergeben.*
Empfehlungen für Tagged PDF	*Folgende Maßnahmen sind unbedingt empfehlenswert:* ► *Alle PDF/UA-Regeln sollten eingehalten werden; für weitere Informationen siehe Abschnitt 11.6, »PDF/UA für Barrierefreiheit«, Seite 356.* ► *Mit der Option* `Lang` *von PDF_begin/end_document( ) sollte die Standardsprache des Dokuments festgelegt werden.* ► *Für Dokumentbestandteile ohne Text, z.B. Rasterbilder, sollte ein Alternativtext mit der Option* `Alt` *von PDF_begin_item( ) oder mit der Option* `tag` *von PDF_fit_image( ), PDF_fit_pdi_page( ) usw. übergeben werden.* ► *Für Abkürzungen und Akronyme sollte mit der Option* `E` *von PDF_begin_item( ) ein geeigneter Expansionstext definiert werden.* ► *Angaben für die Paginierung wie Kopf-/Fußzeilen und Seitenzahlen sollten als* `Artifact` *mit* `artifacttype=pagination` *ausgezeichnet werden.*
Wortgrenzen	*Wörter müssen durch Leerzeichen (U+0020) voneinander getrennt werden. Mit der Option* `autospace` *lässt sich dies automatisieren.*
Textausgabe und PUA-Unicode-Zeichen	*(PDF/A-2a/3a) PUA-Unicode-Zeichen (z.B. Logos und Symbole) benötigen einen geeigneten Ersatztext, der in der Option* `ActualText` *von PDF_begin_item( ) für das Inhaltselement oder in der entsprechenden Option* `tag` *der jeweiligen Ausgabefunktion übergeben wird (siehe unten).*
Anmerkungen	*PDF_create_annotation( ): Bei Anmerkungen ohne Text sollte die Option* `contents` *verwendet werden.*

PUA-Zeichen. Für PDF/A-2a und PDF/A-3a bestehen zusätzliche Anforderungen für Zeichen mit einem Unicode-Wert in der *Private Use Area* (PUA), also hauptsächlich im Bereich U+E000 - U+F8FF (siehe Abschnitt »BMP und PUA«, Seite 98). Bei PUA-Zeichen handelt es sich meist um dekorative Glyphen und Symbolglyphen oder um selbstdefinierte Glyphen wie Firmenlogos. Für PDF/A-2a/3a müssen PUA-Zeichen mit dem Attribut *ActualText* versehen werden, das eine textliche Darstellung des Zeichens enthält. *ActualText* kann einem einzelnen PUA-Zeichen oder einer längeren Sequenz von Zeichen zugewiesen werden, die ein PUA-Zeichen enthält. Wir empfehlen, die Option *ActualText* mit dem Inline-Level-Element *Span* zu übergeben.

Mit *PDF_info_font( )* können Sie den Unicode-Wert eines bestimmten Codes für einen bestimmten Font überprüfen (siehe Abschnitt 5.6.2, »Fontspezifische Abfrage von Encoding, Unicode und Glyphnamen«, Seite 152):

```
uv = (int) p.info_font(font, "unicode", "code=" + c);
```

Wenn der sich daraus ergebende Unicode-Wert *uv* in den PUA-Bereich fällt, benötigt er das Attribut *ActualText*. Im folgenden Codefragment wird ein Font names *PDFlibLogo* verwendet, der eine grafische Darstellung des PDFlib-Firmenlogos enthält. Bei der Platzierung des Logos auf einer Seite wird in der *tag*-Option ein *Span*-Element mit einer entsprechenden Unteroption *ActualText* übergeben, die den Text *PDFlib Logo* enthält:

```
p.fit_textline(text, 50, 700,
        "fontname=PDFlibLogo encoding=unicode embedding fontsize=24 " +
        "tag={tagname=Span ActualText={PDFlib Logo}}");
```

Wenn Sie keine Informationen über die Glyphe und damit keinen passenden *ActualText* zur Verfügung haben, können Sie den Fontnamen der Glyphe verwenden. Dieser lässt sich folgendermaßen ermitteln:

```
gn_idx = (int) p.info_font(font, "glyphname", "code=" + c);
glyphname = p.get_option(gn_idx, "");
```

Der Glyphname kann für *ActualText*, eventuell in Kombination mit einem bestimmten Satz oder einer Phrase verwendet werden. Zum Beispiel enthält der Code *0x1A* im Font *Wingdings* das Bild einer Computertastatur mit dem Glyphnamen *keyboard*. Dieser Glyphe wird der PUA-Wert U+F037 zugeordnet. Für dieses Symbol sollten Sie den ActualText *symbol for keyboard* verwenden. Bedenken Sie dabei, dass die Erstellung von *ActualText* per Programm nur eine Notlösung darstellen kann. Selbstdefinierter Text ist einem maschinell erzeugten *ActualText* immer vorzuziehen.

11.3.7 Import von PDF/A-Dokumenten mit PDI

Wenn Seiten eines vorhandenen PDF-Dokuments in ein PDF/A-kompatibles Ausgabedokument importiert werden sollen, gelten spezielle Regeln (der PDF-Import wird in Abschnitt 7.3, »Import von PDF-Seiten mit PDI«, Seite 202, beschrieben). Alle importierten Dokumente müssen einer PDF/A-Konformitätsstufe genügen, die gemäß Tabelle 11.10 mit dem aktuellen PDF/A-Modus vereinbar ist.

Hinweis *PDFlib kann PDF-Dokumente nicht bezüglich PDF/A-Konformität validieren und ist auch nicht in der Lage, aus beliebigen PDF-Eingabedokumenten konforme PDF/A-Dokumente zu erstellen.*

Wenn in PDFlib eine bestimmte PDF/A-Konformitätsstufe konfiguriert ist und die importierten Dokumente einer damit vereinbaren Kompatibilitätsstufe genügen, so ist gewährleistet, dass die generierte Ausgabe der gewählten PDF/A-Konformitätsstufe genügt. Dokumente, die nicht der aktuellen PDF/A-Stufe genügen, werden in *PDF_open_pdi_document( )* zurückgewiesen.

Tabelle 11.10 Zulässige PDF/A-Eingabestufen für verschiedene PDF/A-Ausgabestufen

	PDF/A-Konformitätsstufe des importierten Dokuments				
PDF/A-Ausgabestufe	**PDF/A-1a:2005**	**PDF/A-1b:2005**	**PDF/A-2a, PDF/A-3a**	**PDF/A-2b, PDF/A-3b**	**PDF/A-2u, PDF/A-3u**
PDF/A-1a:2005	zulässig	–	–	–	–
PDF/A-1b:2005	zulässig	zulässig	–	–	–
PDF/A-2a, PDF/A-3a	zulässig	–	zulässig	–	–
PDF/A-2b, PDF/A-3b	zulässig	zulässig	zulässig	zulässig	zulässig
PDF/A-2u, PDF/A-3u	zulässig	–	zulässig	–	zulässig

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `pdfa/import_pdfa`.

Werden mehrere PDF/A-Dokumente importiert, müssen diese allesamt für eine gemäß Tabelle 11.11 kompatible Druckausgabebedingung erstellt worden sein. Die Druckausgabebedingungen müssen in allen importierten Dokumenten identisch oder vereinbar sein. Sie müssen darauf achten, dass diese Voraussetzung erfüllt ist.

Tabelle 11.11 Vereinbarkeit von Druckausgabebedingungen beim Import von PDF/A-Dokumenten

	Druckausgabebedingung des importierten Dokuments			
Druckausgabebedingung des generierten Dokuments	**keine**	**Graustufen**	**RGB**	**CMYK**
keine	ja	–	–	–
ICC-Profil für Graustufen	ja	ja[1]	–	–
ICC-Profil für RGB	ja	–	ja[1]	–
ICC-Profil für CMYK	ja	–	–	ja[1]

1. Druckausgabebedingung von importiertem und generiertem Dokument müssen identisch sein.

PDFlib kann bestimmte Punkte zwar korrigieren, ist aber nicht für die Überprüfung und Durchsetzung der vollständigen PDF/A-Konformität von importierten Dokumenten gedacht. PDFlib bettet zum Beispiel keine fehlenden Fonts in importierten PDF-Seiten ein.

Um importierte PDF-Seiten zu einem Ausgabedokument zusammenzustellen, das derselben PDF/A-Konformitätsstufe und Druckausgabebedingung wie die Eingabedokumente genügt, können Sie den PDF/A-Level eines importierten PDFs wie folgt abfragen:

```
pdfalevel = p.pcos_get_string(doc, "pdfa");
```

Sofern das importierte Dokument konform zu einem PDF/A-Level ist, gibt diese Anweisung in einem String den entsprechenden PDF/A-Level aus, andernfalls *none*. Mit dem zurückgegebenen String und mit der Option *pdfa* von *PDF_begin_document( )* kann die entsprechende PDF/A-Konformitätsstufe für das Ausgabedokument gesetzt werden.

Kopieren der PDF/A-Druckausgabebedingung eines importierten Dokuments. Es ist generell möglich, die PDF/A-Druckausgabebedingung aus einem importierten Dokument zu kopieren. Da PDF/A-Dokumente (im Gegensatz zu PDF/X) nicht unbedingt eine Druckausgabebedingung enthalten müssen, müssen Sie zunächst mit pCOS überprüfen, ob überhaupt eine Ausgabebedingung vorhanden ist, bevor Sie diese kopieren.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `pdfa/import_pdfa`.

Diese Methode kann als Alternative zur Festlegung der Druckausgabebedingung mit *PDF_load_iccprofile( )* verwendet werden. Hierbei wird die Druckausgabebedingung des importierten Dokuments in das generierte Ausgabedokument kopiert. Das Kopieren der Ausgabebedingung funktioniert für importierte PDF/A- und PDF/X-Dokumente.

11.3.8 XMP-Dokumentmetadaten für PDF/A

PDF/A nutzt bei der Einbettung von Metadaten in PDF-Dokumente das XMP-Format. PDF/A unterstützt zwei Typen von Metadaten auf Dokumentebene: eine Reihe allgemein bekannter Metadatenschemas, die sogenannten vordefinierte Schemas, die der zugrundeliegenden XMP-Spezifikation entnommen werden, sowie selbstdefinierte Extension-Schemas. PDFlib erstellt in den XMP-Metadaten automatisch alle zur PDF/A-Konformität erforderlichen Einträge sowie verschiedene allgemeine Einträge (zum Beispiel *CreationDate)*.

XMP-Dokumentmetadaten können mit der Option *metadata* von *PDF_begin/end_document( )*, *PDF_end_document( )* oder beiden übergeben werden. Im PDF/A-Modus überprüft PDFlib, ob die selbstdefinierten XMP-Dokumentmetadaten den PDF/A-Anforderungen genügen. Für Metadaten auf Komponentenebene (wie Seite oder Bild) gibt es keine speziellen PDF/A-Anforderungen.

XMP-Metadaten aus importierten PDF-Dokumenten lassen sich mit Hilfe des pCOS-Pfads */Root/Metadata* aus dem Eingabe-PDF extrahieren.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `interchange/import_xmp_from_pdf`.

Vordefinierte XMP-Schemas. Die Verwendung von XMP für Dokumentmetadaten bei PDF/A basiert auf den folgenden Spezifikationen:

- PDF/A-1: XMP-2004-Spezifikation
- PDF/A-2 und PDF/A-3: XMP 2005[1]

Bei den in der jeweiligen XMP-Spezifikation beschriebenen Schemas handelt es sich um vordefinierte Schemas, die in Tabelle 11.12 mit dem URI des Namensraums und dem zugehörigen bevorzugten Präfix aufgeführt sind. In PDF/A dürfen nur Properties aus den vordefinierten Schemas verwendet werden, es sei denn, es liegt ein Extension-Schema vor (siehe unten). Eine vollständige Liste der in den vordefinierten XMP-2004-Schemas für PDF/A-1 verfügbaren Eigenschaften finden Sie in TechNote 0008 des PDF/A Competence Center der PDF Association. Bei
PDF/A-2/3 kommen die vordefinierten Schemas aus XMP 2005. Da sich die zusätzlichen Schemas auf Rasterbilder und dynamische Medien beziehen, sind sie für Dokumentmetadaten nicht relevant.

1. Siehe www.aiim.org/documents/standards/xmpspecification.pdf

Tabelle 11.12 Vordefinierte XMP-Schemas für PDF/A (weitere Informationen siehe XMP 2004 und XMP 2005)

Schemaname und -beschreibung	URI des Namensraums	Bevorzugtes Präfix des Namensraums
XMP-2004-Schemas für PDF/A-1, PDF/A-2 und PDF/A-3		
Adobe PDF Schema	`http://ns.adobe.com/pdf/1.3/`	`pdf`
Dublin Core Schema	`http://purl.org/dc/elements/1.1/`	`dc`
EXIF Schema für EXIF-spezifische Properties	`http://ns.adobe.com/exif/1.0/`	`exif`
EXIF Schema für TIFF-Properties	`http://ns.adobe.com/tiff/1.0/`	`tiff`
Photoshop Schema	`http://ns.adobe.com/photoshop/1.0/`	`photoshop`
XMP Basic Job Ticket Schema	`http://ns.adobe.com/xap/1.0/bj`	`xmpBJ`
XMP Basic Schema	`http://ns.adobe.com/xap/1.0/`	`xmp`
XMP Media Management Schema	`http://ns.adobe.com/xap/1.0/mm/`	`xmpMM`
XMP Paged-Text Schema	`http://ns.adobe.com/xap/1.0/t/pg/`	`xmpTPg`
XMP Rights Management Schema	`http://ns.adobe.com/xap/1.0/rights/`	`xmpRights`
Zusätzliche XMP-2005-Schemas für PDF/A-2 und PDF/A-3		
Camera Raw Schema	`http://ns.adobe.com/camera-rawsettings/1.0/`	`crs`
EXIF Schema für zusätzliche EXIF-Properties	`http://ns.adobe.com/exif/1.0/aux/`	`aux`
XMP Dynamic Media Schema	`http://ns.adobe.com/xmp/1.0/DynamicMedia/`	`xmpDM`

XMP-Extension-Schemas. Falls die vordefinierten Schemas Ihren Metadaten-Anforderungen nicht genügen, können Sie ein XMP-Extension-Schema definieren. PDF/A beschreibt einen Erweiterungsmechanismus, der verwendet werden muss, wenn selbstdefinierte Schemas in ein PDF/A-Dokument eingebettet werden sollen. Tabelle 11.13 zeigt die Schemas und ihre Properties, die zur Beschreibung von Extension-Schemas zu verwenden sind. Zu jedem Schema wird der URI des Namensraums sowie das erforderliche Präfix des Namensraums angeführt. Beachten Sie, dass die jeweiligen Namensraum-Präfixe im Gegensatz zu den bevorzugten Namensraum-Präfixen für vordefinierte Schemas obligatorisch sind.

Für weitere Informationen zur Erstellung von XMP-Extension-Schemas siehe TechNote 0009 des PDF/A Competence Center.

Cookbook *Vollständige Code- und XMP-Beispiele finden Sie in den Cookbook-Topics* `pdfa/pdfa_extension_schema` *und* `pdfa/pdfa_extension_schema_with_type`.

Tabelle 11.13 Container-Schema für PDF/A-Extension-Schema und Hilfsschemas

Schemaname und -beschreibung	URI[1] des Namensraums	Erforderliches Präfix des Namensraums
Container-Schema für PDF/A-Extension-Schema: Container für alle Beschreibungen von eingebetteten Extension-Schemas	`http://www.aiim.org/pdfa/ns/extension/`	`pdfaExtension`
Wertetyp für PDF/A-Schema: beschreibt ein einzelnes Extension-Schema mit einer beliebigen Anzahl von Properties	`http://www.aiim.org/pdfa/ns/schema#`	`pdfaSchema`

Tabelle 11.13 Container-Schema für PDF/A-Extension-Schema und Hilfsschemas

Schemaname und -beschreibung	***URI[1] des Namensraums***	***Erforderliches Präfix des Namensraums***
Wertetyp für PDF/A-Properties: beschreibt eine einzelne Property	`http://www.aiim.org/pdfa/ns/property#`	`pdfaProperty`
Wertetyp für PDF/A-ValueType: beschreibt einen benutzerdefinierten Wertetyp, der in den Properties des Extension-Schemas verwendet wird; nur erforderlich, wenn Typen außerhalb der Typenliste von XMP 2004 verwendet werden.	`http://www.aiim.org/pdfa/ns/type#`	`pdfaType`
Feldtyp für PDF/A-Schema: beschreibt ein Feld in einem strukturierten Typ	`http://www.aiim.org/pdfa/ns/field#`	`pdfaField`

1. Beachten Sie, dass die URIs des Namensraums in ISO 19005-1 nicht korrekt aufgeführt wurden; dies wurde im Technical Corrigendum 1 korrigiert.

11.4 PDF/X zur Druckproduktion

11.4.1 PDF/X-Standards

Die in ISO 15930 definierten PDF/X-Standards wurden mit dem Ziel entwickelt, eine konsistente und stabile Untermenge von PDF bereitzustellen, die sich zur Anlieferung von Daten an Druckereien eignet. PDFlib generiert Ausgabe und verarbeitet Eingabe, die zu den im Folgenden beschriebenen PDF/X-Varianten konform ist.

PDF/X-1a:2003, definiert in ISO 15930-4. Dieser Standard ist der Nachfolger von PDF/X-1a:2001. Er basiert auf PDF 1.4, wobei einige Funktionen, wie zum Beispiel Transparenz, unzulässig sind. PDF/X-1a:2003 ist eine Teilmenge von PDF/X-3:2003, unterstützt CMYK- und Schmuckfarbe sowie CMYK-Ausgabegeräte.

PDF/X-3:2003, definiert in ISO 15930-6. Dieser Standard ist der Nachfolger von PDF/X-3:2002. Er basiert auf PDF 1.4 und unterstützt Arbeitsabläufe mit geräteunabhängigen Farben, Graustufen, CMYK und Schmuckfarben. Der Standard ist vor allem in Europa verbreitet. Zulässig sind Monochrom-, RGB- und CMYK-Ausgabegeräte. Einige Funktionen von PDF 1.4, wie zum Beispiel Transparenz, sind unzulässig.

PDF/X-4, definiert in ISO 15930-7. Dieser Standard kann als Nachfolger von PDF/X-1a und PDF/X-3 betrachtet werden. Er basiert auf PDF 1.6 und umfasst die folgenden Varianten:

- Bei PDF/X-4 sind mit einigen Einschränkungen, Transparenz und Ebenen erlaubt; einige andere PDF-1.6-Funktionen sind jedoch immer noch unzulässig.
- Bei PDF/X-4p sind ICC-basierte Druckausgabebedingungen erlaubt, die zur Platzersparnis außerhalb eines PDF-Dokuments gespeichert werden können.

PDFlib implementiert die Version 15930-7:2010 des PDF/X-4-Standards. Verglichen mit der Version von 2008 führt die Version von 2010 Änderungen bei der Verarbeitung von Ebenen ein.

PDF/X-5, definiert in ISO 15930-8. Dieser Standard zielt auf einen »partiellen Datenaustausch« ab, bei dem sich Lieferant und Empfänger einer Datei absprechen müssen. Der Standard kann als Erweiterung von PDF/X-4 und PDF/X-4p (das heißt, basierend auf PDF 1.6) betrachtet werden und umfasst die folgenden Varianten:

- Bei PDF/X-5g darf auf grafische Inhalte außerhalb des PDF-Dokuments verwiesen werden; dies erfordert Absprachen zwischen Lieferant und Empfänger des Dokuments.
- Bei PDF/X-5pg sind externe grafische Inhalte und externe ICC-basierte Druckausgabebedingungen erlaubt.
- PDF/X-5n unterstützt externe ICC-basierte Druckausgabebedingungen für die Druck-Charakterisierung mit n-Farbdarstellung. Diese Variante wird von PDFlib nicht unterstützt.

Wenn keine besonderen PDF/X-5-Funktionen benötigt werden, sollte das Dokument gemäß den allgemeineren Standards PDF/X-4 oder PDF/X-4p erstellt werden.

ISO 15930-8:2008 enthält verschiedene Fehler bezüglich der XMP-Identifikationseinträge für extern referenzierte Grafiken. Die überarbeitete Version ISO 15930-8:2010 er-

setzt die Version von 2008. PDFlib implementiert PDF/X-5:2010 einschließlich Corrigendum 1 aus dem Jahr 2011.

PDF/X-5-Dokumente in Acrobat. Beachten Sie beim Anzeigen von PDF/X-5-Dokumenten in Acrobat Folgendes:

- Beim Öffnen von PDF/X-5pg-Dokumenten mit referenzierten ICC-Druckausgabebedingungen kann sich Acrobat XI aufhängen oder abstürzen. Für weitere Inforationen hierzu und eine Lösung des Problems siehe Abschnitt »Probleme bei Acrobat mit referenzierten ICC-Druckausgabebedingungen«, Seite 341. Für die Validierung solcher Dokumente mit dem Preflight-Werkzeug gilt die gleiche Lösung.
- Referenzierte Seiten erfordern die sorgfältige Konfiguration von Acrobat. Für weitere Informationen siehe Abschnitt »Einsatz referenzierter Seiten in Acrobat«, Seite 80.

11.4.2 Allgemeine Anforderungen

Cookbook *Codebeispiele für die Erzeugung von PDF/X finden Sie in der Kategorie* `pdfx` *des PDFlib Cookbook.*

Wenn sich der PDFlib-Client an die in diesem Kapitel dokumentierten Regeln hält, ist standardkonforme PDF/X-Ausgabe gewährleistet. Entdeckt PDFlib eine Regelverletzung bei der PDF/X-Erstellung, so wird eine Exception ausgelöst. In diesem Fall wird keine PDF-Ausgabe erzeugt. Tabelle 11.14 zeigt die allgemeinen Anforderungen, die zur Erzeugung von PDF/X-konformer Ausgabe erfüllt sein müssen.

Tabelle 11.14 Allgemeine Anforderungen für die Konformität zu PDF/X

Kriterium	*Für PDF/X-Konformität erforderliche PDFlib-Funktionen und -Optionen*
PDF/X-Konformitätsstufe und PDF-Kompatibilität	*PDF_begin_document( ): Die Option* `pdfx` *muss auf die erforderliche PDF/X-Konformitätsstufe gesetzt werden, z.B.* `pdfx=PDF/X-4`**.** *(PDF/X-1a:2003 und PDF/X-3:2003) Operationen, die PDF 1.5 oder höher benötigen, sind nicht zulässig.* *(PDF/X-4 und PDF/X-5) Operationen, die PDF 1.7 oder höher benötigen, sind nicht zulässig.*
Fonts	*Die Fontoption* `embedding` *muss auf* `true` *gesetzt werden. Auch die PDF-Standardfonts müssen eingebettet werden.*
Seitengrößen	*PDF_begin/end_page_ext( ): Die Seitenboxen, die mit den Optionen* `cropbox`, `bleedbox`, `trimbox` *und* `artbox` *gesetzt werden können, müssen die folgenden Anforderungen erfüllen:* ► *Es muss entweder die* `TrimBox` *oder die* `ArtBox` *angegeben werden, aber nicht beide. Ist keine von beiden vorhanden, verwendet PDFlib die* `CropBox` *(sofern vorhanden) als* `TrimBox` *bzw. die* `MediaBox`*, wenn auch die* `CropBox` *nicht vorhanden ist.* ► *Die* `BleedBox`*, sofern vorhanden, muss in der* `ArtBox` *oder der* `TrimBox` *enthalten sein.* ► *Die* `CropBox`*, sofern vorhanden, muss in der* `ArtBox` *oder der* `TrimBox` *enthalten sein.*
Ebenen	*(PDF/X-1a und PDF/X-3) Ebenen sind nicht zulässig.* *(PDF/X-4 und PDF/X-5) Ebenen dürfen verwendet werden, aber einige Optionen von PDF_define_layer( ) und PDF_set_layer_dependency( ) sind unzulässig.*
Dokument-Infofelder und XMP-Metadaten	*Die Infofelder für* `Creator` *und* `Title` *müssen mit PDF_set_info( ) oder (in PDF/X-4 und PDF/X-5) mit den XMP-Properties* `xmp:CreatorTool` *und* `dc:title` *in der Option* `metadata` *von PDF_begin/end_document( ) auf einen nicht leeren Wert gesetzt werden.* *Andere Werte als* `True` *oder* `False` *für das Infofeld* `Trapped` *von PDF_set_info( ) oder die entsprechende XMP-Property* `pdf:Trapped` *in der Option* `metadata` *von PDF_begin/end_document( ) sind nicht zulässig.*

Tabelle 11.14 Allgemeine Anforderungen für die Konformität zu PDF/X

Kriterium	Für PDF/X-Konformität erforderliche PDFlib-Funktionen und -Optionen
Sicherheit	*PDF_begin_document( ): Die Optionen* `userpassword`, `masterpassword` *und* `permissions` *sind nicht zulässig.*
Externer grafischer Inhalt (Referenzen)	*(PDF/X-1a/3/4) Die Option* `reference` *von PDF_begin_template_ext( ), PDF_load_graphics( ) und PDF_open_pdi_page( ) ist nicht zulässig.* *(PDF/X-5g und PDF/X-5pg) Das Ziel, das in der Option* `reference` *von PDF_begin_template_ext( ), PDF_load_graphics( ) oder PDF_open_pdi_page( ) angegeben wird, muss einem der folgenden Standards genügen: PDF/X-1a:2003, PDF/X-3:2003, PDF/X-4, PDF/X-4p, PDF/X-5g oder PDF/X-5pg. Außerdem muss es für die gleiche Druckausgabebedingung aufbereitet worden sein. Da bestimmte XMP-Metadateneinträge im Zieldokument erforderlich sind, sind nicht alle PDF/X-Dokumente als Ziel akzeptabel. Mit PDFlib 8 oder höher erzeugte PDF/X-Dokumente können als Ziel verwendet werden.* *Für weitere Informationen zu der Option* `reference` *und der erforderlichen Acrobat-Konfiguration siehe Abschnitt 3.2.5, »Seiten aus externen PDF-Dokumenten referenzieren«, Seite 80.*
Dateigröße	*(PDF/X-4 und PDF/X-5) Die Dateigröße des generierten PDF-Dokuments darf 2 GB nicht überschreiten, und die Anzahl der PDF-Objekte muss kleiner als 8 388 607 sein. Für weitere Informationen siehe Abschnitt 3.1.5, »Maximalgröße von PDF-Dokumenten und andere Grenzwerte«, Seite 71.*
PDF-Import	*PDF_open_pdi_document( ) ist eingeschränkt, außer wenn* `infomode=true`*; siehe Abschnitt 11.3.7, »Import von PDF/A-Dokumenten mit PDI«, Seite 332.*

11.4.3 Anforderungen für Farbe und Rasterbilder

PDF/X garantiert die farbgetreue Wiedergabe durch die erforderlichen geräteunabhängigen Farbspezifikationen. Farbräume können aus folgenden Quellen stammen:

- Bilder, die direkt mit *PDF_load_image( )* und *PDF_fill_imageblock ( )* geladen werden und indirekt mit *PDF_load_graphics( )*
- Explizite Farbfestlegungen mit *PDF_set_graphics_option( )* oder *PDF_setcolor( )*
- Farbspezifikationen über Optionslisten, zum Beispiel in Textflows
- Mischfarbräume für Transparenzgruppen: *PDF_begin/end_page_ext( )*, *PDF_begin_template_ext( )* und *PDF_open_pdi_page( ):* Option *transparencygroup* mit Unteroption *colorspace*
- Bei interaktiven Elementen kann eine Rahmenfarbe definiert werden

Tabelle 11.15 zeigt die PDF/X-Anforderungen für die Farbverarbeitung, die in allen oben aufgeführten Operationen erfüllt werden müssen. Die Punkte beziehen sich auf alle PDF/X-Konformitätsstufen, sofern nicht anderweitig angemerkt.

Druckausgabebedingungen und Standard-Druckausgabebedingungen. Die Druckausgabebedingung definiert das beabsichtigte Zielgerät oder die Druckbedingung, was insbesondere für zuverlässige Proofs sinnvoll ist. Die Details unterscheiden sich je nach PDF/X-Variante:

- PDF/X-1a/3/4/5g: Die Druckausgabebedingung kann durch Einbettung eines ICC-Profils für das Ausgabegerät.
- Nur bei PDF/X-4p und PDF/X-5pg: durch Referenzierung eines externen ICC-Profils für die Druckausgabebedingung (das *p* im Namen des Standards bedeutet, dass ein externes Profil referenziert wird). Anders als bei Standard-Druckausgabebedingungen wird die ICC-basierte Druckausgabebedingung nicht nur durch den Namen referenziert. Vielmehr wird eine starke Referenz erzeugt, die bei der Generierung des Do-

Tabelle 11.15 Anforderungen für Farbe und Rasterbilder für die Konformität zu PDF/X

Kriterium	Für PDF/X-Konformität erforderliche PDFlib-Funktionen und -Optionen
Druckausgabe-bedingung (output intent)	*Unmittelbar nach PDF_begin_document( ) muss entweder PDF_load_iccprofile( ) mit* `usage=outputintent` *oder PDF_process_pdi( ) mit* `action=copyoutputintent` *aufgerufen werden (aber nicht beide). Werden HKS- oder Pantone-Schmuckfarben, ICC-basierte oder Lab-Farben verwendet, muss ein ICC-Profil für die Druckausgabebedingung eingebettet werden; eine Standard-Druckausgabebedingung ist in solchen Fällen nicht zulässig.* *PDF/X-1a: Das Ausgabegerät muss ein Monochrom- oder CMYK-Gerät sein;* *PDF/X-3: Das Ausgabegerät muss ein Monochrom-, RGB- oder CMYK-Gerät sein.*
Graustufen	*(PDF/X-1a) In PDF_begin_page_ext( ) ist die Option* `defaultgray` *unzulässig.* *(PDF/X-3/4/5) Graustufen-Farben können nur verwendet werden, wenn eine Ausgabebedingung für Graustufen, RGB oder CMYK vorhanden ist oder die Option* `defaultgray` *von PDF_begin_page_ext( ) gesetzt wurde*
RGB-Farbe	*(PDF/X-1a) RGB-Farbe und die Option* `defaultrgb` *in PDF_begin_page_ext( ) sind unzulässig.* *(PDF/X-3/4/5) RGB-Farbe kann nur verwendet werden, wenn eine Ausgabebedingung für RGB vorhanden ist oder die Option* `defaultrgb` *von PDF_begin_page_ext( ) gesetzt wurde.*
CMYK-Farbe	*(PDF/X-1a) Bei PDF_begin_page_ext( ) ist die Option* `defaultcmyk` *unzulässig.* *(PDF/X-3/4/5) CMYK-Farbe kann nur verwendet werden, wenn eine Ausgabebedingung für CMYK vorhanden ist oder die Option* `defaultcmyk` *von PDF_begin_page_ext( ) gesetzt wurde.*
ICC-basierte Farbe	*(PDF/X-1a) Der Farbraum* `iccbasedgray/rgb/cmyk` *in PDF_set_graphics_option( ) oder PDF_setcolor( ) und die Optionen* `iccprofilegray/rgb/cmyk` *sind unzulässig.*
Lab-Farbe	*(PDF/X-1a) Der Farbraum Lab in PDF_set_graphics_option( ) oder PDF_setcolor( ) ist unzulässig.*
Rasterbilder	*Die Optionen* `OPI-1.3`, `OPI-2.0` *in PDF_load_image( ) sind unzulässig.* *(PDF/X-1a) Rasterbilder und Grafiken in RGB-, ICC-basierter, YCbCr- oder Lab-Farbe sind unzulässig. Bei Bildern, die mit der Option* `colorize` *eingefärbt wurden, muss die Alternativfarbe der benutzten Schmuckfarbe denselben Bedingungen genügen.* *(PDF/X-1 und PDF/X-3) JBIG2-Bilder sind unzulässig.* *(PDF/X-4/5) JPEG-2000-Bilder müssen bestimmte Bedingungen erfüllen, siehe Abschnitt »JPEG-2000-Bilder«, Seite 186.*
Transparenz	*(PDF/X-1 und PDF/X-3) Transparenz ist unzulässig; dies betrifft die folgenden API-Funktionen:* ► *PDF_load_image( ): Die Option* `masked` *ist unzulässig, sofern sich die Transparenzmaske nicht auf ein 1-Bit-Bild bezieht.* ► *PDF_load_image( ): Bilder mit impliziter Transparenz (Alphakanal) sind unzulässig; sie müssen mit der Option* `ignoremask` *geladen werden.* ► *PDF_load_graphics( ): SVG-Grafik mit Transparenzelementen ist unzulässig.* ► *PDF_create_gstate( ): Die Optionen* `opacityfill` *und* `opacitystroke` *sind unzulässig, sofern sie nicht den Wert 1 haben;* `blendmode` *darf nur mit dem Wert* `Normal` *verwendet werden.* *(PDF/X-4/5) Transparenz bei Rasterbildern und Grafiken ist erlaubt.*
Transparenzgruppen	*PDF_begin/end_page_ext( ), PDF_begin_template_ext( ), PDF_open_pdi_page( ) und PDF_load_graphics( ): Die Option* `transparencygroup` *ist folgendermaßen eingeschränkt:* *(PDF/X-1a und PDF/X-3) Die Option* `transparencygroup` *ist unzulässig.* *(PDF/X-4/5) Die Unteroption* `colorspace` *der Option* `transparencygroup` *muss die obigen Anforderungen für Graustufen, RGB- und CMYK-Farbe erfüllen.*

kuments ein lokal verfügbares ICC-Profil benötigt. Obwohl das ICC-Profil nicht in die PDF-Ausgabe eingebettet wird, muss es zur Erzeugung einer starken Referenz dennoch zum Zeitpunkt der PDF-Generierung verfügbar sein. Der Speicherort des ICC-

Profils muss mit einer oder mehreren gültigen URLs in der Option *urls* angegeben werden:

```
if (p.load_iccprofile("CGATS TR 001",
        "usage=outputintent urls={http://www.color.org}") == -1)
{
    /* Fehler */
}
```

- Nur bei PDF/X-1a und PDF/X-3: durch Angabe des Namen einer Standard-Druckausgabebedingung ohne Einbettung des entsprechenden ICC-Profils. Die Standard-Druckausgabebedingungen sind PDFlib intern bekannt; eine vollständige Namensliste finden Sie in der PDFlib-Referenz. ICC-Profile für diese Druckbedingungen müssen nicht lokal verfügbar sein. Bei PDF/X-3-Ausgabe mit HKS-, Pantone-, ICC-basierten oder Lab-Farben dürfen Sie keine Standard-Druckausgabebedingungen verwenden, sondern müssen stattdessen das ICC-Profil des Ausgabegeräts einbetten. Bei PDF/X-4/5 gilt folgende spezielle Regel: ein CMYK-Profil für eine Druckausgabebedingung (also über *usage= outputintent* geladen) kann im selben Dokument nicht für einen Farbraum vom Typ *ICCBased* verwendet werden (also über *usage=iccbased* geladen). Der PDF/X-Standard schreibt dies nur für CMYK-Profile vor; Graustufen- oder RGB-Profile sind davon nicht betroffen. Eine ähnliche Bedingung gilt für importierte Dokumente: wenn eine importierte Seite dass gleiche ICC-Profil für CMYK verwendet wie die Druckausgabebedingung des generierten Dokuments, wird sie von *PDF_open_pdi_page( )* zurückgewiesen.

Auswahl einer passenden PDF/X-Ausgabebedingung. Die PDF/X-Ausgabebedingung wählen Sie in der Regel gemeinsam mit Ihrer Druckerei oder den Dienstleistern, die für die Druckproduktion verantwortlich sind. Erhalten Sie von der Druckerei keine Informationen über die Wahl der Ausgabebedingung, so können Sie als Arbeitsgrundlage die in Tabelle 11.16 angeführten gebräuchlichen Druckausgabebedingungen verwenden (entnommen aus den PDF/X-FAQ).

Tabelle 11.16 Für häufige Drucksituationen geeignete PDF/X-Ausgabebedingungen

	Europa	***Nordamerika***
Zeitschriften-anzeigen	*FOGRA28*	*CGATS TR 001*
Zeitungsanzeigen	*IFRA26*	*IFRA30*
Bogenoffset	*Je nach Papiertyp:* *Typen 1 & 2 (gestrichen): FOGRA39* *Type 3 (LWC): FOGRA45* *Type 4 (ungestrichen): FOGRA47*	*Je nach Papiertyp:* *Grade 1 und 2 (premium coated): FOGRA39* *Grad 5: CGATS TR 001* *Ungestrichen: FOGRA47*
Rollenoffset	*Je nach Papiertyp:* *Typen 1 & 2 (gestrichen): FOGRA45* *Typ 4 (ungestrichen, weiß): FOGRA47* *Typ 5 (ungestrichen, gelblich): FOGRA30*	*Je nach Papiertyp:* *Grad 5: CGATS TR 001* *Ungestrichen (weiß): FOGRA47* *Ungestrichen (gelblich): FOGRA30*

Probleme bei Acrobat mit referenzierten ICC-Druckausgabebedingungen. Beachten Sie unbedingt, dass bei Acrobat bezüglich PDF/X-4p/5pg folgendes Problem besteht:

Acrobat XI stürzt ab oder hängt sich auf, wenn ein Dokument mit einem benutzerdefinierten Profil für eine Druckausgabebedingung geöffnet wird, das heißt, bei jedem PDF/X-4p/5pg-Dokument und entsprechenden PDF/VT-Dokumenten, die eine Druckausgabebedingung mit einem ICC-Profil referenzieren, das nicht im selben Verzeichnis wie die Acrobat-Installation liegt.

Lösung: das Problem lässt sich vermeiden, wenn Sie das referenzierte ICC-Profil in dem Verzeichnis ablegen, in dem Acrobat seine ICC-Profile speichert. Je nach Acrobat-Installation sollten Sie das referenzierte ICC-Profil in folgendem oder einem vergleichbaren Verzeichnis ablegen:

```
Windows: C:\Programme\Common Files\Adobe\Color\Profiles
OS X: /Library/Application Support/Adobe/Color/Profiles
```

Das Preflight-Werkzeug von Acrobat X und XI kann PDF/X-4p/5pg-Dateien nur erfolgreich validieren, wenn das referenzierte ICC-Profil so wie oben beschrieben im Acrobat-Verzeichnis vorhanden ist.

11.4.4 Anforderungen für interaktive Funktionen

Tabelle 11.17 führt alle Operationen auf, die bei der Erzeugung von PDF/X-konformer Ausgabe untersagt sind. Der Aufruf einer im PDF/X-Modus unzulässigen Funktion löst eine Exception aus.

Tabelle 11.17 PDF/X-Anforderungen für interaktive Funktionen

Kriterium	***Für PDF/X-Konformität unzulässige oder eingeschränkte PDFlib-Funktionen und -Optionen***
Anmerkungen und Formularfelder	*PDF_create_annotation( ), PDF_create_field( ): Anmerkungen innerhalb der* `BleedBox` *(oder* `TrimBox`/`ArtBox`*, falls keine* `BleedBox` *vorhanden) sind unzulässig.*
Dateianhänge	*(PDF/X-1a/3) PDF_begin/end_document( ): Die Option* `attachments` *ist unzulässig; PDF_create_annotation( ) mit* `type=FileAttachment` *ist unzulässig.*
Aktionen und JavaScript	*PDF_create_action( ): Alle Aktionen inklusive JavaScript sind unzulässig.*
Viewer-Voreinstellungen/Anzeige- und Druckbereiche	*PDF_begin/end_document( ): Die Unteroptionen* `viewwarea`*,* `printarea` *und* `printclip` *für die Option* `viewerpreferences` *dürfen nur mit den Werten* `media` *und* `bleed` *verwendet werden.*

11.4.5 Import von PDF/X-Dokumenten mit PDI

Wenn Seiten eines vorhandenen PDF-Dokuments in ein PDF/X-kompatibles Ausgabedokument importiert werden, gelten spezielle Regeln (für weitere Informationen siehe Abschnitt 7.3, »Import von PDF-Seiten mit PDI«, Seite 202). Alle importierten Dokumente müssen einem PDF/X-Level genügen, der zum generierten Ausgabedokument kompatibel ist (siehe Tabelle 11.18). Bei allen erlaubten Kombinationen von PDF/X-4/5-Ausgabe muss zusätzlich folgende Regel eingehalten werden: Wenn eine importierte Seite dasselbe ICC-basierte CMYK-Profil verwendet wie die Druckausgabebedingung des generierten Dokuments, wird es von *PDF_open_pdi_page( )* zurückgewiesen, da es den PDF/X-4/5-Standard verletzen würde.

Wird in PDFlib eine bestimmte PDF/X-Konformitätsstufe eingestellt und genügen die importierten Dokumente einer kompatiblen PDF/X-Stufe, dann ist die erzeugte Ausgabe garantiert konform zum eingestellten Level. Importierte Dokumente, die der einge-

Tabelle 11.18 Zulässige PDF/X-Eingabestufen für verschiedene PDF/X-Ausgabestufen

	PDF/X-Konformitätsstufe des importierten Dokuments					
PDF/X-Ausgabestufe	**PDF/X-1a:2003**	**PDF/X-3:2003**	**PDF/X-4**	**PDF/X-4p**	**PDF/X-5g**	**PDF/X-5pg**
PDF/X-1a:2003	zulässig					
PDF/X-3:2003	zulässig	zulässig				
PDF/X-4	zulässig	zulässig	zulässig	zulässig		
PDF/X-4p	zulässig	zulässig	zulässig	zulässig[1]		
PDF/X-5g	zulässig	zulässig	zulässig	zulässig	zulässig[2]	zulässig[2]
PDF/X-5pg	zulässig	zulässig	zulässig	zulässig[1]	zulässig[2]	zulässig[1,2]

1. PDF_process_pdi( ) mit `action=copyoutputintent` *kopiert die Referenz zur externen ICC-Druckausgabebedingung.*
2. Bei importierten Seiten mit referenzierten XObjects kopiert PDF_open_pdi_page( ) sowohl Proxy als auch Referenz.

stellten PDF/X-Konformitätsstufe nicht genügen, werden von *PDF_open_pdi_document( )* abgelehnt.

Werden mehrere PDF/X-Dokumente importiert, müssen alle für dieselbe Druckausgabebedingung erzeugt worden sein.

PDFlib kann bestimmte Punkte zwar korrigieren, ist aber nicht für die Überprüfung und Durchsetzung vollständiger PDF/X-Kompatibilität importierter Dokumente gedacht. PDFlib bettet in importierten PDF-Seiten zum Beispiel keine fehlenden Fonts ein und nimmt auch keine Farbkorrekturen vor.

Um importierte PDF-Seiten zu einem Ausgabedokument zusammenzustellen, das derselben PDF/X-Konformitätsstufe und Druckausgabebedingung wie die Eingabedokumente genügt, können Sie den PDF/X-Status eines importierten PDFs wie folgt abfragen:

```
pdfxlevel = p.pcos_get_string(doc, "pdfx");
```

Sofern das importierte Dokument konform zu einem PDF/X-Level ist, gibt diese Anweisung einen String mit der entsprechenden PDF/X-Konformitätsstufe aus, andernfalls *none*. Mit dem zurückgegebenen String kann mit der Option *pdfx* in *PDF_begin_document( )* die entsprechende PDF/X-Konformitätsstufe für das Ausgabedokument gesetzt werden.

Kopieren der PDF/X-Druckausgabebedingung eines importierten Dokuments. Neben der Abfrage der PDF/X-Konformitätsstufe können Sie die PDF/X-Druckausgabebedingung aus einem importierten Dokument kopieren.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `pdfx/import_pdfx`.

Diese Methode kann als Alternative zur Festlegung der Druckausgabebedingung mit *PDF_load_iccprofile( )* verwendet werden. Hierbei wird die Druckausgabebedingung des importierten Dokuments in das generierte Ausgabedokument kopiert. Das Kopieren der Ausgabebedingung funktioniert für importierte PDF/A- und PDF/X-Dokumente.

11.5 PDF/VT für variablen und Transaktionsdruck

Hinweis Allgemeine Informationen zum PDF/VT-Standard finden Sie auf der PDFlib-Website.

11.5.1 Der PDF/VT-Standard

Der PDF/VT-Standard wurde im Jahr 2010 als ISO 16612-2 veröffentlicht. Der Standard ist »dazu entworfen, Variable Document Printing (VDP) in einer Vielzahl von Umgebungen zu ermöglichen«. PDF/VT-Dokumente enthalten die finalen Inhaltselemente mit den entsprechenden Metadaten, aber keine Variablen oder Templates. Der Vorgängerstandard ISO 16612-1:2005 spezifizierte das PPML/VDX-Format, das auf PDF 1.4 basiert. Diesem älteren Standard fehlten jedoch einige PDF-Funktionen und er beruhte auf Konstrukten außerhalb von PDF. Der moderne Standard PDF/VT basiert auf den Standards PDF/X-4 und PDF/X-5 und unterstützt den Funktionsumfang von PDF 1.6 einschließlich Transparenz und Ebenen. Zusätzlich zu den Anforderungen von PDF/X steuert der PDF/VT-Standard weitere Funktionen bei, um den Anforderungen an personalisierten Massendruck zu entsprechen. PDF/VT erlaubt schnelles Rastern (Rendering, *RIPping)* von digitalen Druckdateien, indem es das PDF-Format um effizientes Ressourcen-Management ergänzt.

PDF/VT-1-, PDF/VT-2- und PDF/VT-2s-Konformitätsstufen. In ISO 16612-2 sind drei PDF/VT-Konformitätsstufen spezifiziert, die alle auf PDF 1.6 basieren:

- PDF/VT-1 wurde für den Austausch abgeschlossener Einzeldateien konzipiert und basiert auf PDF/X-4 (PDF/X-4p ist unzulässig). Eine PDF/VT-1-Datei enthält alle erforderlichen Ressourcen für die Wiedergabe des Dokuments.
- PDF/VT-2 ist für den Austausch mehrerer Dateien konzipiert und basiert jeweils auf PDF/X-4p, PDF/X-5g oder PDF/X-5pg. PDF/VT-2-Dokumente können externe ICC-Profile, externe Seiteninhalte oder beides referenzieren. Ein PDF/VT-Dokument mit allen referenzierten PDF-Dateien und externen ICC-Profilen wird als PDF/VT-2-Fileset bezeichnet.
- PDF/VT-2s ermöglicht die Ausgabe von Streams, damit sich die Generierung des Dokuments und das Rendering überlappen können. Ein PDF/VT-2s-Stream ist ein MIME-Paket, das ein oder mehrere PDF/VT-1-Dateien oder PDF/VT-2-Filesets (oder beides) sowie alle referenzierten Dateien enthält.

Mit PDFlib lassen sich PDF/VT-1- und PDF/VT-2-Ausgabe sowie die Komponenten für einen PDF/VT-2s-Stream erzeugen. Das Paketieren von Komponenten als MIME-Stream wird von PDFlib nicht unterstützt und muss über die Anwendung implementiert werden. Für weitere Informationen hierzu siehe Abschnitt 11.5.8, »Erzeugung von MIME-Streams für PDF/VT-2s«, Seite 354.

PDF/VT-2-Dokumente in Acrobat. Beachten Sie beim Anzeigen von PDF/VT-2-Dokumenten in Acrobat Folgendes:

- Beim Öffnen von PDF/VT-2-Dokumenten mit referenzierten ICC-Druckausgabebedingungen kann sich Acrobat XI aufhängen oder abstürzen. Für weitere Informationen hierzu und eine Lösung des Problems siehe Abschnitt »Probleme bei Acrobat mit referenzierten ICC-Druckausgabebedingungen«, Seite 341. Für die Validierung solcher Dokumente mit dem Preflight-Werkzeug gilt die gleiche Lösung.
- Referenzierte Seiten erfordern die sorgfältige Konfiguration von Acrobat. Für weitere Informationen siehe Abschnitt »Einsatz referenzierter Seiten in Acrobat«, Seite 80.

11.5.2 Technische Konzepte von PDF/VT

Dieser Abschnitt bietet einen Überblick über die technischen Konzepte, auf denen PDF/VT beruht.

Document Part Hierarchy. Die Document Part Hierarchy (*DPart*) beschreibt die Abfolge und Beziehung von Dokumenten oder Teilen von Dokumenten in einer PDF/VT-Datei. Üblich ist, dass die PDF/VT-Datei aus Teildokumenten für viele Empfänger besteht und jeder Dokumentteil die Seiten für einen einzelnen Empfänger enthält. Die DPart-Hierarchie kann aber nicht nur Seiten den jeweiligen Empfängern zuordnen, sondern auch komplexere Strukturen abbilden. Beispielsweise könnten die Empfänger nach der Postleitzahl in ihrer Adresse, die Postleitzahlen wiederum nach Bundesland und die Bundesländer nach Staaten sortiert sein. Diese Art der Organisation eines Dokuments schafft eine Baumstruktur, die alle Seiten des Dokuments enthält. Die Elemente dieses Baums heißen DPart-Knoten, wobei jeder interne Knoten weitere DPart-Knoten enthält und jeder Blattknoten eine oder mehrere Seiten für einen Empfänger beschreibt.

Alternativ zum Zugriff über Seitenzahlen oder Seiten-Labels kann diese Hierarchie von Dokumentteilen in einer PDF/VT-Datei benutzt werden, um auf Seiten zuzugreifen. Die DPart-Hierarchie ist in einer PDF/VT-Datei zwingend erforderlich. Der optionale Eintrag *RecordLevel* in einem PDF/VT-Dokument beschreibt die Ebene in der DPart-Hierarchie, auf der sich die Datensätze für einen einzelnen Empfänger befinden. Dies ist für die Scope Hints relevant (siehe unten).

Document Part Metadata (DPM). Jeder Knoten in der Hierarchie der Dokumentteile von der Wurzel bis zu den Blättern im Dokumentbaum kann Document Part Metadata (DPM) enthalten. Solche Metadaten können Informationen über das Dokument eines einzelnen Empfängers und dessen Teile vermitteln. Insbesondere produktionsrelevante Informationen (z.B. Anzahl der Kopien für einen Dokumentteil) oder Informationen über den Empfänger (z.B. die zugehörige Postleitzahl) können in DPM abgelegt werden.

Der PDF/VT-Standard beschreibt die generellen Methoden zur Unterbringung der Document Part Metadata, schreibt jedoch weder Schema noch Kodierung der Metadaten vor. Allerdings wurde der Standard im Hinblick auf das Metadaten-Schema im Job Definition Format (JDF) entwickelt, ein von der *International Cooperation for the Integration of Processes in Prepress, Press, and Postpress Organization (CIP4)* standardisiertes Jobticket-Format. Weitere Informationen dazu finden Sie unter

```
www.cip4.org
```

JDF (oder andere) Produktions-Metadaten sind in PDF/VT nicht zwingend erforderlich, erweisen sich aber in JDF-fähigen Workflows als äußerst nützlich. Zusätzlich beschreibt der PDF/VT-Standard auch eine Methode zur Darstellung von DPM mit XML.

Optimierungen für wiederkehrende grafische Inhalte. Druckelemente werden oft auf vielen Seiten mehrfach genutzt, z.B. ein Firmenlogo oder ein Produktbild, das auf allen Seiten eines Mailings auftaucht. Die optimierte Verarbeitung dieser wiederkehrenden grafischen Inhalte stellt eine wichtige Strategie dar, um sowohl Dateigröße als auch Verarbeitungsgeschwindigkeit der Druckdatei zu optimieren. PDF unterstützt seit jeher das Konzept der XObjects, die die Dateigröße minimieren, indem sie die erforderlichen Daten eines Druckelements nur einmal in der Datei vorhalten. Diese Daten können dann von beliebig vielen Seiten (oder mehrfach auf einer Seite) referenziert werden. XObjects

können Rasterbilder, Text und Vektorgrafik enthalten. Während XObjects in PDF dafür sorgen, die Dateigröße zu optimieren, war es bei PDF bisher nicht möglich, die Verarbeitung von sich wiederholenden Seiteninhalten zu beschleunigen. Ein PDF-Dokument enthält keine Informationen darüber, dass zum Beispiel ein Bild auf einer bestimmten Seite auf einer weiteren Seite im selben oder einem anderen Dokument wiederholt wird. PDF/VT erweitert das existierende Konzept von XObjects in PDF um die folgenden Methoden zur Optimierung der Druckgeschwindigkeit:

- *Eindeutige Identifizierung:* XObjects können eine Identifizierung erhalten (genannt *GTS_XID),* die über alle Dokumente eindeutig ist. Diese Identifizierung kann von Cache-Implementierungen genutzt werden, die gleiche XObjects erkennen müssen. Einfach gesagt, die bereits in Job 1 genutzte Grafik, welche in Job 2 wieder auftaucht, muss nicht noch einmal gerastert werden, sondern die vorhandenen Rasterdaten können aus dem Cache übernommen werden.
- *Scope Hints und Umgebungskontext:* XObjects können Informationen darüber enthalten (genannt *GTS_Scope)*, die beschreiben, welche Seiten oder Dokumente die Inhalte erneut nutzen. Auf diese Weise tragen die XObjects Informationen über die sinnvolle Lebensdauer ihrer Rasterdarstellung im Cache: Wird der Inhalt nur für den aktuellen Empfänger gebraucht, an anderer Stelle in der selben Datei oder im selben Datenstrom wiederverwendet, oder vielleicht überhaupt nicht mehr benötigt? Wird der Umgebungskontext (genannt *GTS_Env)* angegeben, kann das XObject auch für die globale Nutzung vorgesehen sein, das heißt, es wird in mehr als einer PDF/VT-Instanz genutzt. Es gibt keine Einschränkungen, was der String zum Umgebungskontext enthalten kann. So kann zum Beispiel ein Kunden- oder Jobname zur Beschreibung des Kontextes angegeben werden.
- *Encapsulation Hints:* Caching-Algorithmen müssen das Zusammenspiel eines XObject mit dem aufrufenden Kontext und vorhandenen Druckelementen auf der selben Seite (oder anderen Seiten, wenn mehrere Seiten auf dem selben Bogen ausgegeben werden) einbeziehen. Wenn in einem XObject zum Beispiel die Farbe oder Linienstärke nicht festgelegt wird, sondern diese Eigenschaften anhand der in der Umgebung vorherrschenden Farb- und Linieneinstellungen variieren, dann ist das Zwischenspeichern der gerasterten Ergebnisse aufgrund des unterschiedlichen Erscheinungsbilds nutzlos. Eine ähnliche Situation tritt ein, wenn das XObject transparente Elemente enthält, so dass der vorhandene Hintergrund mit dem XObject zusammen berücksichtigt werden muss. Um das Caching von XObjects zu vereinfachen, führt PDF/VT das Konzept der Encapsulated XObjects ein, die als solche (mit Hilfe des Eintrags *GTS_Encapsulated)* markiert werden können. Encapsulated XObjects müssen bestimmte Regeln erfüllen, die das Caching erleichtern.

All diese Einträge sind optional: PDF/VT verlangt keine dieser Optimierungen für wiederkehrende grafische Inhalte. Wenn sie allerdings genutzt werden, ergeben sich erhebliche Geschwindigkeitsvorteile mit einem PDF/VT-fähigen RIP.

11.5.3 Zusammenfassung der Regeln für PDF/VT-1 und PDF/VT-2

Cookbook *Codebeispiele für die Generierung von PDF/VT-1, PDF/VT-2 und PDF/VT-2s finden Sie in der Kategorie* `pdfvt` *des PDFlib Cookbook.*

PDF/VT-1 und PDF/VT-2 lässt sich mit PDFlib folgendermaßen erzeugen:

- Da PDF/VT auf PDF/X basiert, müssen alle Anforderungen für die zugrunde liegenden PDF/X-Konformitätsstufe erfüllt werden. Für die Dokumentoptionen *pdfvt* und *pdfx* müssen passende Werte angegeben werden.
- Dokumentteile müssen mit den *DPart*-Funktionen festgelegt werden. Jeder Knoten in einer Document Part Hierarchy kann optional DPM-Metadaten tragen. Der Eintrag RecordLevel kann im DPart-Baum angegeben werden.
- Für wiederkehrende grafische Inhalte sollten Scope Hints übergeben werden.
- Enthält das Dokument Transparenz, sollten die Bedingungen für Encapsulated XObjects sorgfältig erfüllt werden.
- Beim Import von Seiten aus bestehenden PDF-Dokumenten müssen zusätzliche Regeln eingehalten werden; siehe Abschnitt 11.3.7, »Import von PDF/A-Dokumenten mit PDI«, Seite 332.

Tabelle 11.19 fasst die erforderlichen und optionalen Operationen für die Erzeugung von PDF/VT-konformer Ausgabe zusammen, die über die entsprechenden PDF/X-Anforderungen aus Abschnitt 11.4.2, »Allgemeine Anforderungen«, Seite 338 hinausgehen. Spezielle Aspekte werden in den folgenden Abschnitten weiter erläutert.

Tabelle 11.19 Erforderliche und optionale Operationen für die Konformität zu PDF/VT-1 und PDF/VT-2

Kriterium	***Für PDF/VT-Konformität erforderliche PDFlib-Funktionen und -Optionen***
Konformitätsstufe	*Die Option* `pdfvt` *von PDF_begin_document( ) muss auf die erforderliche PDF/VT-Konformitätsstufe gesetzt sein, z.B.* `pdfvt=PDF/VT-1`**.** *(PDF/VT-1) Die Option* `pdfx=PDF/X-4` *wird automatisch gesetzt; die Angabe aller anderen Werte für die Option* `pdfx` *führt zu einem Fehler.* *(PDF/VT-2) Die Option* `pdfx` *muss ebenfalls angegeben werden, und zwar mit einem der Werte* `PDF/X-4p`, `PDF/X-5g` *oder* `PDF/X-5pg`.
Document Part Hierarchy	*Die Document Part Hierarchy muss angegeben werden. Dies umfasst die folgenden Operationen:* *PDF_begin_document( ): Die Option* `nodenamelist` *muss die Namen aller Ebenen des DPart-Baums enthalten. Mit der Option* `recordlevel` *lässt sich die Ebene des DPart-Baums angeben, die den Datensätzen für einzelne Empfänger entspricht.* *PDF_begin_dpart( ) und PDF_end_dpart( ): Die Document Part Hierarchy muss aufgebaut werden.* *Dokumentteile können optional Document Part Metadata (DPM) enthalten. DPM lässt sich mit der POCA-Schnittstelle durch den Aufbau eines DPM-Dictionarys erzeugen.*
Scope Hints für wiederkehrende grafischen Inhalte	*PDF_load_image( ), PDF_open_pdi_page( ), PDF_begin_template_ext( ) und Option* `templateoptions` *von PDF_load_graphics( ): Die Unteroption* `scope` *sollte an die Option* `pdfvt` *übergeben werden, und zwar mit einem der Werte* `unknown`, `singleuse`, `record`, `file`, `stream` *oder* `global`, *um Benutzerhinweise für wiederkehrende Bilder, Templates und importierte PDF-Seiten bereitzustellen.* *Für* `scope=stream` *und* `scope=global` *sollte die Unteroption* `environment` *einen Umgebungskontext für PDF/VT enthalten, das heißt, einen Identifier, den ein konformer PDF/VT-Reader zur Bereitstellung einer Schnittstelle zum Verwalten von verwandten XObjects verwenden kann. Beispielsweise könnte der Kunden- oder Jobname zur Identifikation der Umgebung verwendet werden.*

Tabelle 11.19 Erforderliche und optionale Operationen für die Konformität zu PDF/VT-1 und PDF/VT-2

Kriterium	*Für PDF/VT-Konformität erforderliche PDFlib-Funktionen und -Optionen*
Externer grafischer Inhalt (referenziert)	*(PDF/VT-1 und PDF/VT2, basierend auf PDF/X-4p) Die Option* `reference` *von PDF_begin_template_ext( ), PDF_load_graphics( ) und PDF_open_pdi_page( ) ist nicht zulässig, da sie vom zugrundeliegenden PDF/X-Standard nicht unterstützt wird.* *(PDF/VT-2, basierend auf PDF/X-5g oder PDF/X-5pg) Das Ziel, das in der Option* `reference` *von PDF_begin_template_ext( ), PDF_load_graphics( ) oder PDF_open_pdi_page( ) angegeben wird, muss einem der folgenden Standards genügen: PDF/X-1a:2003, PDF/X-3:2003, PDF/X-4 oder PDF/X-4p. Außerdem muss es für die gleiche Druckausgabebedingung aufbereitet worden sein. Beachten Sie, dass PDF/X-5g, PDF/X-5pg und PDF/VT-2 keine zulässigen Ziele sind; diese Einschränkung verhindert Ketten von referenzierten Dokumenten.* *Für weitere Informationen zu der Option* `reference` *und der erforderlichen Acrobat-Konfiguration siehe Abschnitt 3.2.5, »Seiten aus externen PDF-Dokumenten referenzieren«, Seite 80.*
Encapsulated XObjects	*PDF_begin_document( ): Wenn das Dokument keine Transparenzelemente enthält, sollte die Option* `usestransparency=false` *übergeben werden, damit PDFlib alle XObjects, die aus Templates erzeugt werden, als encapsulated kennzeichnen kann.* *PDF_load_image( ): Bilder sollte mit den Optionen* `renderingintent` *oder* `mask` *verwendet werden. Ansonsten sollten die Regeln aus Abschnitt 11.5.6, »Encapsulated XObjects«, Seite 352 eingehalten werden, um so viele XObjects wie möglich als eingeschlossen zu kennzeichnen.*

Strategien zum Einsatz von Farbe in PDF/VT. Für die Farbverarbeitung bestehen folgende Strategien, die auch kombiniert zum Einsatz kommen können:

- Geräteunabhängiger Farbraum: unabhängig von der ICC-Druckausgabebedingung können geräteunabhängige Farbräume verwendet werden, das heißt ICC-basiert oder CIELab. Der *Lab*-Farbwert *(0, 0, 0)* definiert reines Schwarz in geräteunabhängiger Art. Um die *Lab*-Farbe schwarz manuell einzustellen, verwenden Sie folgenden Aufruf:

```
p.set_graphics_option("fillcolor={lab 0 0 0}");
```

- Geräteabhängiger Farbraum: geräteabhängige Graustufen, RGB- oder CMYK-Farbräume können verwendet werden. Während Graustufen-Farbräume für jegliche Art von Druckausgabebedingung verwendet werden können, können RGB- oder CMYK-Farbräume nur mit einer passenden Druckausgabebedingung verwendet werden.

Zusätzlich können Schmuckfarben mit entsprechendem alternativen Farbraum verwendet werden. Da PDFlib als alternativen Farbraum für die integrierten HKS- und Pantone-Schmuckfarben CIELab verwendet, können diese bei PDF/VT immer verwendet werden. Bei benutzerdefinierten Schmuckfarben muss der alternative Farbraum so gewählt werden, dass er zur Druckausgabebedingung konform ist.

Die Zusammenfassung der Strategien zum Einsatz von Farbe in Tabelle 11.20 kann bei der Planung von PDF/VT-Anwendungen hilfreich sein.

Tabelle 11.20 Strategien zum Einsatz von Farbe für die Konformität zu PDF/VT

	Farbräume, die im Dokument verwendbar sind					
Druckausgabebedingung	***CIELab***	***ICCBased***	***Pantone, HKS***	***Grayscale[1]***	***RGB[1]***	***CMYK[1]***
Graustufen-Profil	*ja*	*ja*	*ja*	*ja*	*–*	*–*
RGB-Profil[2]	*ja*	*ja*	*ja*	*ja*	*ja*	*–*
CMYK-Profil	*ja*	*ja*	*ja*	*ja*	*–*	*ja*

1. Geräteunabhängiger Farbraum ohne ICC-Profil
2. Der PDF/X-Standard benötigt ein Druckerprofil; Monitorprofile, z.B. sRGB, sind unzulässig. RGB-Druckerprofile sind sehr selten.

Kombinierte PDF/VT- und PDF/A-Dokumente. Es kann sinnvoll sein, PDF/VT-Druckdokumente zu erzeugen, die gleichzeitig konform zu PDF/A für die Archivierung sind (für weitere Informationen siehe Abschnitt 11.3, »PDF/A zur Archivierung«, Seite 325). Beachten sie dabei Folgendes:

- Da bei PDF/A keine externen Verweise zulässig sind, lässt sich PDF/A nur mit PDF/VT-1, aber nicht mit PDF/VT-2 kombinieren. Mit den folgenden Dokumentoptionen lässt sich PDF/VT und PDF/A gleichzeitig erzeugen:

```
ret = p.begin_document("combo.pdf", "pdfa=PDF/A-2b pdfx=PDF/X-4");
```

- Die in diesem Handbuch aufgeführten Anforderungen für PDF/VT und PDF/A müssen erfüllt werden, und es können nur Funktionen verwendet werden, die von beiden Standards unterstützt werden.
- Nur importierte PDF-Dokumente, die sowohl dem PDF/X- als auch dem PDF/A-Standard entsprechen, sind zulässig.

11.5.4 Document Part Hierarchy und Document Part Metadata (DPM)

Cookbook *Codebeispiele für die Generierung von PDF/VT mit einer Document Part Hierarchy finden Sie in der Kategorie* `pdfvt` *des PDFlib Cookbook.*

Erzeugen der Document Part Hierarchy. Alle Seiten eines Dokuments müssen in der Document Part Hierarchy aufgeführt sein. Bei einfachen Fällen, wie zum Beispiel Rechnungen, besteht die Baumstruktur aus zwei Ebenen, den Ebenen *root* und *recipient*. Die Seiten im Dokument bilden eine lineare Anordnung der Datensätze für die Empfänger, wobei jeder Record eine oder mehrere Seiten enthält. Die Struktur der Document Part Hierarchy muss in der Optionsliste für das Dokument angegeben werden, zum Beispiel:

```
if (p.begin_document(outfile,
        "pdfvt=PDF/VT-1 nodenamelist={root recipient} recordlevel=1") == -1)
```

Komplexere Dokumente benötigen eventuell eine tiefere Dokumenthierarchie, zum Beispiel Broschüren mit Vorder- und Rückseite und einem Hauptteil, die auf der Ebene *docpart* verwaltet werden:

```
if (p.begin_document(outfile,
        "pdfvt=PDF/VT-1 nodenamelist={root recipient docpart} recordlevel=1") == -1)
```

Mit der Option *recordlevel* lässt sich ein auf Null basierender Index in der *nodenamelist* definieren, auf dem sich die Empfänger- oder Record-Ebene befindet, was für die Option *scope=record* relevant ist.

Nach der Festlegung der Strukturhierarchie müssen die Seiten mit *PDF_begin/end_dpart( )* zu Dokumenthierarchie-Knoten gruppiert werden:

```
/* Erzeugen des root-Knotens für die DPart-Hierarchie */
p.begin_dpart("");
        p.begin_dpart("");          /* Erzeugen eines neuen Knotens auf Empfängerebene */
            p.begin_page_ext(...);  /* Erzeugen der Seiten */
            ...
            p.end_page_ext(...);
        p.end_dpart("");            /* Beenden des Empfängerknotens */

        p.begin_dpart("");          /* Erzeugen des nächsten Knotens auf Empfängerebene
*/
            p.begin_page_ext(...);  /* Erzeugen der Seiten */
            ...
            p.end_page_ext(...);
        p.end_dpart("");            /* Beenden des Empfängerknotens */
/* Beenden des root-Knotens */
p.end_dpart("");
```

Mit den Aufrufen von *PDF_begin/end_dpart( )* muss eine Baumstruktur gemäß der Option *nodenamelist* erzeugt werden, das heißt, die größte Verschachtelungstiefe muss der Anzahl der Einträge im Array *nodenamelist* entsprechen.

Die Optionen *groups* von *PDF_begin_document( )* sowie *group* und *pagenumber* von *PDF_begin/end_page_ext( )* sind im PDF/VT-Modus nicht zulässig, da sie mit der DPart-Hierarchie in Konflikt geraten würden.

Erzeugen der Document Part Metadata (DPM). Für jeden Knoten in der Document Part Hierarchy können Metadaten übergeben werden, die sich entweder auf die entsprechenden Seiten eines Blattknotens oder den gesamten Unterbaum unter diesem Knoten beziehen. Obwohl PDF/VT keine bestimmten Metadaten vorschreibt, ist es für die Verwendung mit dem JDF-Standard ausgerichtet, der von der CIP4-Organisation veröffentlicht wurde. Vor allem das Dokument »ICS — Common Metadata for Document Production Workflows« (über die CIP4- Website abrufbar) beschreibt die Verwendung von Metadaten bei der Druckproduktion.

Das CIP4-Metadatenformat verwendet gängige PDF-Datentypen, und die Metadaten werden so in die PDF-Ausgabe geschrieben, wie es im PDF/VT-Standard vorgeschrieben ist.

PDFlib-Benutzer können mit der POCA-Schnittstelle *(PDF Object Creation API)* beliebige Datenstrukturen mit PDF-Dictionaries, Arrays oder anderen Datentypen erzeugen. Das Top-Level-Dictionary für die DPM-Metadaten kann an die Option *dpm* von *PDF_begin/end_dpart( )* übergeben werden:

```
dpm           = p.poca_new("type=dict usage=dpm");
cip4_root     = p.poca_new("type=dict usage=dpm");
cip4_metadata = p.poca_new("type=dict usage=dpm");

p.poca_insert(dpm,           "type=dict   key=CIP4_Root value=" + cip4_root);
p.poca_insert(cip4_root,     "type=dict   key=CIP4_Metadata value=" + cip4_metadata);
p.poca_insert(cip4_metadata, "type=string key=CIP4_Conformance value=base");
p.poca_insert(cip4_metadata, "type=string key=CIP4_Creator value=starter_pdfvt1");
```

```
p.poca_insert(cip4_metadata, "type=string key=CIP4_JobID value={Kraxi Systems invoice}");

/* Erzeugen des Knotens in der DPart-Hierarchie und Hinzufügen der DPM-Metadaten */
p.begin_dpart("dpm=" + dpm);

p.poca_delete(dpm, "recursive=true");
```

Cookbook *Das pCOS Cookbook von PDFlib enthält ein Codebeispiel für die Abfrage der DPM aus einem PDF/VT-Dokument und die Erzeugung der entsprechenden XML-Repräsentation.*

11.5.5 Scope Hints für wiederkehrende grafische Inhalte

Scope Hints. Die Unteroption *scope* der Option *pdfvt* sollte mit geeigneten Werten an *PDF_load_image( ), PDF_load_graphics( ), PDF_open_pdi_page( )* und *PDF_begin_template_ext( )* übergeben werden. Dazu muss der Client-Anwendung bekannt sein, wo und wie oft Bilder, Seiten und Templates im aktuellen PDF-Dokument oder besser noch über mehrere Dokumente hinweg verwendet werden. Obwohl die Option *scope* nicht zwingend erforderlich ist, sollte sie unbedingt angegeben werden, da sie wichtige Informationen zur Optimierung des RIP enthält. Ohne diese Informationen sollte die Option *scope* besser weggelassen oder mit dem Wert *unknown* übergeben werden, um die Angabe falscher Werte zu vermeiden.

PDFlib prüft für die Scopes *singleuse*, *record* und *file*, ob der Scope-Wert konsistent zur aktuellen Verwendung des XObjects ist und gibt eine Warnung in der Log-Datei aus, wenn der Scope zu groß ist, zum Beispiel:

```
XObject with handle 9 was assigned PDF/VT scope 'record', but was used only once
(use 'scope=singleuse' instead)
```

Wenn Sie eine solche Warnung erhalten, sollten Sie überprüfen, ob Sie zur Steigerung der Druckleistung den jeweiligen Rasterbildern, Grafiken, importierten PDF-Seiten oder Templates besser geeignete Scope-Werte zuweisen können, um unnötiges Caching im RIP zu vermeiden.

Während zu wenig Verweise auf ein XObject (bezogen auf die Option *scope)* nur den Hinweis auf eine eventuelle Optimierung des Client-Codes auslösen, führen zu häufige Verweise auf ein XObject tatsächlich oft zu einem Fehler, da dies den PDF/VT-Standard verletzen würde. In folgenden Fällen wird daher eine Exception ausgelöst:

- Die Option *scope=singleuse* wurde übergeben und das XObject wird mehr als einmal im Dokument verwendet.
- Die Option *scope=record* wurde übergeben und das XObject wird in mehr als einem Empfänger-Datensatz verwendet.

Für *scope=stream* und *scope=global* muss die Unteroption *environment* übergeben werden. Sie sollte einen geeigneten String zur Identifizierung gecachter Objekte im Dokument enthalten. Für diese Option kann je nach Aufgabenstellung eine Kunden- oder Jobreferenz verwendet werden. Beispielsweise lässt sich ein in mehreren Druckaufträgen häufig auftretendes Firmenlogo so identifizieren.

Eindeutige (Unique) IDs. PDFlib weist allen importierten PDF-Seiten, Rasterbildern und Grafiken automatisch eindeutige IDs zu; bei Grafiken muss die Option *templateoptions* angegeben werden. Um das Caching im RIP zu optimieren, werden gleichen Rasterbildern, Seiten und Grafiken identische ID-Werte zugewiesen. Mit der Unter-

option *xid* der Option *pdfvt* lassen sich eindeutige IDs für Templates vergeben. IDs für Templates sollten für Template-Definitionen identisch sein, die gleichwertige PDF-Form-XObjects gemäß PDF/VT erzeugen (das heißt, Templates mit gleicher visueller Ausgabe). Für unterschiedliche Templates sollten unterschiedliche oder gar keine IDs vergeben werden.

Wenn die Anwendung die von PDFlib erzeugten eindeutigen ID-Strings benötigt, zum Beispiel für DPM-Metadaten, lassen sich diese mit der Option *xid* von *PDF_info_image( )* für Bilder und Templates und von *PDF_info_graphics( )* für Grafiken und von *PDF_info_pdi_page( )* für importierte PDF-Seiten abfragen.

11.5.6 Encapsulated XObjects

Encapsulated XObjects können die Verarbeitungsleistung erheblich verbessern. XObjects lassen sich jedoch nur in Dokumenten ohne Transparenz oder in Dokumenten mit Transparenz als encapsulated markieren, wenn bestimmte Voraussetzungen erfüllt sind.

Transparenz im Dokument. Da PDFlib nicht im Voraus wissen kann, ob Transparenz im Dokument verwendet wird, kann als Hinweis die Option *usestransparency* an *PDF_begin_document( )* übergeben werden. Wenn der Client mit dieser Option angibt, das keine Transparenz verwendet wird, werden alle XObjects als encapsulated gekennzeichnet. Wenn im Dokument Transparenz verwendet wird, muss der Client bestimmte Regeln einhalten, damit XObjects als encapsulated ausgezeichnet werden. Wenn *usestransparency=false* angegeben wird, das Dokument aber trotzdem Transparenzelemente enthält, löst PDFlib eine Exception aus.

Transparenz gilt im generierten Dokument als verwendet, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

- *PDF_load_image( )* oder *PDF_fill_imageblock( )* wird mit einer der folgenden Optionen aufgerufen:
  - das Bild enthält einen Alphakanal und die Option *ignoremask* ist auf *false* gesetzt;
  - die Option *masked* mit einem Handle zu einem Bild mit mehr als 1 Bit pro Pixel.
- Mit *PDF_load_graphics( )* importierte Grafiken enthalten Transparenzelemente.
- *PDF_create_gstate( )* wird mit einer der folgenden Optionen aufgerufen:
  - die Option *softmask* mit einer Optionsliste (das heißt, nicht mit dem Schlüsselwort *none);*
  - eine der Optionen *opacityfill* oder *opacitystroke* mit einem von 1 verschiedenen Wert;
  - die Option *blendmode* mit einem anderen Wert als *None* oder *Normal*.
- Mit *PDF_open_pdi_page( )* oder *PDF_fill_pdfblock( )* importierte PDF-Seiten enthalten Transparenzelemente. Beachten Sie, das sich Transparenz in importierten PDF-Seiten mit der pCOS-Schnittstelle und dem Pseudo-Objekt *usespagetransparency* abfragen lässt (für weitere Informationen siehe die pCOS-Pfadreferenz).

Kennzeichnung von XObjects als encapsulated. PDFlib versucht anhand folgender Regeln, so viele XObjects wie möglich als encapsulated zu kennzeichnen:

- Mit *PDF_load_image( )* oder *PDF_fill_imageblock( )* erzeugte XObjects werden als encapsulated ausgezeichnet, wenn eine der folgenden Bedingungen erfüllt ist:
  - die Option *renderingintent* von *PDF_load_image( )* wurde mit einem von *Auto* verschiedenen Wert übergeben;

- die Option *mask* von *PDF_load_image( )* wurde mit dem Wert *true* übergeben.
- Mit der Option *templateoptions* von *PDF_begin_template_ext( )* oder *PDF_load_graphics( )* erzeugte XObjects werden als encapsulated ausgezeichnet, wenn die Dokumentoption *usestransparency=false* übergeben wurde oder die Option *transparencygroup* mit der Unteroption *isolated=true* übergeben wurde.
- Wenn eine importierte Seite ein bereits als encapsulated markiertes XObject enthält, wird diese Eigenschaft beim Import der Seite beibehalten.

Wenn ein XObject nicht als encapsulated gekennzeichnet werden kann, wird eine Warnung in der Log-Datei ausgegeben.

Option scope für encapsulated XObjects. Wir empfehlen dringend, die Unteroption *scope* der Option *pdfvt* bei allen encapsulated XObjects anzugeben, die mit *PDF_load_image( ), PDF_fill_imageblock( ), PDF_load_graphics( )* und *PDF_begin_template_ext( )* erstellt werden. Da die Unteroption *scope* auch bei nicht encapsulated XObjects angegeben werden kann, kann sie bei allen relevanten API-Aufrufen angegeben werden, wenn die Lebenszeit eines Rasterbildes, einer Grafik, einer importierten Seite oder eines Templates bekannt ist, unabhängig davon, ob das generierte XObject encapsulated ist oder nicht.

11.5.7 Import von PDF/X- und PDF/VT-Dokumenten mit PDI

Wenn Seiten eines vorhandenen PDF-Dokuments in ein PDF/VT-konformes Ausgabedokument importiert werden, gelten spezielle Regeln (für weitere Informationen siehe Abschnitt 7.3, »Import von PDF-Seiten mit PDI«, Seite 202). Alle importierten Dokumente müssen einem PDF/VT-Level genügen, der zum generierten Ausgabedokument kompatibel ist (siehe Tabelle 11.21). Wird in PDFlib eine bestimmte PDF/VT-Konformitätsstufe eingestellt ist und die importierten Dokumente einem kompatiblen Standard genügen, dann ist die erzeugte Ausgabe garantiert konform zum eingestellten Level. Importierte Dokumente, die der eingestellten PDF/VT-Konformitätsstufe nicht genügen, werden abgelehnt.

Tabelle 11.21 Zulässige PDF/X- und PDF/VT-Eingabestufen für verschiedene PDF/VT-Ausgabestufen

	PDF/X- und PDF/VT-Konformitätsstufe des importierten Dokuments			
PDF/VT-Ausgabestufe	**PDF/X-1a:2003 PDF/X-3:2003 PDF/X-4 PDF/VT-1**	**PDF/X-4p PDF/VT-2 basierend auf PDF/X-4p**	**PDF/X-5g PDF/VT-2 basierend auf PDF/X-5g**	**PDF/X-5pg PDF/VT-2 basierend auf PDF/X-5pg**
PDF/VT-1 (immer auf PDF/X-4 basierend)	zulässig	zulässig		
PDF/VT-2 basierend auf PDF/X-4p	zulässig	zulässig[1]		
PDF/VT-2 basierend auf PDF/X-5g	zulässig	zulässig	zulässig[2]	zulässig[2]
PDF/VT-2 basierend auf PDF/X-5pg	zulässig	zulässig[1]	zulässig[2]	zulässig[1,2]

1. PDF_process_pdi( ) mit `action=copyoutputintent` *kopiert die Referenz zur externen ICC-Druckausgabebedingung.*
2. Bei importierten Seiten mit referenzierten XObjects kopiert PDF_open_pdi_page( ) sowohl Proxy als auch Referenz.

Einschränkungen für DPart und DPM beim PDF/VT-Import. Beachten Sie beim Import von PDF/VT-Dokumenten folgende Einschränkungen: Document Part Hierarchy (DPart) und Document Part Metadata (DPM) werden nicht importiert. Sie lassen sich über die pCOS-Schnittstelle abfragen und mit *PDF_begin/end_dpart( )* und POCA-Funktionen wiederherstellen.

11.5.8 Erzeugung von MIME-Streams für PDF/VT-2s

Cookbook Ein Codebeispiel für die Generierung von PDF/VT-2s finden Sie in der Kategorie `pdfvt` *des PDFlib Cookbook.*

PDF/VT-2s beschreibt einen Datenstrom, der eine Reihe von PDF/VT-Bestandteilen, das heißt, eine oder mehrere PDF/VT-1- oder PDF/VT-2-Dateien sowie alle referenzierten Dateien enthält. Das Paketieren der Datenströme in MIME-Streams[1] wird von PDFlib nicht unterstützt und muss von der Anwendung implementiert werden. Ein PDF/VT-2s-Stream kann sich aus folgenden Teilen zusammensetzen:

- eine oder mehrere PDF/VT-1-Dateien, PDF/VT-2-Filesets oder beide Varianten, also die direkten (nicht referenzierten) Dokumente;
- PDF-Dokumente in den folgenden Formaten, die von PDF/VT-2-konformen Dokumenten referenziert werden: PDF/X-1a, PDF/X-3, PDF/X-4, PDF/X-4p, PDF/VT-1, wenn der Stream eine oder mehrere PDF/VT-2-Dateien enthält, die auf PDF/X-5g oder PDF/X-5pg basieren.
- Externe ICC-Profile für Druckausgabebedingungen, wenn der Stream eine oder mehrere PDF/VT-2-Dateien enthält, die auf PDF/X-4p oder PDF/X-5pg basieren.

PDF/VT-2s ist nicht als Speicherformat im Dateisystem, sondern nur für die Übertragung vorgesehen. Zur Speicherung von Inhalten im Dateisystem sollte ein Satz einzelner PDF/VT-Dokumente erzeugt werden.

Anforderungen bei der PDF-Erzeugung. Bei der Erzeugung von PDF-Dokumenten für PDF/VT-2s müssen folgende Anforderungen erfüllt werden. Da PDFlib Statusinformationen nicht über Dokumentgrenzen hinweg speichert, ist die Anwendung für die Einhaltung dieser Regeln verantwortlich:

- Die Druckausgabebedingung muss für alle PDF/VT-Dokumente in einem PDF/VT-2s-Stream identisch sein, das heißt, ICC-Profile, die mit *PDF_load_iccprofile( )* und *usage=outputintent* oder mit *PDF_process_pdi( )* und *action=copyoutputintent* geladen werden, müssen für alle Dokumente identisch sein.
- Die Option *nodenamelist* von *PDF_begin_document( )* muss in allen direkten (also nicht referenzierten) PDF/VT-Dateien in einem PDF/VT-2s-Stream den gleichen Wert haben. Ebenso muss die Option *recordlevel* (wenn vorhanden) in allen direkten PDF/VT-Dateien in dem Stream den gleichen Wert haben. Beachten Sie, dass der PDF/VT-Standard auch für referenzierte PDF/VT-Dateien identische Werte bei *nodenamelist* und *recordlevel* erwartet, aber dies scheint ein Versehen zu sein.
- Die Unteroptionen *scope=stream* oder *scope=global* der Option *pdfvt* sollten (sofern sinnvoll) an *PDF_load_image( ), PDF_open_pdi_page( )* und *PDF_begin_template_ext( )* übergeben werden, zusammen mit einem geeigneten Wert für die Unteroption *environment*.

1. Die MIME-Spezifikation finden Sie unter `www.ietf.org/rfc/rfc2045.txt`

- Wir empfehlen dringend, die Option *xid* an *PDF_begin_template_ext( )* zu übergeben, wenn *scope=stream* oder *scope=global* ist, um das Caching von XObjects über Dokumente hinweg zu ermöglichen.

Anforderungen bei der Erzeugung von MIME-Streams. Bei der Erzeugung von MIME-Streams für PDF/VT-2s müssen die in Tabelle 11.22 aufgeführten Anforderungen erfüllt werden.

Tabelle 11.22 Anforderungen bezüglich MIME für PDF/VT-2s-Streams

Einheit	*Content-Typ Header*	*weitere Header und zusätzliche Anforderungen*
gesamter PDF/VT-2s-Stream	`multipart/mixed`	`X-PDFVT-Stream-version: 1` *muss vor dem Header* `Content-Type` *stehen*
referenzierte PDF-Dateien	`application/pdf`	`Content-Disposition: attachment; filename=...` `filename`-*Parameter erforderlich; in Referenzen zu verwenden*
externe ICC-Profile	`application/vnd.iccprofile`	`Content-Disposition: attachment; filename=...` `filename`-*Parameter erforderlich; in Referenzen zu verwenden*
direkte PDF/VT-Dateien	`application/pdf`	`Content-Disposition: inline` *müssen nach allen referenzierten PDF-Dateien und externen ICC-Dateien im MIME-Stream kodiert sein; sie dürfen nicht von einer PDF/VT-2-Datei im Stream referenziert werden*
alle MIME-Bestandteile		`Content-Transfer-Encoding: binary` *empfohlen*

11.6 PDF/UA für Barrierefreiheit

11.6.1 Der PDF/UA-Standard

PDF/UA-1 verbessert die Barrierefreiheit von PDF-Dokumenten durch Angabe von Elementen des Tagged PDF und anderen Dokumentaspekten. PDF/UA kann als Anwendung von WCAG 2.0 *(Web Content Accessibility Guidelines*[1]*)* auf PDF-Dokumente betrachtet werden. Wenn Sie WCAG-2.0-konforme Dokumente benötigen, können Sie PDF/UA verwenden, um dieses Ziel zu erreichen. Wenn Sie allerdings planen, Multimedia, Scripting oder Aktionen in Ihre generierten Dokumente zu integrieren, müssen Sie zusätzlich zu PDF/UA auch WCAG 2.0 berücksichtigen.

PDF/UA basiert auf Tagged PDF, wie es in PDF 1.7 und ISO 32000-1 definiert ist. Es fügt dem PDF-Dateiformat keine neuen Funktionen hinzu, aber schreibt einige in PDF 1.7 optionalen Aspekte der Zugänglichkeit und des Taggings verbindlich vor. Darüber hinaus wird die Beziehung der unterschiedlichen Typen von Strukturelementen zueinander klargestellt.

PDF/UA-1, definiert in ISO 14289-1. PDF/UA verbessert die Zugänglichkeit von PDF-Dokumenten mit folgenden Mitteln:

- Einhaltung bestimmter Anforderungen in Bezug auf die Dokumentstruktur, das heißt, Regeln für das Anbringen von Tags;
- Angabe bestimmter Hilfsinformationen wie Metadaten und Alternativtext für Grafiken;
- Vermeidung bestimmter PDF-Elemente, die die Zugänglichkeit einschränken.

PDFlib unterstützt PDF/UA-1 auf Basis des folgenden Dokuments:

- PDF/UA-1-Standard (ISO 14289-1:2012)

Erzeugung kombinierter PDF/UA- und PDF/A-Dokumente. Es kann sinnvoll sein, PDF/UA-Dokumente zu erzeugen, die gleichzeitig konform zu PDF/A für die Archivierung sind (für weitere Informationen siehe Abschnitt 11.3, »PDF/A zur Archivierung«, Seite 325). Tatsächlich empfehlen wir, wenn Sie PDF/A-1a/2a/3a erstellen möchten, die PDF/UA-Anforderungen zu erfüllen, um die Zugänglichkeit der erstellten Dokumente zu verbessern. Zur Erzeugung kombinierter PDF/A- und PDF/UA-Dokumente übergeben Sie geeignete Werte für die Optionen *pdfa* und *pdfua* von *PDF_begin_document( )*, zum Beispiel:

```
ret = p.begin_document("combo.pdf", "pdfa=PDF/A-2a pdfua=PDF/UA-1");
```

Beachten Sie bei solchen kombinierten Dateien, dass diese die Anforderungen für beide Standards erfüllen müssen. Die PDF-Kompatibilitätsstufe ist das Minimum an PDF-Kompatibilität der betroffenen Standards; importierte PDF-Dokumente müssen sowohl den PDF/UA- als auch den PDF/A-Standard einhalten.

Wir empfehlen, PDF/A-1a für Tagged PDF zu vermeiden und stattdessen mit den neueren Standards PDF/A-2a oder PDF/A-3a zu arbeiten, da es sonst zu einigen kleineren Konflikten zwischen PDF/UA-a und PDF/A-1a kommen kann:

1. Siehe www.w3.org/TR/WCAG20/

- Das Attribut *Scope* ist in PDF/UA erforderlich, aber in PDF 1.4, worauf PDF/A-1a basiert, nicht verfügbar. In kombinierten Dokumenten können für Tabellen daher keine ordnungsgemäßen Tags für Überschriftenzellen vergeben werden. Dies betrifft sowohl das manuelle als auch das automatische Anbringen von Tabellen-Tags. Deshalb können in PDF/A-1a nur Tabellen ohne Überschriftenzellen ordnungsgemäß mit Tags versehen werden.
- Bei Anmerkungen auf einer Seite erwartet PDF/UA die Seitenoption *taborder=structure*. Allerdings benötigt die Option *taborder* PDF 1.5 und kann daher nicht in PDF/A-1a verwendet werden. Deshalb können in kombinierten PDF/A-1a- und PDF/UA-Dokumenten keine Anmerkungen verwendet werden.

Cookbook *Am Beispiel* `starter_invoice` *im PDFlib Cookbook wird gezeigt, wie man ein kombiniertes PDF/UA-1- und PDF/A-2a-Dokument erstellt.*

Entdeckt PDFlib eine Regelverletzung bei der PDF/UA-Erstellung, so wird eine Exception ausgelöst. In diesem Fall wird keine PDF-Ausgabe erzeugt. Tabelle 11.23 zeigt die allgemeinen Anforderungen, die zur Erzeugung von PDF/UA-konformer Ausgabe erfüllt sein müssen.

Tabelle 11.23 Allgemeine Anforderungen für die Konformität zu PDF/UA

Kriterium	***Für PDF/UA-Konformität erforderliche PDFlib-Funktionen und -Optionen***
allgemeine Dokumentanforderungen	*PDF_begin/end_document( ): die Option* `pdfua` *muss auf* `PDF/UA-1` *gesetzt werden, was Tagged PDF erfordert (die Option* `tagged` *wird automatisch gesetzt). Operationen, die PDF 1.7ext3 oder höher erfordern (z.B. Rich-Media-Anmerkungen, Portfolios) sind unzulässig.* *Option* `viewerpreferences`*: für die Unteroption* `displaydoctitle` *ist nur* `true` *zulässig* *Option* `permissions`*: das Schlüsselwort* `noaccessible` *ist unzulässig*
Tagged PDF	*Es müssen alle Regeln für Tagged PDF eingehalten werden (siehe Abschnitt »Verschachtelungsregeln für Strukturelemente«, Seite 288). Die Dokumentoption* `checktags` *darf nicht auf* `false` *gesetzt werden.*
Fonts	*Die Fontoption* `embedding` *muss auf* `true` *gesetzt werden. Die Optionen* `unicodemap=false` *und* `dropcorewidths=true` *sind unzulässig. Auch die PDF-Standardfonts müssen eingebettet werden.*
Textausgabe und PUA-Unicode-Zeichen	*PUA-Unicode-Zeichen (z.B. Logos und Symbole) benötigen einen geeigneten Ersatztext, der in der Option* `ActualText` *von PDF_begin_item( ) für das Inhaltselement oder in der entsprechenden Option* `tag` *der jeweiligen Ausgabefunktion übergeben wird (siehe »PUA-Zeichen«, Seite 331).*
unsichtbarer Text	*Die einzige Ausnahme zur Anforderung, Fonts einzubetten, bezieht sich auf Fonts, die ausschließlich in unsichtbarem Text verwendet werden (hauptsächlich bei OCR-Ergebnissen relevant), das heißt,* `textrendering=3`*. Dies lässt sich mit der Option* `optimizeinvisible` *steuern.* *Wenn unsichtbarer Text kein darstellbares Äquivalent hat (z.B. ein gescanntes Bild), dann muss er als* `Artifact` *ausgezeichnet werden.*
Ebenen	*Ebenen sind zulässig, aber einige Optionen von PDF_define_layer( ) und PDF_set_layer_dependency( ) sind unzulässig (siehe PDFlib-Referenz).*
externer Inhalt	*PDF_begin_template_ext( ), PDF_load_graphics( ) und PDF_open_pdi_page( ): die Option* `reference` *ist nicht zulässig.*
PDF-Import	*PDF_open_pdi_document( ): importierte Dokumente müssen PDF/UA-konform sein; siehe Abschnitt 11.6.4, »Import von PDF/UA-Dokumenten mit PDI«, Seite 361*
Metadaten	*PDF_set_info( ) mit* `key=Title` *oder* `metadata` *in PDF_begin/end_document( ) mit* `dc:title` *darf im übergebenen XMP keinen leeren Wert haben.*

11.6.2 Anforderungen bezüglich der Tags

Da PDF/UA auf den Anforderungen für Tagged PDF basiert, müssen die in Abschnitt 10.3, »Grundlagen von Tagged PDF«, Seite 282 beschriebenen Anforderungen erfüllt werden. Um die Barrierefreiheit zu verbessern, enthält PDF/UA allerdings eine Reihe von zusätzlichen Anforderungen für das Anbringen von Tags.

Semantische Anforderungen. Benutzer müssen eine Dokumenthierarchie erstellen und die unten aufgeführten semantischen Regeln einhalten. Die Auswahl geeigneter Strukturelemente ist ein entscheidender Faktor bei der Herstellung der PDF/UA-Konformität. Es ist wichtig zu verstehen, dass für die Einhaltung dieser Regeln die Anwendung verantwortlich ist, da PDFlib sie nicht überprüfen kann:

- Beim Anbringen von Tags müssen die zur Dokumentstruktur passenden Strukturelemente verwendet werden: Eine Überschrift muss als solche ausgezeichnet werden. Eine Tabelle muss als Tabelle ausgezeichnet werden.
- Semantisch nicht relevante Inhalte dürfen nicht in die Dokumenthierarchie aufgenommen, sondern müssen als *Artifact* gekennzeichnet werden.
- Strukturelemente müssen in der logischen Lesereihenfolge angeordnet sein. Dies lässt sich am einfachsten erreichen, wenn die Tags in Lesereihenfolge erstellt werden. Bei komplexen Layouts lässt sich dies auch mit *PDF_activate_item( )* bewerkstelligen (siehe Abschnitt 10.4.4, »Erzeugung von Seiteninhalt in abweichender Reihenfolge«, Seite 308).
- Wenn Informationen aufgrund von Farbe, Format oder Layout nicht anders zugänglich sind, muss der Inhalt entsprechend ausgezeichnet sein.
- Text in einer Grafik verlangt die Option *Alt* mit einer Erklärung, falls der Text keine natürliche Sprache ist (z.B. bei Font- oder Skript-Beispielen).
- Bildunterschriften müssen mit dem Tag *Caption* versehen sein.
- Wenn der Inhalt als Liste gelesen werden soll, müssen Listenelemente *(L)* erstellt werden.
- Für Links muss das zugehörige Element *Link* hinterlegt sein.
- Kopf-und Fußzeilen müssen als *Artifact* mit *artifacttype=Pagination* und *artifactsubtype=Header* oder *Footer* ausgezeichnet werden.
- Für logisch zusammenhängende Gruppen von Grafikelementen darf nur ein einziges Tag *Figure* vergeben werden.
- Fußnoten, Endnoten, Labels für Anmerkungen sowie Verweise auf Stellen im Dokument müssen entsprechend als *Note* oder *Reference* ausgezeichnet werden.

Tag-spezifische Anforderungen. In Tabelle 11.24 werden die Anforderungen für die Konformität zu PDF/UA bei bestimmten Tags aufgeführt. Diese Regeln müssen auch bei benutzerdefinierten Elementtypen eingehalten werden, für die es eine Rollenzuordnung auf die aufgeführten Standardtypen gibt. Wenn beispielsweise das benutzerdefinierte Tag *Illustration* in der Option *rolemap* eine Rollenzuordnung auf *Figure* hat, muss es auch die Bedingungen für *Figure* erfüllen.

Tabelle 11.24 Tag-spezifische Anforderungen für die Konformität zu PDF/UA

Elementtyp	Anforderungen für die Konformität zu PDF/UA
Standard-Elementtypen	*PDF_begin_document( ): die Option* `rolemap` *darf keine Rollenzuordnungen für Standard-Elementtypen enthalten.*
Figure	*Eine der Optionen* `Alt` *oder* `ActualText` *muss übergeben werden.*
Formula	*Die Option* `Alt` *muss übergeben werden.*
Table	*Tabellenelemente dürfen nur für logische Tabellen erstellt werden, aber nicht für Tabellen, die nur zu Layout-Zwecken erstellt wurden. Von PDFlib formatierte Tabellen können automatisch mit Tags versehen werden, siehe Abschnitt 10.4.1, »Automatisches Erstellen von Tabellen-Tags«, Seite 300.*
TH	*Tabellen sollten Kopfzeilen enthalten. Für* `TH`*-Elemente ist die Option* `Scope` *erforderlich (beim automatischen Erstellen von Tabellen-Tags ist dies gewährleistet, wenn die Option* `header` *von PDF_fit_table( ) verwendet wird).*
L	*Der Elementtyp* `L` *(list) erwartet die Option* `ListNumbering`*. Wenn keines seiner Unterelemente* `LI` *(Listeneintrag) ein* `Lbl`*-Element (label) enthält, muss* `ListNumbering` *auf* `None` *gesetzt werden. Wenn zwar* `ListNumbering=None`*, aber dennoch sichtbare Listen-Labels vorhanden sind, sollten diese als Artefakte ausgezeichnet werden.*
Note	*Für Fuß- und Endnoten, die als* `Note` *ausgezeichnet sind, muss die Option* `id` *übergeben werden.*

Überschriften. Alle Überschriften müssen mit den entsprechenden *Heading*-Tags ausgezeichnet werden. Für die Verschachtelung von Überschriften in einem PDF-Dokument gibt es zwei Ansätze:

- Stark strukturierte Dokumente: Gruppierungselemente werden so tief wie nötig verschachtelt, um die Gliederung des Inhalts in Artikel, Abschnitte, Unterabschnitte usw. wiederzugeben. Auf jeder Ebene sollten die Unterelemente des Gruppierungselements aus einer Überschrift *H*, einem oder mehreren Absätzen *P* für den Inhalt auf dieser Ebene und zusätzlichen Gruppierungselementen für untergeordnete Abschnitte bestehen. Starke Strukturierung ist typisch für XML-Dokumente.
- Schwach strukturierte Dokumente: die Strukturhierarchie des Dokuments ist relativ flach, mit nur ein bis zwei Ebenen von Gruppierungselementen, wobei Überschriften, Abschnitte und andere BLSEs unmittelbar auf der Ebene darunter angeordnet sind. Die logische Struktur spiegelt nicht die Gliederung des Inhalts wider, kann aber durch geeignete Überschriften-Ebenen *H1, H2, H3* ausgedrückt werden. *Heading*-Tags können keine Unterelemente haben. Schwache Strukturierung wird in der Regel bei HTML verwendet.

Im PDF/UA-Modus muss die obige Unterscheidung mit der Option *structuretype* von *PDF_begin_document( )* explizit angegeben werden. Abhängig von der Art der Dokumentstruktur gelten in PDFlib für die Verwendung von Heading-Elementen die folgenden Regeln.

- Alle Dokumente:
  - Die Option *Title* sollte in allen *Heading*-Tags zur Kennzeichnung von Kapiteln/Abschnitten des Dokument angegeben werden (zum Beispiel »Kapitel 1«)
  - Überschriftenelemente *H, H1, H2,* usw. dürfen keine Unterelemente haben.

- Schwach strukturierte Dokumente *(structuretype=weak):*
  - Sequenzen von Überschriften müssen bei *H1* beginnen und dürfen keine Ebene überspringen. Die Sequenz *H1 H3* ist zum Beispiel unzulässig.
  - *H7, H8* usw. dürfen bei mehr als sechs Überschriften-Ebenen verwendet werden.
  - Unnummerierte Überschriftenelemente *H* sind unzulässig.
- Stark strukturierte Dokumente *(structuretype=strong):*
  - *H* ist bei Überschriften erforderlich, jedoch darf es pro Knoten nur ein *H*-Tag in der Strukturhierarchie geben.
  - Nummerierte Überschriftenelemente *H1, H2,* usw. sind unzulässig.

11.6.3 Zusätzliche Anforderungen für bestimmte Inhaltstypen

Tabelle 11.25 führt die PDF/UA-Anforderungen und -Empfehlungen für verschiedene Inhaltstypen und interaktive Elemente auf.

Tabelle 11.25 PDF/UA-Anforderungen und -Empfehlungen für spezifische Inhaltstypen und interaktive Elemente

Inhaltstyp	***Für PDF/UA-Konformität erforderliche PDFlib-Funktionen und -Optionen***
Text	*Die natürliche Sprache des Textes auf der Seite muss angegeben werden; Sprachwechsel innerhalb einer Textsequenz müssen ebenfalls angegeben werden. Die natürliche Sprache kann mit der Option* `lang` *von PDF_begin_document( ) und anderen Mitteln angegeben werden; für weitere Informationen siehe Abschnitt »Angabe der Sprache«, Seite 293. Das Sprachattribut eines Elements wird an all seine untergeordneten Elemente vererbt.* *Die Fontoption* `replacementchar` *wird auf den Wert* `error` *gezwungen, um Glyphen vom Typ* `.notdef` *in der Ausgabe zu vermeiden.*
Vektorgrafik und Rasterbilder	*Rasterbilder und Vektorgrafik müssen als* `Artifact`, `Figure` *oder* `Formula` *ausgezeichnet werden.* *Dies bezieht sich auf niedrige Funktionen für Pfadangaben wie PDF_rect( ) usw., Pfadobjekte mit PDF_draw_path( ), PDF_fit_image( ) und ähnliche Funktionen. Diese Anforderung gilt auch für SVG-Grafik, die Vektorgrafik oder Rasterbilder enthält.*
importierte PDF-Seiten	*Mit PDF_fit_pdi_page( ) platzierte PDF-Seiten mit einer Grafik sollten als* `Artifact` *oder* `Figure` *ausgezeichnet werden.*
Anmerkungen	*Bei Anmerkungen auf der Seite: PDF_begin/end_page_ext( ): für die Option* `taborder` *ist nur der Wert* `structure` *erlaubt.* *PDF_create_annotation( ) für sichtbare Anmerkungen mit* `type=Link`*:* ► *Die Option* `contents` *ist erforderlich.* ► *Die Option* `ismap` *von PDF_create_action( ) ist unzulässig für Aktionen in Link-Anmerkungen.* *PDF_create_annotation( ) für sichtbare Anmerkungen mit einem von* `Link` *verschiedenen Typ:*[1] ► *Ein Strukturelement muss erstellt und in die Dokumenthierarchie in logischer Lesereihenfolge eingefügt werden.* ► *Die Option* `contents` *oder die Option* `tag` *mit der Unteroption* `ActualText` *ist erforderlich.*
Formularfelder	*Bei Formularfeldern auf der Seite: PDF_begin/end_page_ext( ): nur der Wert* `structure` *ist für die Option* `taborder` *zulässig.* *PDF_create_field( ) und die Optionen* `fieldname`, `fieldtype` *von PDF_add_table_cell( ): ein* `Form`*-Tag muss mit PDF_create_field( ) oder der Option* `tag` *erstellt werden. Die Option* `tooltip` *von PDF_create_field( ) und PDF_create_fieldgroup( ) ist erforderlich.*

Tabelle 11.25 PDF/UA-Anforderungen und -Empfehlungen für spezifische Inhaltstypen und interaktive Elemente

Inhaltstyp	Für PDF/UA-Konformität erforderliche PDFlib-Funktionen und -Optionen
Seiten-Label	*Mit der Option* `labels` *von PDF_begin/end_document( ) und der Option* `label` *von PDF_begin/end_page_ext( ) erstellte Seiten-Label sollten semantisch korrekt sein.*
Lesezeichen	*Wir empfehlen, Lesezeichen mit PDF_create_bookmark( ) zu erstellen. Die Lesezeichen sollten die logische Lesereihenfolge und die Verschachtelungstiefe des Inhalts widerspiegeln.*
Dateianhänge	*PDF_load_asset( ): die Option* `description` *wird empfohlen. Dateianhänge sollten selbst auch zugänglich sein.*

1. Eine Anmerkung wird als sichtbar behandelt, wenn ihr Rechteck zumindest teilweise innerhalb der CropBox der Seite liegt und die Option `display` *von PDF_create_annotation( ) von* `hidden` *und* `noview` *verschieden ist.*

11.6.4 Import von PDF/UA-Dokumenten mit PDI

Wenn Seiten eines vorhandenen PDF-Dokuments in ein PDF/UA-konformes Ausgabedokument importiert werden, gelten spezielle Regeln (für weitere Informationen siehe Abschnitt 7.3, »Import von PDF-Seiten mit PDI«, Seite 202). Alle importierten Dokumente und Seiten müssen bezüglich folgender Kriterien kompatibel zum aktuellen Dokument sein, andernfalls werden sie abgelehnt:

- *PDF_open_pdi_document( ):* nur PDF/UA-Dokumente können importiert werden und die Option *usetags* muss auf *true* gesetzt werden. Dokumente, die gemäß der Standardidentifikation in den XMP-Metadaten des Dokuments nicht PDF/UA-konform sind, werden von *PDF_open_pdi_document( )* abgelehnt.
- *PDF_open_pdi_page( ):* die Rollenzuordnung des importierten Dokuments muss kompatibel sein mit der Zuordnung, die in der Option *rolemap* von *PDF_begin_document( )* angegeben ist. Das bedeutet, dass benutzerdefinierte Elementtypen von der Option *rolemap* und der Rollenzuordnung des gerade oder früher importierten Dokuments nicht auf unterschiedliche Standardtypen zugeordnet werden können.
- *PDF_open_pdi_page( ):* die Struktur der Überschriften der importierten Seite muss kompatibel sein mit dem Strukturtyp des generierten Dokuments, das heißt, bei *structuretype=weak* dürfen nur *H1*, *H2*, usw. (aber nicht *H)* auf der Seite verwendet werden; bei *structuretype=strong* dürfen nur *H* (aber nicht *H1*, *H2*, usw.) auf der importierten Seite verwendet werden. Seiten mit sowohl nummerierten als auch unnummerierten *Heading*-Tags werden abgelehnt.

Wird in PDFlib PDF/UA-Konformität eingestellt und genügen die importierten Dokumente dieser Einstellung, dann gilt dies garantiert auch für die erzeugte Ausgabe.

Hinweis *PDFlib kann PDF-Dokumente nicht bezüglich PDF/UA-Konformität validieren und ist auch nicht in der Lage, aus beliebigen PDF-Dokumenten zulässige PDF/UA-Dokumente zu erstellen.*

12 PPS und das PDFlib Block-Plugin

Der PDFlib Personalization Server (PPS) bietet einen PDF-Workflow zur Verarbeitung variabler Daten auf Basis von Templates. Mit Hilfe des Blockkonzepts können importierte Seiten mit variablen Mengen von ein- oder mehrzeiligen Texten, Rasterbildern, PDF-Seiten oder Vektorgrafiken angereichert werden. Damit lassen sich auf einfache Art Anwendungen implementieren, die individualisierte PDF-Dokumente erfordern, zum Beispiel:

- Serienbriefe
- flexible Erstellung personalisierter Massenbriefe
- Berichtswesen und Rechnungserstellung
- Personalisierung von Visitenkarten

Mit dem PDFlib Block-Plugin können Sie interaktiv Blöcke erstellen und bearbeiten und mit dem Plugin zur Konvertierung von Formularfeldern PDF-Formularfelder in PDFlib-Blöcke konvertieren. Blöcke können mit PPS gefüllt werden. Die Ergebnisse der Block-Befüllung mit PPS können Sie in einer Vorschau in Acrobat ansehen, da das Block-Plugin eine integrierte Version des PPS enthält.

Hinweis *Zur Blockverarbeitung ist der PDFlib Personalization Server (PPS) erforderlich. PPS ist zwar in allen PDFlib-Paketen enthalten, Sie müssen dafür aber einen eigenen Lizenzschlüssel erwerben; ein Lizenzschlüssel für PDFlib oder PDFlib+PDI reicht nicht aus. Außerdem ist das PDFlib Block-Plugin für Adobe Acrobat erforderlich, um Blöcke in PDF-Templates interaktiv anzulegen.*

Cookbook *Codebeispiele zur Verarbeitung variabler Daten mit Blöcken finden Sie in der Kategorie* `blocks` *im PDFlib Cookbook.*

12.1 Installation des Block-Plugins

Das Block-Plugin funktioniert mit folgenden Versionen von Acrobat:

- Windows: Acrobat 8/9/X/XI Standard, Professional und Pro Extended
- OS X: Acrobat 8/9/X/XI Professional

Das Block-Plugin funktioniert nicht mit Acrobat Elements oder Acrobat Reader.

Installation des Block-Plugins für Acrobat 8/9/X/XI unter Windows. Zur Installation des Block-Plugins sowie des Plugins zur Konvertierung von PDF-Formularfeldern in Acrobat kopieren Sie die Plugin-Dateien in ein Unterverzeichnis des Acrobat-Plugin-Verzeichnisses. Dies wird von der Installationsroutine des Plugins automatisch durchgeführt, kann aber auch manuell erfolgen. Unter Windows heißen die Dateien *Block.api* und *AcroFormConversion.api*. Das Verzeichnis für die PDFlib-Plugins lautet üblicherweise in etwa:

```
C:\Programme\Adobe\Acrobat 11.0\Acrobat\plug_ins\PDFlib Block Plugin
```

Installation des Block-Plugins für Acrobat 8/9/X/XI unter OS X. Bei Acrobat ist der Plugin-Ordner im Finder nicht sichtbar. Statt die Plugin-Dateien in den Plugin-Ordner zu ziehen, gehen Sie wie folgt vor (beenden Sie Acrobat, falls geöffnet):

- Durch Doppelklick auf das Disk Image extrahieren Sie die Dateien für das PDFlib Block-Plugin in einen Ordner.

- Suchen Sie im Finder nach dem Symbol für die *Adobe Acrobat*-Anwendung. Normalerweise befindet es sich in einem Verzeichnis, das in etwa folgendermaßen lautet:

  ```
  /Programme/Adobe Acrobat 11.0
  ```

- Klicken Sie auf das Symbol der Acrobat-Anwendung, öffnen Sie das Kontextmenü und selektieren Sie *Paketinhalt anzeigen*.
- Navigieren Sie im dann erscheinenden Finder-Fenster zum Verzeichnis *Inhalt/Plugins* und kopieren Sie den im ersten Schritt angelegten Ordner *PDFlib Block Plugin* in dieses Verzeichnis.

Mehrsprachige Benutzeroberfläche. Das PDFlib Block-Plugin unterstützt mehrere Sprachen in der Benutzeroberfläche. Die Sprache für das Block-Plugin wird automatisch eingestellt und richtet sich nach der Sprache der Acrobat-Benutzeroberfläche. Derzeit stehen Englisch, Deutsch und Japanisch zur Verfügung. Wenn Acrobat in einer anderen Sprache läuft, wird die englische Benutzeroberfläche für das Block-Plugin gewählt.

Fehlerbehebung. Falls das PDFlib Block-Plugin nicht wie erwartet funktioniert, überprüfen Sie Folgendes:

- Stellen Sie sicher, dass unter *Bearbeiten, Voreinstellungen, [Allgemein...], Allgemein* das Kontrollkästchen *Nur zertifizierte Zusatzmodule verwenden* deaktiviert ist. Die Plugins werden nicht geladen, wenn Acrobat im zertifizierten Modus ausgeführt wird.
- Das Block-Plugin und auch andere Acrobat-Plugins funktionieren manchmal nicht korrekt bei PDF-Formularen, die mit Adobe Designer erstellt wurden, da sie mit dem internen Sicherheitsmodell von Acrobat kollidieren. Sie sollten die statischen PDF-Formulare von Adobe Designer vermeiden und als Input für das Block-Plugin nur dynamische PDF-Formulare nutzen.

12.2 Überblick über das Blockkonzept

12.2.1 Trennung von Dokumentdesign und Programmcode

Mit PDFlib-Blöcken ist es auf einfache Art möglich, variablen Text, Rasterbilder, PDF-Seiten oder Vektorgrafiken auf importierte Seiten zu platzieren. Im Gegensatz zu einfachen PDF-Seiten enthalten Seiten mit Blöcken Informationen über die Art der Verarbeitung, die später auf der Serverseite stattfindet. Das Blockkonzept von PDFlib trennt die folgenden Aufgaben voneinander:

- Der Designer erstellt das Seitenlayout und legt dabei die Position von variablem Text und Bildelementen sowie deren Eigenschaften wie Schriftgröße, Farbe oder Bildskalierung fest. Nach der Erstellung des Layouts als PDF-Dokument legt der Designer mit dem Block-Plugin für Acrobat die variablen Blöcke und ihre Eigenschaften fest.
- Der Programmierer schreibt den Code, der die Informationen in den PDF-Blöcken auf den importierten PDF-Seiten mit dynamischen Daten wie zum Beispiel Datenbankfeldern verknüpft. Der Programmierer benötigt keine genaueren Kenntnisse über einen Block (ob dieser einen Namen oder eine Postleitzahl enthält, wo genau er sich auf der Seite befindet oder wie er formatiert ist usw.). Er ist deshalb von Layout-Änderungen unabhängig. PPS kümmert sich um alle blockspezifischen Details, die aus den in der Datei abgelegten Blockeigenschaften ermittelt werden.

Anders ausgedrückt ist der vom Programmierer entwickelte Code »datenblind«, das heißt, er ist generisch und hängt nicht von blockspezifischen Eigenschaften ab. Der Designer kann zum Beispiel den Block mit dem Namen des Empfängers auf der Seite verschieben oder die Schriftgröße verändern. Der generische Code zur Blockverarbeitung braucht dann nicht geändert zu werden. Die Ausgabe wird korrekt generiert, sobald der Designer die Blockeigenschaften mit dem Acrobat-Plugin entsprechend geändert hat.

Als Zwischenschritt kann das Füllen der Blöcke in einer Vorschau in Acrobat angezeigt werden, um Entwicklungs-und Testzyklen zu beschleunigen. Die Block-Vorschau basiert auf Voreinstellungen (z. B. Text oder ein Bilddateiname), die in den Blockdefinitionen angegeben sind.

12.2.2 Blockeigenschaften

Das Verhalten von Blöcken lässt sich über die Blockeigenschaften steuern. Diese werden einem Block mit dem Block-Plugin zugeordnet.

Vordefinierte Blockeigenschaften. PDFlib-Blöcke sind als Rechtecke auf der Seite definiert, denen ein Name, ein Typ und eine offene Menge von Eigenschaften zugewiesen sind, die später vom PPS verarbeitet werden. Der Name zur Identifikation des Blocks kann beliebig gewählt werden, zum Beispiel *firstname*, *lastname* oder *zipcode*. PDFlib unterstützt unterschiedliche Blocktypen:

- Blöcke vom Typ *Textline* enthalten eine einzelne Textzeile, die mit der Textline-Ausgabemethode in PPS verarbeitet wird.
- Blöcke vom Typ *Textflow* enthalten eine oder mehrere Zeilen mit Textdaten. Mehrzeiliger Text wird mit dem Textflow-Formatierer in PPS formatiert. Blöcke lassen sich so verbinden, dass Text von einem Block in den nächsten fließt (siehe Abschnitt »Verbinden von Textflow-Blöcken«, Seite 386).

- Blöcke vom Typ *Image* enthalten ein Rasterbild. Dies entspricht dem Einfügen eines TIFF- oder JPEG-Bildes in einer DTP-Anwendung.
- Blöcke vom Typ *PDF* enthalten beliebige PDF-Inhalte, die aus einer Seite eines anderen PDF-Dokuments importiert wurden. Dies entspricht dem Einfügen einer PDF-Seite in einer DTP-Anwendung.
- Blöcke vom Typ *Graphics* enthalten Vektorgrafiken. Dies entspricht dem Einfügen einer SVG-Datei in einer DTP-Anwendung.

Blöcke haben je nach Typ bestimmte vordefinierte Eigenschaften. Blockeigenschaften können mit dem Block-Plugin erstellt und verändert werden (siehe Abschnitt 12.3.2, »Bearbeiten von Blockeigenschaften«, Seite 372). Eine Liste aller vordefinierten Blockeigenschaften finden Sie in Abschnitt 12.6, »Blockeigenschaften«, Seite 390. Für einen Textblock kann zum Beispiel die Schriftart und -größe des Texts und für einen Bild- oder PDF-Block der Skalierungsfaktor oder die Drehung festgelegt werden. Für jeden Blocktyp bietet PPS eine eigene Verarbeitungsfunktion, zum Beispiel *PDF_fill_textblock( )*. Anhand des Blocknamens suchen die Funktionen auf einer eingefügten PDF-Seite nach dem Block, analysieren seine Eigenschaften und platzieren die vom Client übergebenen Daten (ein- oder mehrzeiligen Text, Rasterbild, Vektorgrafik oder PDF-Seite) entsprechend der jeweiligen Blockeigenschaften auf der neuen Seite. Als Programmierer können Sie die Blockeigenschaften durch Angabe der entsprechenden Blockoptionen in den Block-Füllfunktionen überschreiben.

Eigenschaften für Vorgabewerte. Für die Vorgabewerte eines Blocks können spezielle Blockeigenschaften definiert werden. Dies sind Text-, Bild-, PDF- oder Grafik- Inhalte, die in dem Block platziert werden, wenn keine variablen Daten an die Block-Füllfunktionen geliefert werden oder wenn die Blockinhalte derzeit zwar konstant sind, sich in der nächsten Auflage aber ändern können.

Vorgabewerte werden auch von der Vorschaufunktion des Block-Plugins verwendet (siehe Abschnitt 12.4, »Block-Vorschau in Acrobat«, Seite 380).

Benutzerdefinierte Blockeigenschaften. Die vordefinierten Blockeigenschaften ermöglichen eine schnelle Implementierung von Anwendungen zur Verarbeitung variabler Daten. Designer sind damit jedoch auf die Eigenschaften beschränkt, die PPS intern kennt und automatisch verarbeitet. Um mehr Flexibilität zu bieten, wird deshalb zusätzlich die Möglichkeit geboten, einen Block mit selbst definierten Eigenschaften zu versehen. Durch diese Erweiterung wird das Blockkonzept selbst Anwendungen gerecht, die höhere Anforderungen an die Verarbeitung variabler Daten stellen.

Für selbst definierte Eigenschaften gibt es keinerlei Regeln, da sie von PPS in keiner Weise verarbeitet, sondern lediglich dem Client verfügbar gemacht werden, der sie inspizieren und auf geeignete Art darauf reagieren kann. Auf Basis der selbst definierten Eigenschaften eines Blocks könnte zum Beispiel über Layout oder Datenerfassung entschieden werden. So könnte für eine wissenschaftliche Anwendung festgelegt werden, mit wie vielen Stellen eine Zahl ausgegeben wird, oder die Blockeigenschaft könnte den Namen eines Datenbankfeldes definieren, dessen Inhalt dann zum Füllen des Blocks benutzt wird.

12.2.3 Was spricht gegen PDF-Formularfelder?

Der erfahrene Acrobat-Benutzer mag sich fragen, warum wir ein neues Blockkonzept implementiert haben, statt die bestehenden Formularfelder von PDF zu nutzen. Der entscheidende Unterschied besteht darin, dass PDF-Formularfelder in erster Linie dafür konzipiert wurden, vom Benutzer ausgefüllt zu werden, während PDFlib-Blöcke automatisch ausgefüllt werden. Anwendungen, die sowohl interaktives als auch automatisches Ausfüllen benötigen, können PDF-Formulare und PDFlib-Blöcken mit Hilfe des Plugins zur Konvertierung von Formularfeldern kombinieren (siehe Abschnitt 12.3.4, »Konvertieren von PDF-Formularfeldern in PDFlib-Blöcke«, Seite 375).

Wenngleich die beiden Konzepte in vielen Punkten übereinstimmen, bieten PDFlib-Blöcke gegenüber PDF-Formularfeldern doch einige Vorteile. Diese werden in Tabelle 12.1 aufgezeigt.

Tabelle 12.1 Vergleich zwischen PDF-Formularfeldern und PDFlib-Blöcken

Funktion	***PDF-Formularfelder***	***PDFlib-Blöcke***
Konzeption	*für interaktiven Gebrauch*	*für automatisches Ausfüllen*
typografische Funktionen (neben Schriftart und -größe)	*–*	*Unterschneiden, Wort- und Zeichenabstand, Unterstreichen/Überstreichen/Durchstreichen*
OpenType Layout-Funktionen	*–*	*viele OpenType Layout-Funktionen, z.B. Ligaturen, Swash-Zeichen, Mediävalziffern*
Unterstützung für komplexe Schriftsysteme	*eingeschränkt*	*Anpassung der Zeichenform und bidirektionale Formatierung, z.B. für Arabisch und Devanagari*
Font-Behandlung	*Font-Einbettung*	*Font-Einbettung, Einbettung von Untergruppen, Encoding*
Textformatierung	*linksbündig, rechtsbündig, mittig*	*linksbündig, rechtsbündig, mittig, Blocksatz; verschiedene Formatierungsalgorithmen und Steuermöglichkeiten; Inline-Optionen können zur Steuerung der Textdarstellung verwendet werden*
Wechsel des Fonts und anderer Textattribute innerhalb des Texts	*–*	*ja*
kombiniertes Ergebnis ist Bestandteil der PDF-Seitenbeschreibung	*–*	*ja*
Benutzer können kombinierte Feldinhalte editieren	*ja*	*nein*
Funktionalität erweiterbar	*–*	*ja (selbst definierte Blockeigenschaften)*
Verwendung von Bilddateien zum Füllen	*–*	*BMP, CCITT, GIF, PNG, JPEG, JBIG2, JPEG 2000, TIFF*
Verwendung von Vektorgrafik zum Füllen		*SVG*
Farbunterstützung	*RGB*	*Graustufen, RGB, CMYK, Lab, Schmuckfarbe (HKS- und PANTONE-Schmuckfarben im Block-Plugin integriert)*
PDF-Standards	*–*	*PDF/A, PDF/X, PDF/VT, PDF/UA*
Eigenschaften von Text und Grafik beim Füllen veränderbar	*–*	*ja*
Verbinden von Textblöcken möglich	*–*	*ja*

12.3 Bearbeiten von Blöcken mit dem Block-Plugin

12.3.1 Anlegen von Blöcken

Aktivieren des Blockwerkzeugs. Das Block-Plugin zur Erzeugung von PDFlib-Blöcken ist dem Acrobat-Formularwerkzeug recht äh?nlich. Solange das Blockwerkzeug ausgewählt ist, sind alle Blöcke auf der Seite sichtbar. Wenn Sie ein anderes Acrobat-Werkzeug auswählen, werden die Blöcke ausgeblendet, sind aber nach wie vor vorhanden. Sie können das Blockwerkzeug auf folgende Arten aktivieren:

- Klicken Sie auf das Blocksymbol  in der Leiste *Werkzeuge, Erweiterte Bearbeitung* (Acrobat X/XI) bzw. in der Werkzeugleiste *Erweiterte Bearbeitung* (Acrobat 9). Wenn Acrobat diese Leiste nicht anzeigt, können Sie sie über *Anzeige, Werkzeuge, Erweiterte Bearbeitung* (Acrobat X/XI) bzw. über *Anzeige, Werkzeugleiste, Erweiterte Bearbeitung* (Acrobat 9) einblenden.
- Wählen Sie den Menübefehl *PDFlib Block, PDFlib Block-Werkzeug*.

Anlegen und Ändern von Blöcken. Nach der Auswahl des Blockwerkzeugs legen Sie einen Block an, indem Sie mit dem Fadenkreuz-Zeiger einfach an der gewünschten Position auf der Seite ein Rechteck geeigneter Größe aufziehen. Blöcke sind immer Rechtecke, deren Kanten parallel zu den Seitenrändern verlaufen. (Verwenden Sie für Blockinhalte, die nicht parallel zu den Seitenrändern verlaufen, die Eigenschaft *rotate*.) Bei der Erstellung eines neuen Block-Rechtecks erscheint der Dialog *PDFlib Blockeigenschaften*, in dem Sie die Eigenschaften einstellen können (siehe Abschnitt 12.3.2, »Bearbeiten von Blockeigenschaften«, Seite 372). Das Block-Plugin erzeugt einen Blocknamen, den Sie im Dialog *PDFlib Blockeigenschaften* jederzeit ändern können. Der Blockname muss innerhalb einer Seite eindeutig sein, kann aber auf einer anderen Seite wiederverwendet werden.

Sie können den Blocktyp im oberen Bereich zum Typ *Textline, Textflow, Image, PDF* oder *Graphics* ändern. Für die Darstellung der Blocktypen werden verschiedene Farben verwendet (siehe Abbildung 12.1). Im Dialog *PDFlib Blockeigenschaften* sind die Eigenschaften je nach Blocktyp in Gruppen und Untergruppen gegliedert.

Hinweis *Benutzen Sie nach dem Hinzufügen neuer Blöcke oder Bearbeiten existierender Blöcke den Acrobat-Befehl »Speichern unter« (und nicht »Speichern«), um die Dateigröße zu minimieren.*

Hinweis *Wenn Sie mit dem Acrobat-Plugin Enfocus PitStop Dokumente bearbeiten, die PDFlib-Blöcke enthalten, erhalten Sie unter Umständen die Meldung »Dieses Dokument enthält anwendungsspezifische Informationen von PDFlib. Klicken Sie auf 'OK' zum Fortfahren oder 'Abbrechen' zum Beenden.« Sie brauchen dieser Meldung keine weitere Beachtung zu schenken und können problemlos OK drücken.*

Selektieren von Blöcken. Blockoperationen wie Kopieren, Verschieben, Bearbeiten oder Löschen beziehen sich auf einen oder mehrere selektierte Blöcke. Mit dem Blockwerkzeug lassen sich ein oder mehrere Blöcke wie folgt auswählen:

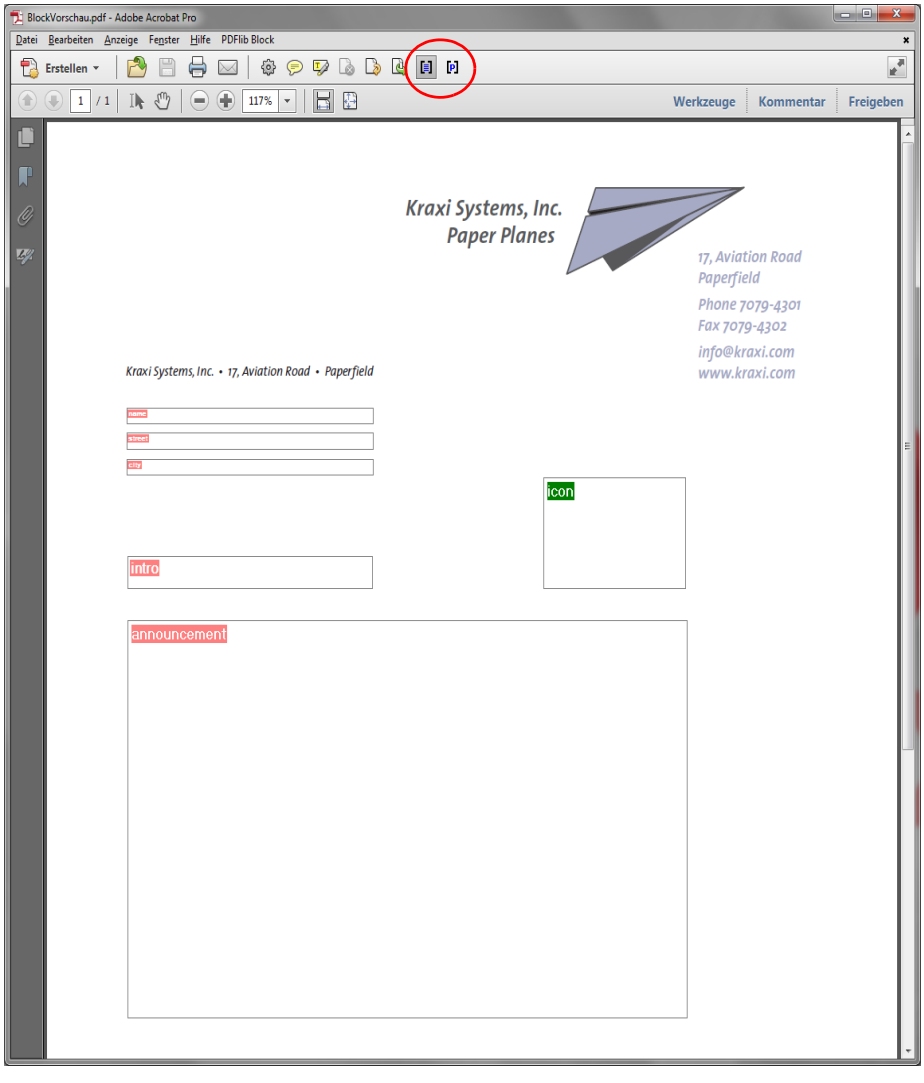
- Um einen einzelnen Block auszuwählen, klicken Sie einfach darauf.
- Um mehrere Blöcke auszuwählen, halten Sie die Shift-Taste gedrückt und klicken Sie dabei auf die gewünschten Blöcke.
- Um alle Blöcke auf der Seite auszuwählen, drücken Sie Strg-A (Windows) bzw. Cmd-A (OS X) oder *Bearbeiten, Alles auswählen*.

Das Kontextmenü. Sind einer oder mehrere Blöcke selektiert, können Sie das Kontextmenü zum schnellen Zugriff auf blockspezifische Funktionen nutzen (die sich auch im Menü PDFlib Block befinden). Zum Öffnen des Kontextmenüs klicken Sie mit der rechten Maustaste (Windows) oder klicken bei gedrückter Strg-Taste (OS X) auf den oder die selektierten Blöcke.

Um zum Beispiel einen Block zu löschen, selektieren Sie ihn mit dem Blockwerkzeug und drücken die *Entf*-Taste oder wählen Sie den Befehl *Bearbeiten, Löschen* im Kontextmenü.

Wenn Sie in einen Seitenbereich ohne Blöcke rechts-klicken (oder unter OS X bei gedrückter Strg-Taste klicken), können Sie eine Block-Vorschau erstellen oder die Vorschau-Funktion konfigurieren.

Abb. 12.1 Darstellung von Blöcken



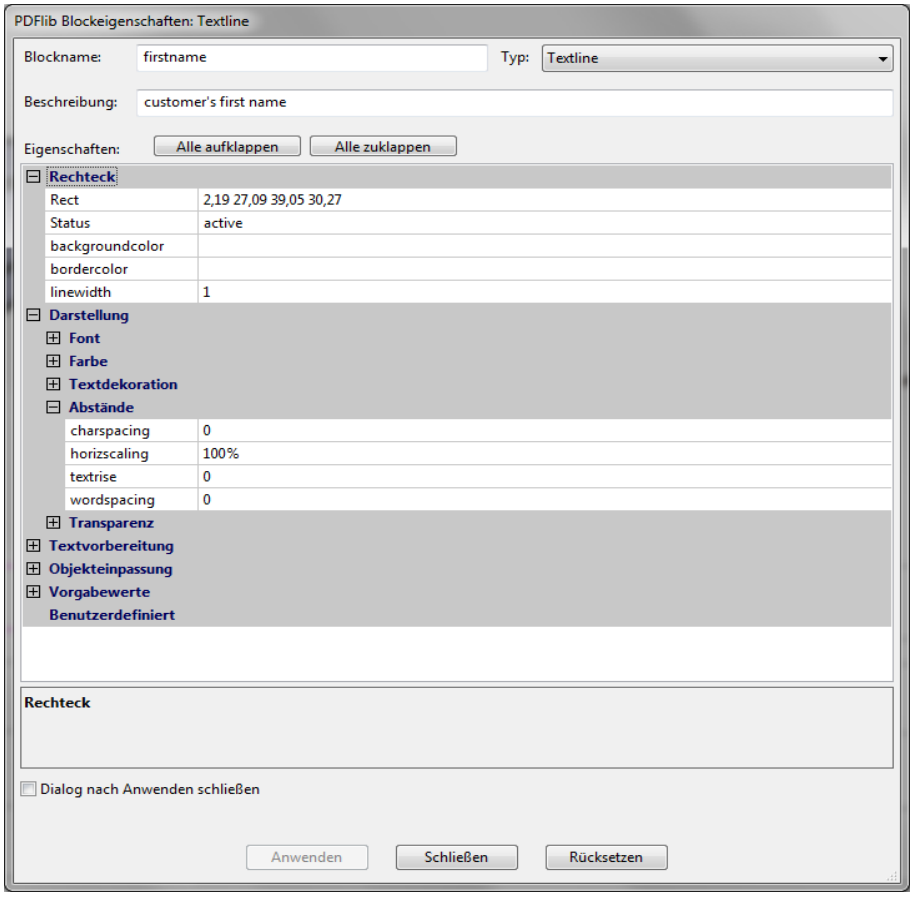


Abb. 12.2 Dialog PDFlib Blockeigenschaften

Blockgröße und -position. Mit dem Blockwerkzeug können Sie einen oder mehrere Blöcke an eine andere Position bewegen. Wenn Sie die Umschalttaste (Shift) während des Ziehens gedrückt halten, sind nur horizontale und vertikale Bewegungen möglich, was die exakte Ausrichtung von Blöcken erleichtert. Wenn Sie den Zeiger in die Nähe einer Blockecke bewegen, verwandelt er sich in einen Pfeil, mit dem Sie die Blockgröße ändern können. Um die Position oder Größe mehrerer Blöcke anzupassen, wählen Sie diese aus und wählen im Menü *PDFlib Block* oder im Kontextmenü die Befehle aus den Untermenüs *Ausrichten, Zentrieren, Verteilen* und *Skalieren*. Die Position eines oder mehrerer Blöcke lässt sich auch feinstufig mit den Pfeiltasten ändern.

Alternativ dazu können Sie im *Blockeigenschaften*-Dialog numerische Blockkoordinaten eingeben. Der Ursprung des Koordinatensystems liegt in der linken oberen Ecke der Seite. Die Koordinaten werden in der Einheit angezeigt, die in Acrobat gerade eingestellt ist:

- Zum Ändern der Einheit in Acrobat 9/X/XI gehen sie folgendermaßen vor: Wählen Sie den Menübefehl *Bearbeiten, Voreinstellungen, [Allgemein...], Einheiten und Hilfslinien* und selektieren Sie unter *Einheit, Seiten- und Linealeinheiten* nach Bedarf Punkt, Millimeter, Zoll, Pica oder Zentimeter.

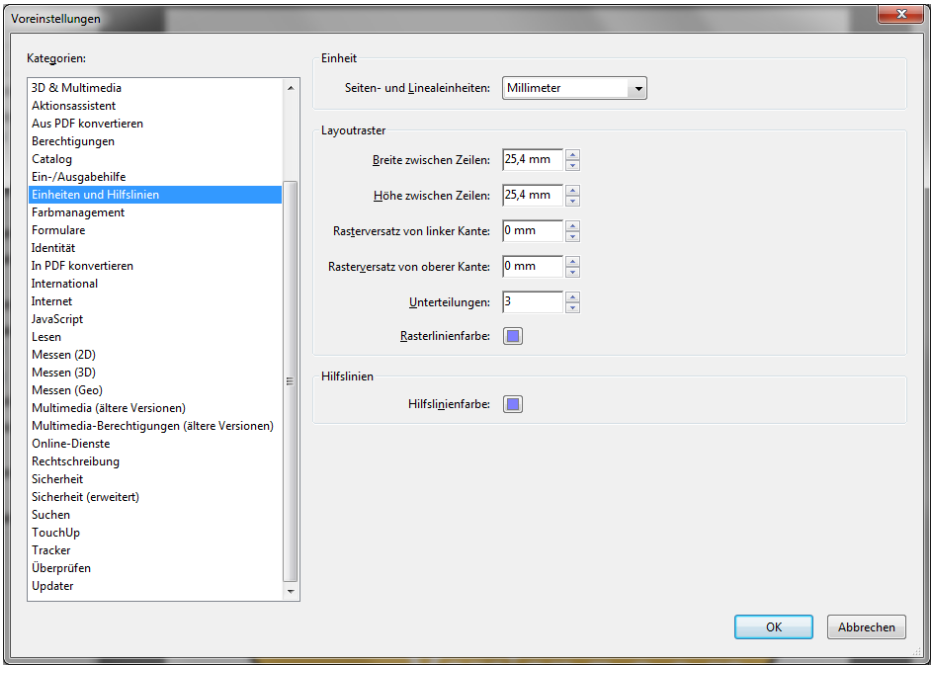


Abb. 12.3 Acrobat-Voreinstellungen für Raster

- Zum Anzeigen der Cursor-Koordinaten wählen Sie den Menübefehl *Anzeige, Ein-/Ausblenden, Cursorkoordinaten* (Acrobat X/XI) bzw. *Anzeige, Cursorkoordinaten* (Acrobat 9).

Beachten Sie, dass die gewählte Einheit ausschließlich auf die Eigenschaft *Rect* und sonst keine numerische Eigenschaft (z.B. Schriftgröße) angewandt wird.

Blöcke mit Hilfe eines Rasters positionieren. Mit Hilfe der Raster-Funktion von Acrobat können Sie Blöcke exakt positionieren und in der Größe anpassen:

- Raster einblenden: *Anzeige, Ein-/Ausblenden, Lineale und Raster* (Acrobat X/XI) oder *Anzeige, Raster (*Acrobat 9).
- Raster-Ausrichtung einschalten: *Anzeige, Ein-/Ausblenden, Lineale und Raster, Am Raster ausrichten* (Acrobat X/XI) oder *Anzeige, Am Raster ausrichten* (Acrobat 9).
- Raster ändern: *Bearbeiten, Voreinstellungen..., [Allgemein...], Einheiten und Hilfslinien*. Ändern Sie dort Abstand, Position und Farbe des Rasters.

Wenn *Am Raster ausrichten* aktiviert ist, werden Größe und Position der Blöcke am konfigurierten Raster ausgerichtet. *Am Raster ausrichten* beeinflusst neu erzeugte Blöcke sowie bestehende Blöcke, die verschoben oder mit dem Block-Plugin angepasst werden.

Anlegen von Blöcken durch Auswahl eines Rasterbildes oder einer Vektorgrafik. Statt das Rechteck für einen Block manuell aufzuziehen, können Sie die Blockgröße auch anhand von vorhandenem Seiteninhalt definieren. Dazu müssen Sie zunächst sicherstellen, dass der Menüpunkt *PDFlib Block, Zur Blockdefinition Objekt anklicken* aktiv ist. Wenn Sie nun mit dem Blockwerkzeug auf ein Rasterbild auf der Seite klicken, wird ein Block in derselben Größe und Position wie das Bild erzeugt. Sie können ebenso auf andere gra-

fische Objekte klicken. Das Blockwerkzeug versucht, die umgebende Grafik (zum Beispiel ein Logo) zu selektieren. Diese Objekt-Klick-Funktion ist als Hilfsmittel zur Definition von Blöcken gedacht. Sie können einen Block auch nachträglich ohne Einschränkungen verschieben oder in der Größe verändern. Der Block ist nicht an das Rasterbild oder das grafische Objekt gebunden, das als Hilfsmittel verwendet wurde.

Die Objekt-Klick-Funktion versucht zu erkennen, welche Vektorgrafiken oder Rasterbilder ein logisch zusammengehörendes Element auf der Seite bilden. Wenn Sie auf einen Seiteninhalt klicken, so wird dessen *Bounding Box* (das umschließende Rechteck) ermittelt und selektiert, sofern das Objekt nicht weiß oder sehr groß ist. Im nächsten Schritt werden weitere Objekte, die sich teilweise im ermittelten Rechteck befinden, zum selektierten Bereich hinzugenommen und so weiter. Auf der Grundlage des endgültigen Bereichs wird das Blockrechteck generiert. Die Objekt-Klick-Funktion versucht letztendlich, vollständige Grafiken und nicht nur einzelne Linien zu selektieren.

Automatische Erkennung von Fonteigenschaften. Das Block-Plugin ist in der Lage, den Font zu analysieren, der sich an der Stelle befindet, an der der Textline- oder Textflow-Block positioniert wird. Darauf aufbauend kann es die folgenden entsprechenden Blockeigenschaften automatisch füllen:

```
fontname, fontsize, fillcolor, charspacing, horizscaling, wordspacing,
textrendering, textrise
```

Da die automatische Erkennung von Fonteigenschaften zu unerwünschten Ergebnissen führen kann, wenn der Hintergrund ignoriert werden soll, kann diese Funktion mit *PDFlib Block, Automatische Font- und Farberkennung* aktiviert oder deaktiviert werden. Standardmäßig ist diese Funktion deaktiviert.

Sperren von Blöcken. Blöcke können gegen versehentliches Bewegen, Verkleinern/Vergrößern oder Löschen gesperrt werden. Dazu wählen Sie das Blockwerkzeug, selektieren den gewünschten Block und wählen *Sperren* aus dem Kontextmenü. Ist ein Block gesperrt, lässt er sich weder bewegen, in der Größe ändern oder löschen noch können seine Eigenschaften bearbeitet werden.

12.3.2 Bearbeiten von Blockeigenschaften

Beim Anlegen eines neuen Blocks, beim Doppelklick auf einen vorhandenen Block und bei der Selektion von *Eigenschaften* im Kontextmenü des Blocks erscheint der Dialog *PDFlib Blockeigenschaften*, in dem Sie die Eigenschaften des selektierten Blocks einstellen können (siehe Abbildung 12.2). Wie in Abschnitt 12.6, »Blockeigenschaften«, Seite 390 beschrieben, gibt es je nach Blocktyp mehrere Gruppen von Eigenschaften. Die Schaltfläche *Anwenden* ist nur aktiv, wenn Sie eine oder mehrere Eigenschaften in dem Dialog geändert haben. Bei gesperrten Blöcken ist sie deaktiviert.

Hinweis *Abhängig vom Blocktyp und bestimmten Einstellungen bei den Eigenschaften können manche Eigenschaften deaktiviert sein. Zum Beispiel ist die Eigenschaftsgruppe* `Tabulatoren für Textflowww für hortabmethod=ruler` *zur Tabulatoreinstellung nur aktiv, wenn die Eigenschaft* `hortabmethod` *in der Gruppe* `Textformatierung, Tabulatoren` *auf* `ruler` *gesetzt ist.*

Um den Wert einer Eigenschaft zu ändern, geben Sie die gewünschte Zahl oder den gewünschten Text in das Eingabefeld der Eigenschaft ein (z.B. bei *linewidth)*, selektieren den Wert aus einem Klappmenü (z.B. bei *fitmethod, orientate)* oder selektieren mit dem

»...«-Button rechts einen Font, Farbwert oder Dateinamen (z.B. bei *background color, defaultimage)*. Bei der Eigenschaft *fontname* können Sie die Schrift aus einer Liste mit allen im System installierten Schriften auswählen oder einen Schriftnamen eingeben. Unabhängig von der Auswahlmethode muss der gewählte Font auf dem System installiert sein, auf dem die Blöcke mit PPS gefüllt werden.

Geänderte Eigenschaften werden im *Blockeigenschaften*-Dialog fett dargestellt. Wenn Sie eine Eigenschaft in einem Block geändert haben, wird das Suffix (*) an den Blocknamen gehängt. Klicken Sie auf die *Anwenden* Schaltfläche, um Ihre Änderungen zu speichern und den Block zu aktualisieren. Die soeben definierten Eigenschaften werden als Bestandteil der Blockdefinition in der PDF-Datei gespeichert.

Übereinander liegende Blöcke. Überlappende Blöcke lassen sich oft nur schwer selektieren, da beim Klicken in einen Bereich nur der oberste Block selektiert wird. In solchen Fällen kann der Befehl *Block auswählen* im Kontextmenü genutzt werden, um einen der Blöcke anhand seines Namens zu selektieren. Die nächste Aktion, die im Bereich des selektierten Blocks so durchgeführt wird, bezieht sich dann nur auf diesen Block. Drücken Sie zum Beispiel die Enter-Taste, um die gewählten Blockeigenschaften zu bearbeiten. Auf diese Weise lassen sich Blöcke problemlos bearbeiten, selbst wenn sie teilweise oder vollständig von anderen Blöcken verdeckt werden.

Verwendung und Wiederherstellung von wiederverwendeten Werten. Um sich Tipp- und Klickaufwand zu ersparen, merkt sich das Blockwerkzeug die Werte, die im *Blockeigenschaften*-Dialog für den zuletzt definierten Block eingegeben wurden. Diese werden beim Anlegen eines neuen Blocks wiederverwendet. Natürlich können Sie diese Werte jederzeit umdefinieren.

Mit dem Button *Alle rücksetzen* im Fenster *PDFlib Blockeigenschaften* werden die meisten Eigenschaften auf ihre Standardwerte zurückgesetzt. Folgende Eigenschaften bleiben unverändert:

- die Eigenschaften *Blockname*, *Typ*, *Beschreibung* und *Rect*
- alle selbstdefinierten Eigenschaften

Hinweis *Verwechseln Sie die Standardwerte von vordefinierten Blockeigenschaften nicht mit den Vorgabewerten in den Eigenschaften* `defaulttext`, `defaultimage`, `defaultpdf` *und* `defaultgraphics`, *die Platzhalter-Daten zur Generierung der Vorschau enthalten (siehe Abschnitt »Vordefinierte Blockeigenschaften«, Seite 365).*

Gleichzeitige Bearbeitung mehrerer Blöcke. Um Zeit zu sparen, können Sie mehrere Blöcke gleichzeitig bearbeiten. So wählen Sie mehrere Blöcke aus:

- Aktivieren Sie das Block-Werkzeug über das Menü *PDFlib Block*, *PDFlib Block-Werkzeug*.
- Klicken Sie auf den ersten Block, den Sie auswählen möchten. Dies ist der Master-Block. Klicken Sie bei gedrückter Shift-Taste auf weitere Blöcke, die Sie bearbeiten möchten. Um alle Blöcke auf der aktuellen Seite auszuwählen, können Sie auch den Befehl *Alle Auswählen* aus dem *Bearbeiten* Menü verwenden.
- Doppelklicken Sie auf einen der Blöcke, um den *Blockeigenschaften*-Dialog zu öffnen. Dieser Block wird zum neuen Master-Block. Alternativ klicken Sie auf einen einzelnen Block, um ihn zum Master-Block zu machen und drücken dann die Enter-Taste, um den *Blockeigenschaften*-Dialog zu öffnen.

Der *Blockeigenschaften*-Dialog zeigt die gemeinsamen Eigenschaften aller ausgewählten Blöcke. Es werden die Werte des Master-Blocks verwendet. Folgendermaßen übertragen Sie Eigenschaften auf alle ausgewählten Blöcke:

- Um nur einzelne Eigenschaften zu übertragen: Ändern Sie die Werte im *Blockeigenschaften*-Dialog wie gewünscht. Die geänderten Werte werden fett dargestellt. Deaktivieren Sie *Alle Eigenschaften des Master-Blocks anwenden*. Klicken Sie auf *Anwenden*. Nur die geänderten Werte werden auf die ausgewählten Blöcke übertragen.
- Um alle Eigenschaften des Master-Blocks zu übertragen: Ändern Sie gegebenenfalls Werte im *Blockeigenschaften*-Dialog wie gewünscht. Die geänderten Werte werden fett dargestellt. Aktivieren Sie *Alle Eigenschaften des Master-Blocks anwenden*. Klicken Sie auf *Anwenden*. Alle Werte des Master-Blocks, einschließlich der geänderten, werden auf die ausgewählten Blöcke übertragen.

Die folgenden Eigenschaften können nicht für mehrere Blöcke gleichzeitig bearbeitet oder übertragen werden:

```
Name, Description, Subtype, Type, Rect, Status
```

12.3.3 Kopieren von Blöcken zwischen Seiten und Dokumenten

Das Block-Plugin bietet mehrere Methoden zum Verschieben oder Kopieren von Blöcken auf der aktuellen Seite, innerhalb des aktuellen Dokuments oder zwischen Dokumenten:

- Blöcke können durch Ziehen mit der Maus oder durch Einfügen auf einer anderen Seite oder in ein anderes offenes Dokument verschoben oder kopiert werden.
- Blöcke können auf eine oder mehrere Seiten desselben Dokuments durch normales Kopieren/Einfügen dupliziert werden.
- Blöcke können in eine neue Datei (mit leeren Seiten) oder in ein vorhandenes Dokument (auf die vorhandenen Seiten) exportiert werden.
- Blöcke können aus anderen Dokumenten importiert werden.

Um den Seiteninhalt unter Beibehaltung der Blockdefinitionen zu aktualisieren, können Sie die zugrunde liegenden Seiten ohne Veränderung der Blöcke ersetzen. Dazu verwenden Sie den Menübefehl *Werkzeuge, Seiten, Ersetzen* (Acrobat X/XI) bzw. *Dokument, Seiten ersetzen...* (Acrobat 9).

Verschieben und Kopieren von Blöcken. Sie können Blöcke verschieben oder kopieren, indem Sie einen oder mehrere Blöcke selektieren und bei gedrückter Strg-Taste (Windows) bzw. Alt-Taste (OS X) an eine neue Position ziehen. So lange die Taste gedrückt ist, nimmt der Mauszeiger eine andere Form an. Ein kopierter Block hat die gleichen Eigenschaften wie der ursprüngliche Block mit Ausnahme des Namens, der automatisch im neuen Block geändert wird.

Ebenso können Sie Blöcke mit Kopieren/Einfügen an eine andere Position auf der selben Seite, auf einer anderen Seite des selben Dokuments oder eines anderen in Acrobat geöffneten Dokuments kopieren:

- Wählen Sie das Blockwerkzeug und selektieren Sie die zu kopierenden Blöcke.
- Kopieren Sie die selektierten Blöcke mit Strg-C (Windows) bzw. Cmd-C (OS X) oder *Bearbeiten, Kopieren* in die Zwischenablage.
- Navigieren Sie zur gewünschten Zielseite.

- Stellen Sie sicher, dass das Block-Werkzeug aktiviert ist und fügen Sie mit Strg-V (Windows) bzw. Cmd-V (OS X) oder *Bearbeiten, Einfügen* die Blöcke aus der Zwischenablage an der gewünschten Position auf der Seite oder im Dokument ein.

Duplizieren von Blöcken auf andere Seiten. Sie können einen oder mehrere Blöcke gleichzeitig auf eine beliebige Anzahl von Seiten im Dokument duplizieren:

- Wählen Sie das Blockwerkzeug und selektieren Sie die zu duplizierenden Blöcke.
- Wählen Sie *Import und Export, Duplizieren...* im Menü *PDFlib Block* oder im Kontextmenü.
- Wählen Sie aus, welche Blöcke dupliziert werden sollen *(Ausgewählte Blöcke* oder Alle *Blöcke auf dieser Seite)* und auf welche Zielseiten sie kopiert werden sollen.

Import und Export von Blöcken. Mit den Funktionen zum Import und Export von Blöcken können alle Blockdefinitionen auf der Seite oder in einem Dokument über mehrere PDF-Dateien verteilt werden. Dies ist zum Beispiel bei der Aktualisierung des Seiteninhalts sinnvoll, wenn vorhandene Blockdefinitionen erhalten bleiben sollen. Um die Blockdefinitionen in eine eigene Datei zu exportieren, gehen Sie wie folgt vor:

- Wählen Sie das Blockwerkzeug und selektieren Sie die zu exportierenden Blöcke.
- Wählen Sie den Befehl *Import und Export, Export...* im Menü *PDFlib Block* oder im Kontextmenü.
- Geben Sie den Seitenbereich sowie einen Namen für die PDF-Datei ein, die die Blockdefinitionen enthalten soll.

Mit dem Befehl *Import und Export, Import...* im Menü *PDFlib Block* oder im Kontextmenü importieren Sie Blockdefinitionen. Dabei können Sie auswählen, ob die importierten Blöcke auf alle Seiten oder nur einen Seitenbereich im Dokument platziert werden. Bei mehreren selektierten Seiten werden die Blockdefinitionen unverändert auf diese kopiert. Enthält der Seitenbereich des Zieldokuments mehr Seiten als die Datei mit den zu importierenden Blockdefinitionen, können Sie die Checkbox *Vorlage wiederholen* selektieren. Die Blöcke in der zu importierenden Datei werden dann so lange wiederholt auf das Zieldokument übertragen, bis das Dokumentende erreicht ist.

Kopieren von Blöcken in ein anderes Dokument beim Export. Beim Exportieren von Blöcken können Sie diese unmittelbar auf die Seiten eines anderen Dokuments übertragen. Dazu wählen Sie ein bereits existierendes Dokument als Exportdatei. Wenn Sie die Checkbox *Vorhandene Blöcke löschen* selektieren, werden alle im Zieldokument bereits vorhandenen Blöcke gelöscht, bevor die neuen Blöcke in das Dokument kopiert werden.

12.3.4 Konvertieren von PDF-Formularfeldern in PDFlib-Blöcke

Statt PDFlib-Blöcke manuell zu erstellen, können Sie PDF-Formularfelder auch automatisch in PDFlib-Blöcke konvertieren. Dies ist insbesondere dann von Vorteil, wenn Sie bereits über komplexe PDF-Formulare mit zahlreichen Feldern verfügen, die in Zukunft automatisch mit PPS gefüllt werden sollen oder eine große Anzahl vorhandener PDF-Formulare umwandeln müssen, damit sie automatisch ausfüllbar sind. Um alle Formularfelder auf einer Seite in PDFlib-Blöcke zu konvertieren, wählen Sie *PDFlib Block, Formularfelder konvertieren, Aktuelle Seite*. Um alle Formularfelder in einem Dokument zu konvertieren, verwenden Sie stattdessen *Alle Seiten*. Um nur die selektierten Formularfelder zu konvertieren (zur Selektion wählen Sie das Formular-Werkzeug oder das Objektauswahl-Werkzeug von Acrobat), verwenden Sie *Ausgewählte Formularfelder*.

Konvertierung einzelner Eigenschaften. Bei der automatischen Konvertierung von Formularfeldern werden Formularfelder der im Dialogfeld *PDFlib Block, Formularfelder konvertieren, Konvertierungseinstellungen...* in Blöcke vom Typ *Textline* oder *Textflow* umgewandelt. Standardmäßig werden alle Formularfeldtypen konvertiert. Die Eigenschaften der Formularfelder werden gemäß Tabelle 12.3 in entsprechende Blockeigenschaften umgewandelt.

Formularfelder mit gleichen Namen. Gleichnamige Formularfelder auf der Seite sind erlaubt, Blocknamen dagegen müssen auf einer Seite eindeutig sein. Bei der Formularkonvertierung werden deshalb der Eindeutigkeit halber Zahlen an die generierten Blocknamen angehängt (siehe auch Abschnitt »Zu welchem Formularfeld gehört ein Block?«, Seite 376).

Beachten Sie, dass aufgrund eines Fehlers in Acrobat die Attribute von Formularfeldern gleichen Namens nicht korrekt wiedergegeben werden. Haben mehrere Felder denselben Namen, aber unterschiedliche Attribute, werden diese Unterschiede nicht korrekt in die generierten Blöcke übernommen. Bei der Konvertierung erscheint in solchen Fällen eine Warnung mit den Namen der betroffenen Formularfelder. Sie sollten die Eigenschaften in den generierten Blöcken dann genau überprüfen.

Zu welchem Formularfeld gehört ein Block? Da die Formularfeldnamen nicht unverändert übernommen werden, wenn mehrere Felder gleichen Namens konvertiert werden (zum Beispiel bei Radiobuttons), lässt sich schwer nachvollziehen, welcher Block eigentlich zu welchem Formularfeld gehört. Dies wäre aber insbesondere dann wichtig, wenn die Blöcke aus einer FDF- oder XFDF-Datei gefüllt werden, und das Ergebnis auch wie das ausgefüllte Formular aussehen soll.

Zur Lösung dieses Problems speichert das AcroFormConversion-Plugin bei der Erstellung des Blocks detaillierte Informationen über das zugrunde liegende Formularfeld in benutzerdefinierten Eigenschaften. Tabelle 12.3 zeigt alle benutzerdefinierten Eigenschaften, die zur zuverlässigen Ermittlung eines Blocks verwendet werden können; alle Eigenschaften haben den Typ *string*.

Tabelle 12.2 Benutzerdefinierte Eigenschaften zur Identifizierung des einem Block zugrunde liegenden Formularfelds

Selbstdefinierte Eigenschaft	***Bedeutung***
PDFlib:field:name	*Vollständig qualifizierter Name des Formularfelds*
PDFlib:field:pagenumber	*Nummer der Seite (als String) im Originaldokument, auf der sich das Formularfeld befand*
PDFlib:field:type	*Typ des Formularfelds; mögliche Werte sind* `pushbutton`, `checkbox`, `radiobutton`, `listbox`, `combobox`, `textfield`, `signature`
PDFlib:field:value	*(Nur für* `type=checkbox`*) Exportwert des Formularfelds*

Tabelle 12.3 Konvertierung von PDF-Formularfeldern in PDFlib-Blöcke

PDF-Formularfeldeigenschaft...	***...wird konvertiert in PDFlib-Blockeigenschaft***
Alle Felder	
Position	`Rect`
Name	`Name`

Tabelle 12.3 Konvertierung von PDF-Formularfeldern in PDFlib-Blöcke

PDF-Formularfeldeigenschaft...	***...wird konvertiert in PDFlib-Blockeigenschaft***
Tooltip	`Description`
Darstellung, Text, Schrift	`fontname`
Darstellung, Text, Schriftgröße	`fontsize`*; die Schriftgröße* `auto` *wird in eine feste Größe von 2/3 der Blockhöhe konvertiert, außerdem wird für die Eigenschaft* `fitmethod` *der Wert* `auto` *eingetragen. Bei mehrzeiligen Feldern/Blöcken ergibt diese Kombination eine passende Schriftgröße, die kleiner als der Anfangswert von 2/3 der Blockhöhe sein kann.*
Darstellung, Text, Textfarbe	`strokecolor` *und* `fillcolor`
Darstellung, Umrandung und Farbe, Umrandungsfarbe	`bordercolor`
Darstellung, Umrandung und Farbe, Füllfarbe	`backgroundcolor`
Darstellung, Umrandung und Farbe, Linienstärke	`linewidth`*: Thin=1, Medium=2, Thick=3*
Allgemein, Allgemeine Eigenschaften, Formularfeld	`Status`*:* *Sichtbar*=`active` *Unsichtbar*=`ignore` *Sichtbar, aber Drucken nicht möglich*=`ignore` *Unsichtbar, aber Drucken ist möglich*=`active`
Allgemein, Allgemeine Eigenschaften, Ausrichtung	`orientate`*: 0*=`north`*, 90*=`west`*, 180*=`south`*, 270*=`east`
Textfelder	
Optionen, Standardwert	`defaulttext`
Optionen, Ausrichtung	`position`*:* *Links*={`left center`} *Zentriert*={`center center`} *Rechts*={`right center`}
Optionen, Mehrere Zeilen	*erstellt einen Textflow-Block wenn aktiviert* *erstellt einen Textline-Block wenn deaktiviert*
Optionsfelder und Kontrollkästchen	
Wenn »Schaltfläche ist standardmäßig aktiviert« angeklickt ist: Optionen, Schaltflächenstil bzw. Optionen, Kontrollkästchenstil	`defaulttext`*:* *Häkchen=4* *Kreis=l* *Kreuz=8* *Karo=u* *Quadrat=n* *Stern=H* *(Diese Zeichen stellen die jeweiligen Symbole im Font ZapfDingbats dar.)*
Kombinationsfelder und Listenfelder	
Optionen, ausgewähltes (Standard)element	`defaulttext`
Schaltflächen	
Optionen, Symbol und Beschriftung, Beschriftung	`defaulttext`

Binden von Blöcken an zugehörige Formularfelder. Um PDF-Formularfelder und die daraus generierten PDFlib-Blöcke aufeinander abgestimmt zu halten, können die generierten Blöcke an die entsprechenden Formularfelder gebunden werden. Das Blockwerkzeug erhält dann die Beziehung zwischen Formularfeldern und Blöcken aufrecht. Wird der Konvertierungsprozess erneut durchgeführt, so wird ein gebundener Block gemäß der Eigenschaften des zugehörenden PDF-Formularfeldes aktualisiert. Gebundene Blöcke verhindern, dass dieselbe Arbeit doppelt erledigt werden muss: Bei der Aktualisierung eines Formulars zur interaktiven Verwendung kann der entsprechende Block automatisch mit aktualisiert werden.

Wenn Sie die Formularfelder nach der Konvertierung nicht mehr benötigen, wählen Sie im Dialogfeld *PDFlib Block, Formularfelder konvertieren, Konvertierungseinstellungen...* die Option *Konvertierte Formularfelder löschen*. Bei dieser Option werden die Formularfelder nach der Konvertierung gelöscht. Alle Aktionen (zum Beispiel JavaScript), die den betroffenen Feldern zugeordnet sind, werden ebenfalls aus dem Dokument entfernt.

Stapelkonvertierung. Wenn Sie die Formularfelder vieler PDF-Dokumente in PDFlib-Blöcke konvertieren möchten, können Sie die Stapelkonvertierung nutzen, die automatisch eine beliebige Anzahl von Dokumenten verarbeitet. Der Dialog zur Stapelverarbeitung kann über *PDFlib Block, Formularfelder konvertieren, Stapelkonvertierung...* geöffnet werden:

- Es können einzelne Dateien oder der vollständige Inhalt eines Verzeichnisses zur Verarbeitung ausgewählt werden.
- Die Ausgabedateien können im Verzeichnis der Eingabedateien oder in einem anderen Verzeichnis abgelegt werden. Sie können mit einem Präfix versehen werden, um sie von den Eingabedateien zu unterscheiden.
- Bei der Verarbeitung sehr vieler Dokumente sollte eine Log-Datei angegeben werden. In dieser werden alle verarbeiteten Dateien aufgelistet und zu jeder Datei Einzelheiten zur Konvertierung einschließlich eventueller Fehlermeldungen protokolliert.

Während der Konvertierung sind die PDF-Dokumente in Acrobat sichtbar, es können aber keinerlei Acrobat-Menüfunktionen oder Werkzeuge verwendet werden, bis zu Konvertierung abgeschlossen ist.

12.3.5 Anpassen der Benutzeroberfläche des Block-Plugins mit XML

Einige Aspekte der Block-Plugin-Benutzeroberfläche werden bei jeder Acrobat-Sitzung gestartet/neu geladen und können über eine XML-Konfigurationsdatei gesteuert werden. Das Produktpaket enthält die Beispiel-Konfigurationsdatei *factory settings.xml*. Wenn die Konfiguration geändert wurde, werden die neuen Einstellungen in der Datei *user settings.xml* gespeichert. Die geänderte Konfiguration wird jedes Mal beim Start von Acrobat geladen und beim Beenden von Acrobat gespeichert. Die Konfigurationsdatei befindet sich in einem Verzeichnis, das in etwa folgendermaßen lautet:

```
Windows XP:     C:\Dokumente und Einstellungen\<benutzer>\Eigene Dateien\PDFlib\
                Block Plugin 5
Windows 7/8:    C:\Benutzer\<benutzer>\Eigene Dokumente\PDFlib\Block Plugin 5
OS X:           <Home>/Dokumente/PDFlib/Block Plugin 5
```

Zur manuellen Bearbeitung der Konfigurationsdatei können Sie die folgenden XML-Elemente verwenden:

- Das Element */Block_Plugin/MainDialog/CloseOnApply* steuert den Ausgangsstatus des Kontrollkästchens *Dialog nach Anwenden schließen* im *Blockeigenschaften*-Dialog. Dieses Kontrollkästchen bestimmt, ob der *Blockeigenschaften*-Dialog nach dem Erstellen eines Blocks oder dem Ändern von Blockeigenschaften geöffnet bleibt.
- Das Element */Block_Plugin/FontDialog/ShowBaseFonts* steuert, ob die 14 Basis-Schriftarten in der Liste der Schriftarten des Blocke*igenschaften*-Dialogs angezeigt werden (Eigenschaftengruppe *Darstellung*, Eigenschaft *fontname)*, zusätzlich zu den auf dem System installierten Schriftarten.
- Das Element */Block_Plugin/Command/ControlByClick* steuert den Ausgangsstatus des Menüpunkts *PDFlib Block*, *Zur Blockdefinition Objekt anklicken*.
- Das Element */Block_Plugin/Command/DetectFonts* steuert den Ausgangsstatus der Menüpunkts *PDFlib Block*, *Automatische Font- und Farberkennung*.
- (Nicht unterstützt) Das Element */Block_Plugin/Command/KeyAccelerator* mit den möglichen Werten *control* (Bezeichnung für die Strg-Taste auf Windows und die Cmd-Taste unter OS X), *control+shift* oder *none* steuert die Zugriffstaste für die folgenden Tastaturkürzel:

  ```
  A (Alles auswählen), C (Kopieren), I (Dialog PPDFlib Blockeigenschaften),
  V (Einfügen), X (Ausschneiden)
  ```

  Die Änderung wirkt nur beim Start von Acrobat, weil Tastenkombinationen nicht zur Laufzeit geändert werden können. Wenn dieser Eintrag nicht vorhanden ist, stehen keine Tastaturkürzel zur Verfügung. Der Standardwert ist *control*.

12.4 Block-Vorschau in Acrobat

Hinweis *Sie können die Vorschau-Funktion mit dem Dokument* `block_template.pdf` *aus dem PDFlib-Paket ausprobieren. Die erforderlichen Daten (z.B. Font und Rasterbild) sind ebenfalls im Paket enthalten.*

PDFlib-Blöcke werden von PPS verarbeitet, wobei das Füllen eines Blocks bezüglich der Datenquellen (z.B. Text aus einer Datenbank, Bilddateien auf der Festplatte) sowie visuelle und interaktive Aspekte der generierten Dokumente angepasst werden können. Für weitere Informationen siehe Abschnitt 12.5, »Füllen von Blöcken mit PPS«, Seite 385.

Außerdem enthält das Block-Plugin eine integrierte Version von PPS, mit der Sie Vorschau-Versionen der gefüllten Blöcke in Acrobat ohne Programmierkenntnisse erzeugen können. Obwohl diese Vorschaufunktion nicht die gleiche Flexibilität wie eine maßgeschneiderte Programmierung bieten kann, gibt sie doch einen schnellen Überblick über die Verarbeitung der Blöcke. Die Block-Vorschau können Sie zur Verbesserung der Position und Größe der Blöcke als auch für die Überprüfung der Block-Eigenschaften verwenden (z. B. Schriftart und Größe). Sie können die Blöcke ändern und eine neue Vorschau erstellen, bis Sie mit dem in der Vorschau angezeigten Ergebnis zufrieden sind. Vorschauen können für die aktuelle Seite oder das gesamte Dokument erzeugt werden.

Die Vorschau wird immer in einem neuen PDF-Dokument angezeigt. Das Originaldokument mit den Blöcken wird durch Erzeugen einer Vorschau nicht verändert. Sie können das Vorschau-Dokument nach Ihren Bedürfnissen speichern oder verwerfen.

Vorgabewerte. Da die serverseitigen Datenquellen (z.B. eine Datenbank) für Text, Vektorgrafiken, Rasterbilder oder PDF-Inhalte eines Blocks nicht im Plugin verfügbar sind, wird in der Vorschau-Funktion immer der Vorgabewert des Blocks verwendet, das heißt die Werte der Eigenschaften *defaulttext*, *defaultimage*, *defaultpdf* oder *defaultgraphics*. Üblicherweise wird ein Beispiel-Datensatz für die Vorgabewerte verwendet, der für den in PPS verwendeten realen Blockinhalt charakteristisch ist. Blöcke ohne Vorgabewerte werden bei der Erzeugung der Vorschau ignoriert, ebenso wie Blöcke mit *Status* = *ignoredefault*.

Die Vorgabewerte für neue Blöcke sind leer. Bevor Sie die Vorschau-Funktion verwenden, müssen Sie die Eigenschaften *defaulttext*, *defaultimage*, *defaultpdf* oder *defaultgraphics* (je nach Blocktyp) in der Eigenschaftengruppe *Vorgabewerte* festlegen oder geeignete Werte für gleichnamige Objekte im Dialog *Erweiterte PPS-Optionen* angeben.

Hinweis *Den Vorgabewert für Symbol-Fonts einzugeben, kann etwas knifflig sein, siehe Abschnitt »Verwendung von Symbolfonts für Vorgabetext«, Seite 384.*

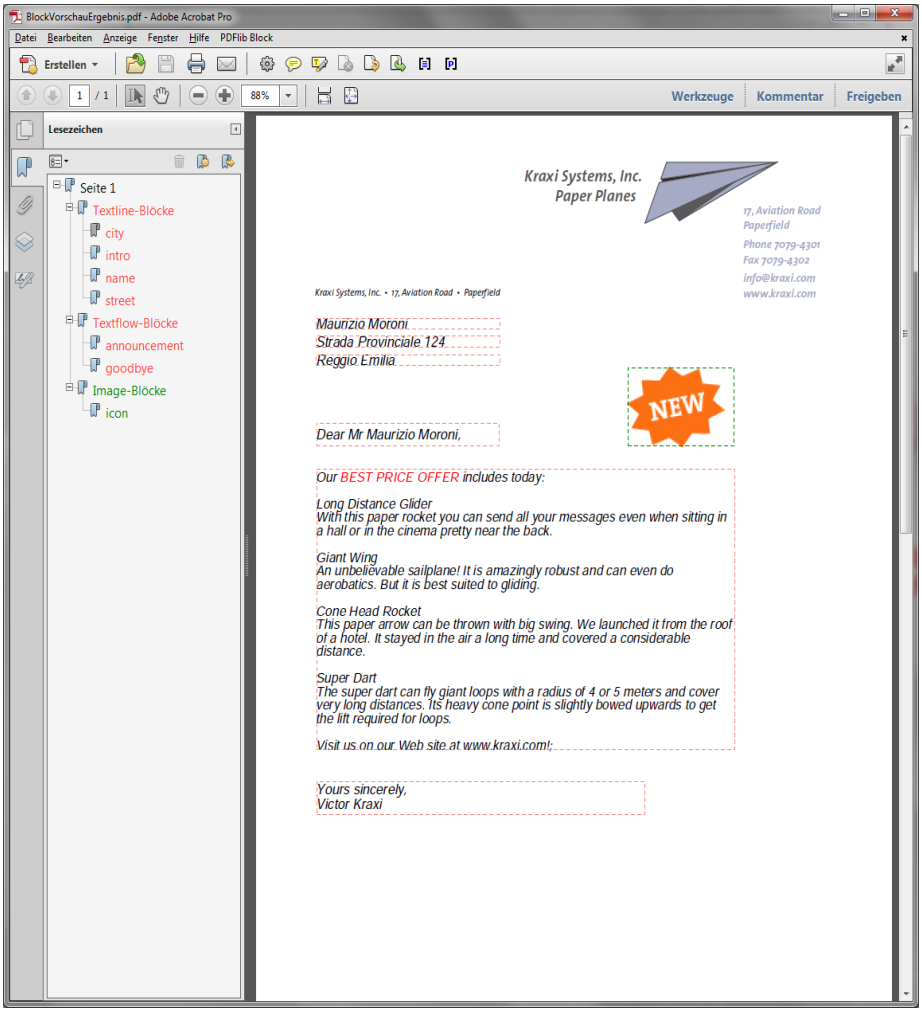
Einrichten der Block-Vorschau. Sie können die Block-Vorschau auf folgende Arten erstellen:

- Über den Menübefehl *PDFlib Block*, *Vorschau*, *Vorschau erstellen*.
- Durch Anklicken des Symbols für die PDFlib Block-Vorschau  in der Werkzeugleiste unter *Erweiterte Bearbeitung* (Acrobat X/XI) bzw. über die Werkzeugleiste *Erweiterte Bearbeitung* (Acrobat 9). Wenn Acrobat diese Leiste nicht anzeigt, können Sie sie über *Anzeige, Werkzeuge, Erweiterte Bearbeitung* (Acrobat X/XI) bzw. über *Anzeige, Werkzeugleiste, Erweiterte Bearbeitung* (Acrobat 9) einblenden.

- Bei aktiviertem Blockwerkzeug können Sie auch außerhalb eines Blocks rechtsklicken und das Kontextmenü mit den Einträgen *Vorschau erstellen* und *Vorschau-Konfiguration* öffnen.

Die Vorschau wird auf der Grundlage der PDF-Datei auf der Festplatte erstellt. Alle Änderungen, die Sie in Acrobat vorgenommen haben, werden nur in der Vorschau berücksichtigt, wenn Sie die Block-PDF mit *Datei*, *Speichern* oder *Datei*, *Speichern unter...* vorher auf der Festplatte gespeichert haben. Modifizierte Blöcke können Sie an dem Stern hinter dem Blocknamen erkennen. Sie können die Vorschau-Funktion so konfigurieren, dass die Blockdatei automatisch gespeichert wird, bevor Sie eine Vorschau erstellen. So können Sie sicherstellen, dass ihre Änderungen unverzüglich in der Vorschau berücksichtigt werden.

Abb. 12.4 Vorschau-PDF für das Container-Dokument aus Abbildung 12.1 mit Info-Ebenen und Annotationen für Blöcke



Konfigurieren der Vorschau. Sie können die Erstellung der Block-Vorschau und des zugrunde liegenden PPS-Verhaltens über den Menübefehl *PDFlib Block*, *Vorschau*, *Vorschau-Konfiguration*... konfigurieren:

- Vorschau für die aktuelle Seite oder das gesamte Dokument;
- Ausgabe-Verzeichnis für die erstellten Vorschau-Dokumente;
- Automatisches Speichern der Blockdatei vor dem Erstellen der Vorschau;
- Hinzufügen von Info-Ebenen und Annotationen für Blöcke;
- Blöcke in die erzeugte Ausgabe kopieren
- *PDF/A-, PDF/UA- oder PDF/X-Status der Block-Datei klonen:* Da diese Standards die Verwendung von Ebenen und Annotationen einschränken, kann die Option *Info-Ebenen und Annotationen erzeugen* nicht gleichzeitig aktiviert sein.
- *Kopiere Blöcke in Vorschaudatei* erlaubt das Kopieren von PDFlib-Blöcken in die generierte Vorschau beim Befüllen. Alle Blöcke werden kopiert, unabhängig davon, ob sie erfolgreich gefüllt werden konnten.
- Im Dialog *Erweiterte PPS-Optionen* können Sie zusätzliche Optionslisten für die PPS-Funktionen auf Basis der PPS-Programmschnittstelle erstellen. Zum Beispiel kann die Option *searchpath* für *PDF_set_option( )* verwendet werden, um ein Verzeichnis anzugeben, in dem sich Fonts und Rasterbilder für das Füllen der Blöcke befinden. Wir empfehlen, erweiterte Optionen in Zusammenarbeit mit dem Entwickler des PPS-Codes festzulegen.

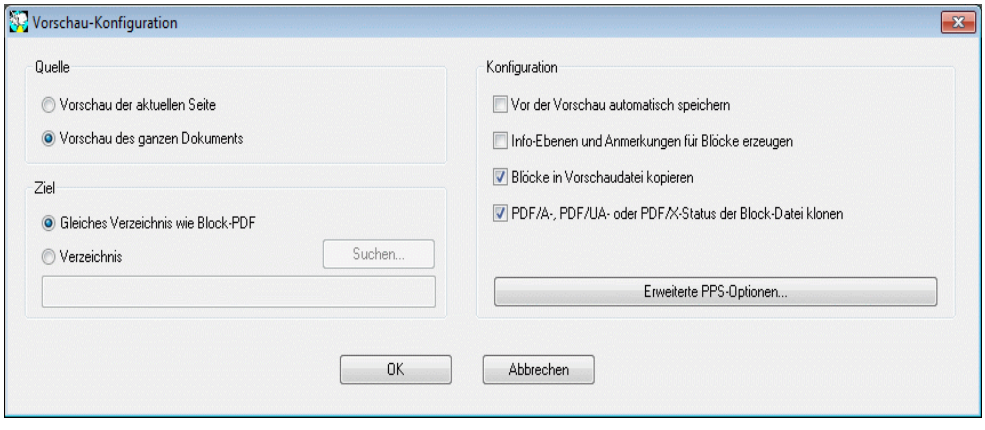


Abb. 12.5 Blockvorschau-Konfiguration

Mit der Vorschau bereitgestellte Informationen. Die erzeugten Vorschau-Dokumente enthalten die ursprünglichen Seiteninhalte (Hintergrund), die gefüllten Blöcke und gegebenenfalls verschiedene andere Informationen. Diese Informationen können nützlich sein zur Überprüfung und Verbesserung von Block- und PPS-Konfiguration. Die folgenden Elemente werden für jeden aktiven Block mit Standardinhalt erstellt:

- *Fehlermarkierungen:* Blöcke, die nicht erfolgreich gefüllt werden konnten, werden durch ein durchgestrichenes Rechteck gekennzeichnet, so dass sie leicht identifiziert werden können. Fehlermarkierungen werden immer erzeugt, wenn ein Block nicht verarbeitet werden konnte.

- *Lesezeichen:* Die verarbeiteten Blöcke werden in Lesezeichen zusammengefasst, die nach Seitenzahl, Blocktyp und möglichen Fehlern strukturiert sind. Die Lesezeichen können Sie über den Menübefehl *Anzeige, Ein-/Ausblenden, Navigationsfenster, Lesezeichen* (Acrobat X/XI) bzw. über *Anzeige, Navigationsfenster, Lesezeichen* (Acrobat 9) anzeigen. Lesezeichen werden immer erstellt.
- *Annotationen:* Für jeden verarbeiteten Block wird eine Annotation auf der Seite neben dem eigentlichen Blockinhalt erstellt. Das Annotations-Rechteck visualisiert die ursprüngliche Blockgröße (abhängig von Vorgabewert und Füllmodus kann dies von der Ausdehnung des Blockinhalts abweichen). Die Annotation enthält den Namen des Blocks und eine Fehlermeldung, wenn der Block nicht gefüllt werden konnte. Annotationen werden standardmäßig generiert, können aber in der Vorschau-Konfiguration deaktiviert werden. Da die Verwendung von Annotationen in den Standards PDF/A und PDF/X beschränkt ist, werden Annotationen nicht erstellt, wenn die Option *PDF/A-, PDF/UA- oder PDF/X-Status der Block-Datei klonen* aktiviert ist.
- *Ebenen*: Seiteninhalte werden auf Ebenen platziert, um Analyse und Fehlersuche zu erleichtern. Eine separate Ebene wird für den Seitenhintergrund erstellt (das heißt für den Inhalt der ursprünglichen Seite), ebenso für jeden Blocktyp, für Fehlerblöcke, die nicht gefüllt werden konnten, sowie für die Annotationen mit den Block-Informationen. Wenn eine Ebene leer bleibt (z.B. weil keine Fehler aufgetreten sind), wird sie nicht erstellt. Die Ebenen können Sie über den Menübefehl *Anzeige, Ein-/Ausblenden, Navigationsfenster, Ebenen* anzeigen lassen. Standardmäßig werden alle Ebenen auf der Seite angezeigt. Um den Inhalt einer Ebene auszublenden, klicken Sie links neben dem Namen der Ebene auf das Augensymbol. Sie können das Erstellen von Ebenen in der Vorschau-Konfiguration deaktivieren. Da die Verwendung von Ebenen in den Standards PDF/A-1, PDF/X-1a und PDF/X-3 beschränkt ist, werden die Ebenen nicht erstellt, wenn die Option *PDF/A-, PDF/UA- oder PDF/X-Status der Block-Datei klonen* aktiviert ist.

PDF/A-, PDF/UA- oder PDF/X-Status klonen. Mit der Option *PDF/A-, PDF/UA- oder PDF/X-Status der Block-Datei klonen* können Sie PDF-Dateien gemäß eines dieser Standards erstellen. Der Klon-Modus kann aktiviert werden, wenn die Quelldatei einem der folgenden Standards entspricht:

```
PDF/A-1a:2005, PDF/A-1b:2005
PDF/A-2a, PDF/A-2b, PDF/A-2u
PDF/A-3a, PDF/A-3b, PDF/A-3u

PDF/UA-1

PDF/X-1a:2003
PDF/X-3:2003
PDF/X-4, PDF/X-4p
PDF/X-5g, PDF/X-5pg
```

Wenn eine Vorschau im Klon-Modus erstellt wird, dupliziert PPS die folgenden Aspekte der Block-PDF in der generierten Vorschau:

- die PDF-Standard-Identifizierung;
- Druckausgabebedingung;
- Seitengrößen einschließlich aller Seitenangaben;
- Tagged PDF: Dokumentsprache (falls vorhanden)
- XMP-Metadaten des Dokuments.

Beim Klonen standardkonformer PDF-Dokumente müssen alle Block-Füllvorgänge dem jeweilige Standard entsprechen. Wenn zum Beispiel keine Druckausgabebedingung vorhanden ist, können keine RGB-Bilder ohne ICC-Profil verwendet werden. Ebenso müssen alle verwendeten Fonts eingebettet werden. Die vollständige Liste der Anforderungen finden Sie in Abschnitt 11.3, »PDF/A zur Archivierung«, Seite 325 und Abschnitt 11.4, »PDF/X zur Druckproduktion«, Seite 337. Wenn ein Block-Füllvorgang im Klon-Modus den ausgewählten Standard verletzen würde (z. B. weil ein Vorgabebild RGB-Farben verwendet, aber das Dokument keine geeignete Druckausgabebedingung enthält), erscheint eine Fehlermeldung und es wird keine Vorschau erzeugt. Auf diese Weise können Benutzer potenzielle Standard-Verletzungen sehr früh im Workflow erkennen.

Verwendung von Symbolfonts für Vorgabetext. Sie können Symbol-Fonts auf zwei Arten im Vorgabetext verwenden:

- Bei der Arbeit mit 8-Bit-Legacy-Codes, wie z. B. in der Windows-Zeichentabelle gezeigt: Erstellen Sie die 8-Bit-Codes für *defaulttext* entweder durch Eingabe der entsprechenden 8-Bit-Zeichen (z.B. durch Kopieren/Einfügen aus der Windows-Zeichentabelle) oder als numerische Escape-Sequenz. In diesem Fall müssen Sie den Vorgabewert der Eigenschaft *charref* in der Eigenschaftsgruppe *Textvorbereitung* als *false* beibehalten, und Sie können nicht mit Character-Referenzen arbeiten. Zum Beispiel erstellt der folgende Vorgabetext die Glyphe »Smiley« aus dem Symbol-Font Wingdings, wenn *charref = false* gesetzt ist:

  ```
  J
  \x4A
  \112
  ```

- Bei der Arbeit mit Unicode-Werten oder Glyphnamen aus dem verwendeten Font: Stellen Sie die Eigenschaft *charref* der Eigenschaftsgruppe *Textvorbereitung* auf *true* und geben Sie die Character-Referenzen oder Glyphnamen für die Symbole an (siehe Abschnitt 4.6.2, »Character-Referenzen«, Seite 114). Zum Beispiel erstellt der folgende Standardtext die Glyphe »Smiley« aus dem Symbol-Font Wingdings, wenn *charref = true* gesetzt ist:

  ```
  &#xF04A;
  &.smileface;
  ```

  Beachten Sie, dass bei beiden Methoden statt der tatsächlichen symbolischen Glyphen eine alternative Darstellung im *Blockeigenschaften*-Dialog angezeigt wir.

12.5 Füllen von Blöcken mit PPS

Um Blöcke mit PPS zu füllen, müssen Sie zunächst die Seite mit den Blöcken mit der Funktion *PDF_fit_pdi_page( )* auf der Ausgabeseite platzieren. Nachdem Sie die Seite platziert haben, können Sie sie mit den Funktionen *PDF_fill_* block( )* füllen.

Einfaches Beispiel: Hinzufügen von variablem Text zu einem Template. Eine häufige Aufgabe besteht darin, ein PDF-Template mit dynamischem Text anzureichern. Das folgende Codefragment öffnet eine Seite in einem Eingabe-Dokument (dem PDF-Template oder Block-Container), platziert diese auf der Ausgabeseite und füllt einen Textblock namens *firstname* mit variablem Text:

```
doc = p.open_pdi_document(filename, "");
if (doc == -1)
        throw new Exception("Fehler: " + p.get_errmsg());

page = p.open_pdi_page(doc, pageno, "");
if (page == -1)
        throw new Exception("Fehler: " + p.get_errmsg());

p.begin_page_ext(width, height, "");
/* Platzieren der importierten Seite */
p.fit_pdi_page(page, 0.0, 0.0, "");

/* Füllen eines einzelnen Blocks auf der platzierten Seite */
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");

p.close_pdi_page(page);
p.end_page_ext("");
p.close_pdi_document(doc);
```

Cookbook *Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic* `blocks/starter_block`.

Überschreiben von Blockeigenschaften. In manchen Situationen möchte der Programmierer nur gewisse Eigenschaften in einer Blockdefinition verwenden, andere dagegen mit eigenen Werten überschreiben. Dies kann in verschiedenen Situationen nützlich sein:

- Bestimmte Überschreibungen können von der Business-Logik her sinnvoll sein.
- Der Skalierungsfaktor für ein Bild oder eine PDF-Seite wird nicht der Blockdefinition entnommen, sondern berechnet.
- Die Blockkoordinaten werden vom Programm angepasst, etwa um eine Rechnung mit einer variablen Zahl von Einträgen zu erstellen.
- Im Programm werden individuelle Schmuckfarbnamen angegeben, um bei der Erstellung von Drucksachen den Kundenanforderungen gerecht zu werden.

Blockeigenschaften können überschrieben werden, indem man den Namen der Blockeigenschaft und die gewünschten Werte in der Optionsliste der *PDF_fill_*block( )*-Funktionen wie folgt angibt:

```
p.fill_textblock(page, "firstname", "Serge", "fontsize=12");
```

Diese Anweisung überschreibt die interne Eigenschaft *fontsize* des Blocks mit dem angegebenen Wert 12. Fast alle Namen von Blockeigenschaften können als Optionen benutzt werden.

Das Überschreiben von Blockeigenschaften wirkt sich nur auf den jeweiligen Funktionsaufruf aus. Die Änderung wird nicht in der Blockdefinition gespeichert.

Platzieren importierter Seiten über gefüllten Blöcken. Bevor Blöcke gefüllt werden können, muss die importierte Seite auf der Ausgabeseite platziert werden. Das bedeutet in der Regel, dass sich die Originalseite unterhalb der Blockinhalte befindet. Manchmal ist es jedoch wünschenswert, dass die Seite oberhalb der gefüllten Blöcke zu liegen kommt. Dies erreichen Sie folgendermaßen: Platzieren Sie die Seite mit der Option *blind* von *PDF_fit_pdi_page( )*, damit die Blöcke und ihre Position für PPS bekannt sind. Platzieren Sie sie nach dem Füllen der Blöcke dann erneut, um die tatsächlichen Seiteninhalte anzuzeigen:

```
/* Zur Vorbereitung der Blöcke Seite unsichtbar im Blind-Modus positionieren */
p.fit_pdi_page(page, 0.0, 0.0, "blind");

/* Blöcke füllen */
p.fill_textblock(page, "firstname", "Serge", "encoding=winansi");
/* ... weitere Blöcke füllen ... */

/* Seite erneut platzieren, diesmal sichtbar */
p.fit_pdi_page(page, 0.0, 0.0, "");
```

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `blocks/block_below_contents`.

Ignorieren der Container-Seite beim Füllen von Blöcken. Importierte Blöcke können manchmal auch als Platzhalter ohne Bezug zum darunterliegenden Seiteninhalt sinnvoll sein. Beispielsweise können Sie eine Container-Seite mit Blöcken im *blind*-Modus, d.h. mit der Option *blind* von *PDF_fit_pdi_page( )* auf mehreren Ausgabeseiten platzieren und dann die Blöcke füllen. Auf diese Art können Sie die Blockeigenschaften nutzen und sogar Blöcke auf vielen Seiten (oder derselben Ausgabeseite) duplizieren, ohne die Container-Seite auf die Ausgabeseite zu platzieren.

Cookbook Ein vollständiges Codebeispiel hierzu finden Sie im Cookbook-Topic `blocks/duplicate_block`.

Verbinden von Textflow-Blöcken. Blöcke lassen sich so verbinden, dass Text von einem Block in den nächsten fließt. Wenn Sie langen variablen Text unter Umständen auf einer zweiten Seite fortsetzen müssen, können Sie zwei Blöcke verbinden und den verbleibenden Text des ersten Blocks im zweiten Block platzieren.

PPS erzeugt aus dem an *PDF_fill_textblock( )* übergebenen Text sowie den Blockeigenschaften intern einen Textflow. Ohne verbundene Blöcke wird der Textflow im Block platziert und das zugehörige Textflow-Handle am Ende des Aufrufs gelöscht; übriger Text geht dabei verloren.

Bei verbundenen Textflow-Blöcken kann übrig bleibender Text des ersten Blocks im nächsten Block platziert werden. Statt einen neuen Textflow zu erzeugen, wird der übrige Textflow als Blockinhalt verwendet. Textflow-Blöcke werden wie folgt verbunden:

- Beim ersten Aufruf von *PDF_fill_textblock( )* innerhalb einer Folge von verbundenen Textflow-Blöcken wird in der Option *textflowhandle* der Wert -1 (in PHP: 0) überge-

ben. *PDF_fill_textblock( )* gibt ein intern erzeugtes Textflow-Handle zurück, das von der Anwendung gespeichert werden muss.

- Beim nächsten Aufruf von *PDF_fill_textblock( )* kann in der Option *textflowhandle* das im vorigen Schritt erhaltene Textflow-Handle übergeben werden (der im Parameter *text* übergebene Text wird in diesem Fall ignoriert und sollte leer sein). Der Block wird mit dem Rest des Textflows gefüllt.
- Dieser Vorgang kann mit weiteren Textflow-Blöcken fortgeführt werden.
- Das zurückgegebene Textflow-Handle kann an *PDF_info_textflow( )* übergeben werden, um das Resultat des Vorgangs zu ermitteln, z.B. den Zustand oder die Textposition bei Vorgangsende.

Beachten Sie, dass die Eigenschaft *fitmethod* auf *clip* gesetzt werden sollte (dies ist ohnehin die Standardeinstellung, wenn *textflowhandle* übergeben wird). Das grundlegende Code-Fragment zum Verbinden von Textflow-Blöcken lautet wie folgt:

```
p.fit_pdi_page(page, 0.0, 0.0, "");
tf = -1;

for (i = 0; i < blockcount; i++)
{
        String optlist = "encoding=winansi textflowhandle=" + tf;
        int reason;
        tf = p.fill_textblock(page, blocknames[i], text, optlist);
        text = null;

        if (tf == -1)
            break;

        /* Ergebnis des letzen Aufrufs von fit_textflow() untersuchen */
        reason = (int) p.info_textflow(tf, "returnreason");
        result = p.get_string(reason);

        /* Schleife beenden, wenn der Text vollständig platziert wurde */
        if (result.equals("_stop"))
        {
            p.delete_textflow(tf);
            break;
        }
}
```

Cookbook Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic `blocks/linked_textblocks`.

Reihenfolge der Block-Befüllung. Die Blockfunktionen *PDF_fill_*block( )* verarbeiten Blockeigenschaften und Blockinhalte in der folgenden Reihenfolge:

- Hintergrund: Wenn die Eigenschaft *backgroundcolor* vorhanden ist und das Schlüsselwort *colorspace* einen Wert ungleich *None* hat, wird der Blockbereich mit der festgelegten Farbe gefüllt.
- Rahmen: Wenn die Eigenschaft *bordercolor* vorhanden ist und das Schlüsselwort *colorspace* einen Wert ungleich *None* hat, wird der Blockrand in der festgelegten Farbe und Linienstärke gezeichnet.
- Inhalt: Die Blockinhalte und alle anderen Eigenschaften außer *bordercolor* und *linewidth* werden verarbeitet.
- Textline- und Textflow-Blöcke: Wenn weder Text noch Vorgabetext übergeben wird, gibt es keinerlei Ausgabe, auch nicht für die Hintergrundfarbe oder die Blockränder.

Verschachtelte Blöcke. Bevor Blöcke gefüllt werden können, muss die importierte Seite auf der Ausgabeseite platziert werden (da PPS sonst die Lage der Blöcke nach dem Skalieren, Drehen und Verschieben der Seite nicht erkennen würde). Wenn die Seite nur als Block-Container ohne statische Inhalte auf der neue Seite verwendet wird, können Sie die importierten Seite auch mit der Option *blind* platzieren.

Für eine erfolgreiche Block-Befüllung spielt es keine Rolle, wie die importierte Seite auf der Ausgabeseite platziert wurde:

- Die Seite kann direkt mit *PDF_fit_pdi_page( )* platziert werden.
- Die Seite kann indirekt in einer Tabellenzelle mit *PDF_fit_table( )* platziert werden.
- Die Seite kann als Inhalt eines anderen PDF-Blocks mit *PDF_fill_pdfblock( )* platziert werden.

Die dritte Methode, das heißt das Füllen eines PDF-Blocks mit einer anderen Seite mit Blöcken ermöglicht verschachtelte Block-Container. Damit lassen sich interessante Anwendungsfälle implementieren. Zum Beispiel können Sie Bogenmontage und Personalisieren in einem zweistufigen Block-Füllvorgang implementieren:

- Die Block-Containerseite auf erster Ebene enthält mehrere große PDF-Blöcke, die die Hauptbereiche auf dem zu bedruckenden Papier markieren. Die Anordnung der PDF-Blöcke spiegelt die geplante Nachbearbeitung des Papiers wider (z. B. Falten oder Schneiden).
- Jeder der PDF-Blöcke auf erster Ebene wird dann mit einer PDF-Containerseite auf zweiter Ebene gefüllt. Diese enthält Text-, Bild-, PDF- oder Grafik-Blöcke, die zur individuellen Gestaltung mit variablem Text befüllt werden können.

Mit diesem Verfahren können Block-Container verschachtelt werden. Obwohl Blöcke beliebig tief verschachtelt werden können, wird eine Verschachtelungstiefe von drei oder mehr Ebenen nur selten benötigt werden.

Die Block-Container auf zweiter Ebene (z.B. eine Vorlagenseite für einen Brief) können für jede montierte Seite identisch oder verschieden sein. Wenn sie identisch sind, müssen erst die Blöcke auf der Briefvorlage gefüllt werden, bevor die Briefvorlage selbst in den nächsten Block erster Ebene platziert wird, da PPS immer den Ort der letzten Platzierung der Vorlagenseite verwendet.

Cookbook *Ein vollständiges Codebeispiel finden Sie im Cookbook-Topic* `blocks/nested_blocks`.

Block-Koordinaten. Die Rechteck-Koordinaten für einen Block beziehen sich auf das Standardkoordinatensystem von PDF. Wird die Seite, die den Block enthält, mit PPS auf der Ausgabeseite platziert, können an *PDF_fit_pdi_page( )* verschiedene Optionen zur Positionierung und Skalierung übergeben werden. Diese Parameter werden bei der Verarbeitung des Blocks berücksichtigt. Damit wird es möglich, eine Template-Seite mehrfach auf der Ausgabeseite zu platzieren, wobei deren Blöcke jedes Mal mit Daten gefüllt werden. So könnte das Template für eine Visitenkarte zum Beispiel viermal auf ein Blatt montiert werden. Die Blockfunktionen kümmern sich um Transformationen des Koordinatensystems und die korrekte Platzierung des Texts für alle Blöcke bei allen Aufrufen der Seite. Vorausgesetzt wird hier lediglich, dass der Client die Seite platziert und alle Blöcke auf der platzierten Seite verarbeitet. Die Seite kann dann auf der Ausgabeseite an anderer Stelle erneut platziert werden, wobei weitere Blockverarbeitung an der neuen Position stattfindet usw.

Das Block-Plugin zeigt die Blockkoordinaten anders an, als sie intern in der PDF-Datei gespeichert werden. Während das Plugin mit der bei Acrobat üblichen Notation ar-

beitet und den Koordinatenursprung in der linken oberen Ecke hat, beziehen sich die internen Koordinaten (die im Block gespeichert werden) auf die PDF-Konvention und haben den Ursprung in der linken unteren Ecke der Seite. Die Koordinatenanzeige im *Blockeigen-schaften*-Dialog ist abhängig von den in Acrobat festgelegten Einheiten (siehe Abschnitt »Blockgröße und -position«, Seite 370).

Schmuckfarben in Blockeigenschaften. Um in einer Blockeigenschaft eine Schmuckfarbe zu verwenden, können Sie mit der Schaltfläche »...« eine Liste aller HKS- und Pantone-Schmuckfarben anzeigen. Die angezeigten Farbnamen sind in PPS integriert (siehe Abschnitt 3.4.1, »Color Management mit ICC-Profilen«, Seite 88) und können ohne weitere Vorkehrungen verwendet werden. Für benutzerdefinierte Schmuckfarben kann im Block-Plugin eine Alternativfarbe definiert werden. Ist in den Blockeigenschaften keine Alternativfarbe spezifiziert, muss die benutzerdefinierte Schmuckfarbe in der PPS-Anwendung bereits früher mit *PDF_makespotcolor( )* definiert worden sein. Andernfalls schlagen die Blockfunktionen fehl.

12.6 Blockeigenschaften

PPS und das Block-Plugin bieten einige allgemeine Eigenschaften (»Properties«), die jedem Blocktyp zugeordnet werden können. Daneben gibt es Eigenschaften, die spezifisch für die Blocktypen *Textline, Textflow*, *Image, PDF* und *Graphics* sind.

Eigenschaften unterstützen dieselben Datentypen wie Optionslisten mit Ausnahme von Handles und Aktionslisten. Viele Blockeigenschaften heißen wie die Optionen für API-Funktionen wie *PDF_fit_image( ), PDF_fit_textline( )* (zum Beispiel *fitmethod, charspacing)*. In solchen Fällen verhält sich die Eigenschaft genau so wie die gleichnamige Option.

12.6.1 Administrative Eigenschaften

Administrative Eigenschaften beziehen sich auf alle Arten von Blöcken. Alle erforderlichen Einträge werden vom Block-Plugin automatisch generiert. Tabelle 12.4 gibt eine Übersicht über die administrativen Eigenschaften.

Tabelle 12.4 Administrative Blockeigenschaften

Schlüsselwort	Mögliche Werte und Erklärung
Description	*(String) Vom Benutzer lesbare Beschreibung der Funktion des Blocks, die in PDFDocEncoding oder Unicode kodiert ist (und im letzteren Fall mit einem BOM beginnt). Diese Eigenschaft dient nur zur Information des Benutzers und wird von PPS ignoriert.*
Locked	*(Boolean) Ist diese Eigenschaft gleich* `true`*, lassen sich der Block und seine Eigenschaften nicht mit dem Block-Plugin bearbeiten. Diese Eigenschaft wird von PPS ignoriert. Standardwert:* `false`
Name	*(String; erforderlich) Name des Blocks. Blocknamen müssen auf der Seite, aber nicht innerhalb des Dokuments eindeutig sein. Die drei Zeichen* `[ ] /` *sind in Blocknamen nicht erlaubt. Blocknamen dürfen maximal 125 Zeichen lang sein.*
Subtype	*(Schlüsselwort; erforderlich) Je nach Blocktyp entweder* `Text`*,* `Image`*,* `PDF` *oder* `Graphics`*. Beachten Sie, dass Textline- und Textflow-Blöcke als Subtyp beide* `Text` *haben, aber durch die Eigenschaft* `textflow` *unterschieden werden.*
textflow	*(Boolean) Steuert die ein- oder mehrzeilige Verarbeitung. Diese Eigenschaft ist nicht explizit in der Benutzeroberfläche des Block-Plugin verfügbar, sondern als Blocktyp Textline oder Textflow dargestellt (Standardwert:* `false`*):* ***false*** *Textline-Block: Text muss einzeilig sein und wird mit PDF_fit_textline( ) verarbeitet.* ***true*** *Textflow-Block: Text kann mehrzeilig sein und wird mit PDF_fit_textflow( ) verarbeitet. Neben den vordefinierten Eigenschaften für Text können alle Textflow-spezifischen Eigenschaften festgelegt werden (siehe Tabelle 12.9).*
Type	*(Schlüsselwort; erforderlich) Immer* `Block`

12.6.2 Eigenschaften für Rechtecke

Eigenschaften für Rechtecke beziehen sich auf alle Arten von Blocktypen. Sie legen das Aussehen des Blockrechtecks fest. Alle erforderlichen Einträge werden vom Block-Plugin automatisch generiert. Tabelle 12.5 gibt eine Übersicht über die Eigenschaften für Rechtecke.

Tabelle 12.5 Blockeigenschaften für Rechtecke

Schlüsselwort	Mögliche Werte und Erklärung
backgroundcolor	*(Color) Ist diese Eigenschaft vorhanden und enthält sie ein von* `None` *verschiedenes Schlüsselwort zur Festlegung des Farbraums, wird ein Rechteck gezeichnet und mit der angegebenen Farbe gefüllt. Damit lassen sich vorhandene Seiteninhalte verdecken. Standardwert:* `None`
bordercolor	*(Color) Ist diese Eigenschaft vorhanden und enthält sie ein von* `None` *verschiedenes Schlüsselwort zur Festlegung des Farbraums, wird ein Rechteck mit einem Rand in der angegebenen Farbe gezeichnet. Standardwert:* `None`
linewidth	*(Float; muss größer* `0` *sein) Breite der Linie, mit der das Blockrechteck gezeichnet wird; wird nur verwendet, wenn* `bordercolor` *gesetzt ist. Standardwert: 1*
Rect	*(Rechteck; erforderlich) Blockkoordinaten. Der Ursprung des Koordinatensystems liegt in der linken unteren Ecke der Seite. Das Block-Plugin zeigt die Koordinaten aber in der Notation von Acrobat an, das heißt, mit dem Ursprung in der linken oberen Ecke der Seite. Die Koordinaten werden in der Einheit angezeigt, die in Acrobat gerade ausgewählt ist. In der PDF-Datei werden sie immer in Punkt gespeichert.*
Status	*(Schlüsselwort) Beschreibt, wie der Block von PPS und der Vorschau-Funktion verarbeitet wird. (Standardwert:* `active`*):* ***active*** *Der Block wird entsprechend seiner Eigenschaften komplett verarbeitet.* ***ignore*** *Der Block wird ignoriert.* ***ignoredefault*** *Wie* `active`*, außer dass die Eigenschaften und Optionen* `defaulttext/image/pdf/graphics` *ignoriert werden, d.h. der Block bleibt leer, wenn keine variablen Inhalte verfügbar sind (vor allem in der Vorschau). Damit können Sie sicherstellen, dass die Block-Vorgabewerte nicht zum serverseitigen Füllen der Blöcke verwendet werden, obwohl der Block-Vorgabewerte zur Erzeugung der Vorschau enthalten kann. Es kann auch verwendet werden, um die Vorgabewerte für die Block-Vorschau zu deaktivieren, ohne die Vorgabewerte aus den Blockeigenschaften zu entfernen.* ***static*** *Es wird kein variabler Inhalt platziert, sondern der Vorgabewert für Text, Rasterbild, PDF oder Grafik wird verwendet, sofern vorhanden.*

12.6.3 Darstellungsspezifische Eigenschaften

Eigenschaften für die Darstellung legen Formatdetails fest:

- Tabelle 12.6 legt transparenzspezifische Eigenschaften fest.
- Tabelle 12.7 legt textspezifische Eigenschaften für Blöcke vom Typ Textline und Textflow fest.

Tabelle 12.6 Transparenzspezifische Eigenschaften für alle Blocktypen

Schlüsselwort	Mögliche Werte und Erklärung
blendmode	*(Liste von Schlüsselwörtern; Im PDF/A-1-Modus muss der Wert* `Normal` *verwendet werden) Mögliche Werte:* `None`, `Color`, `ColorDodge`, `ColorBurn`, `Darken`, `Difference`, `Exclusion`, `HardLight`, `Hue`, `Lighten`, `Luminosity`, `Multiply`, `None`, `Normal`, `Overlay`, `Saturation`, `Screen`, `SoftLight`. *Standard:* `None`
opacityfill	*(Float; Im PDF/A-1-Modus muss der Wert* `1` *verwendet werden) Deckkraft für das Füllen von Flächen im Bereich* `0`... `1`. *Der Wert* `0` *bedeutet vollständig transparent;* `1` *bedeutet völlig undurchsichtig.*
opacitystroke	*(Float; Im PDF/A-1- Modus muss der Wert* `1` *verwendet werden) Deckkraft für das Durchziehen von Linien im Bereich* `0`... `1`. *Der Wert* `0` *bedeutet vollständig transparent;* `1` *bedeutet völlig undurchsichtig.*

Tabelle 12.7 Textspezifische Eigenschaften für Blöcke vom Typ Textline und Textflow

Schlüsselwort	Mögliche Werte und Erklärung
charspacing	*(Float oder Prozentwert) Zeichenabstand. Prozentangaben beziehen sich auf* `fontsize`. *Standardwert:* `0`
decoration-above	*(Boolean) Mit true wird die mit* `underline`, `strikeout` *und* `overline` *aktivierte Textdekoration oberhalb des Textes gezeichnet, ansonsten unterhalb des Textes. Wird die Reihenfolge der Textausgabe geändert, wirkt sich dies auf die Sichtbarkeit der Dekorationslinien aus. Standard:* `false`
fillcolor	*(Color) Füllfarbe des Texts. Standardwert:* `gray 0` *(=schwarz)*
fontname[1]	*(String) Name der Schrift, so wie bei PDF_load_font( ) erforderlich. Das Block-Plugin zeigt alle installierten Systemschriften in einer Liste an. Beachten Sie jedoch, dass diese Fontnamen nicht unbedingt zwischen OS X, Windows und Unix portierbar sind. Wenn* `fontname` *mit einem @-Zeichen beginnt, wird der Font in vertikalem Textausgabemodus angewendet.* *Der Encoding für den Text muss beim Füllen der Blöcke als Option für PDF_fill_textblock( ) angegeben werden, es sei denn, die Option* `font` *wurde benutzt.*
fontsize[1]	*(Float) Schriftgröße in Punkt*
horizscaling	*(Float oder Prozentwert) Horizontale Skalierung von Text. Standardwert: 100%*
italicangle	*(Float) Neigungswinkel von kursivem Text in Grad. Standardwert: 0*
kerning	*(Boolean) Unterschneidung. Standardwert:* `false`
overline	*(Boolean) Modus für Überstreichen. Standardwert:* `false`
shadow	*(Composite) Schatteneffekt erzeugen (Standardwert:* `no shadow`*). Es gibt folgende Untereigenschaften:* **fillcolor** *(Color) Füllfarbe des Texts. Standardwert:* `{gray 0.8}` **offset** *(Liste von 2 Floats oder Prozentwerte) Der Schattenabstand vom Bezugspunkt des Textes angegeben in den Koordinaten des Benutzers oder als Prozentwert der Schriftgröße. Standardwert:* `{5% -5%}`
strikeout	*(Boolean) Modus für Durchstreichen. Standardwert:* `false`
strokecolor	*(Color) Farbe, in der Text gezeichnet wird. Standardwert:* `gray 0` *(= schwarz)*

Tabelle 12.7 Textspezifische Eigenschaften für Blöcke vom Typ Textline und Textflow

Schlüsselwort	Mögliche Werte und Erklärung
strokewidth	*(Float, Prozentwert oder Schlüsselwort; nur wirksam, wenn* `textrendering` *auf* `Umrisslinien zeichnen` *gesetzt ist) Linienstärke für Umrisslinien (in Benutzerkoordinaten oder als Prozentwert der Schriftgröße). Das Schlüsselwort* `auto` *und der gleichbedeutende Wert* `0` *verwenden einen eingebauten Standardwert. Standardwert:* `auto`
textrendering	*(Integer) Darstellungsmodus für Text. Nur der Wert 3 hat einen Einfluss auf Typ-3-Fonts (Standardwert:* `0`*):* **0**  *Text füllen** **1**  *Umrisslinien zeichnen** **2**  *Umrisslinien zeichnen, füllen** **3** *Unsichtbarer Text** **4**  *Text füllen, zum Beschneidungspfad hinzufügen** **5**  *Umrisslinien zeichnen und zum Beschneidungspfad hinzufügen** **6**  *Umrisslinien zeichnen, füllen und zum Beschneidungspfad hinzufügen** **7**  *Text zum Beschneidungspfad hinzufügen (nicht für Blöcke)*
textrise	*(Float oder Prozentwert) Wert für den vertikalen Textversatz. Prozentwerte beziehen sich auf* `fontsize`*. Standardwert:* `0`
underline	*(Boolean) Modus für Unterstreichen. Standardwert:* `false`
underline-position	*(Float, Prozentwert oder Schlüsselwort) Position der Linie für unterstrichenen Text, relativ zur Grundlinie. Prozentwerte beziehen sich auf* `fontsize`*. Standardwert:* `auto`
underline-width	*(Float, Prozentwert oder Schlüsselwort) Stärke der Linie für unterstrichenen Text. Prozentwerte beziehen sich auf* `fontsize`*. Standardwert:* `auto`
wordspacing	*(Float oder Prozentwert) Wortabstand. Prozentwerte beziehen sich auf* `fontsize`*. Standardwert:* `0`

1. Diese Eigenschaft wird für Textline- und Textflow-Blöcke benötigt; sie wird vom Block-Plugin unterstützt.

12.6.4 Eigenschaften zur Textvorbereitung

Eigenschaften zur Textvorbereitung legen Vorverarbeitungsschritte für Textline- und Textflow-Blöcke fest. Tabelle 12.8 gibt eine Übersicht.

Table 12.8 Eigenschaften zur Textvorbereitung für Blöcke vom Typ Textline und Textflow

Schlüsselwort	Mögliche Werte und Erklärung
charref	*(Boolean) Mit* true *wird die Ersetzung von numerischen oder Character-Referenzen und Glyphnamen-Referenzen aktiviert. Standardwert: globale Option* charref
escape-sequence	*(Boolean) Mit* true *wird die Ersetzung von Escape-Sequenzen in Content-Strings, Hypertext-Strings und Name-Strings aktiviert. Standardwert: globale Option* escapesequence
features	*(Liste von Schlüsselwörtern) Bestimmt, welche typografischen Features eines OpenType-Fonts auf den Text angewendet werden, entsprechend der Optionen* script *und* language. *Schlüsselwörter für im Font nicht vorhandene Features werden ignoriert. Die folgenden Schlüsselwörter können übergeben werden:* **_none** *Kein Feature auf den Font anwenden. Als Ausnahme muss das Feature* vert *explizit mit dem Schlüsselwort* novert *deaktiviert werden.* **<name>** *Aktivierung eines Features durch Angabe seiner vier Zeichen langen OpenType-Bezeichnung. Einige häufig benutzte Feature-Namen sind* liga, ital, tnum, smcp, swsh, zero. *Die vollständige Liste der Namen und Beschreibungen aller unterstützten Features finden Sie in Abschnitt 6.3.1, »Unterstützte OpenType-Layoutfunktionen«, Seite 160.* ***no<name>*** *Das Präfix* no *vor einem Feature-Namen (z.B.* noliga*) deaktivert dieses Feature.* *Standardwert:* _none *für horizontalen Textausgabemodus. Im vertikalen Textausgabemodus wird automatisch* vert *verwendet.* *Für die Unterstützung von OpenType-Features ist die Option* readfeatures *in PDF_load_font( ) erforderlich.*
language	*(Schlüsselwort; nur wenn* script *übergeben wird) Der Text wird bezüglich der angegebenen Sprache verarbeitet, was für die Optionen* features *und* shaping *relevant ist. Eine vollständige Liste aller Schlüsselwörter finden Sie in Abschnitt 6.4.2, »Schrift und Sprache«, Seite 169, zum Beispiel* ARA *(Arabisch),* JAN *(Japanisch),* HIN *(Hindi). Standardwert:* _none *(Sprache nicht definiert)*
script	*(Schlüsselwort; erforderlich, falls* shaping=true*) Der Text wird bezüglich des verwendeten Schriftsystems verarbeitet, das für die Optionen* features, shaping *und* advancedlinebreaking *festgelegt wurde. Die häufigsten Schlüsselwörter für Schriftsysteme sind:* _none *(Schriftsystem nicht definiert),* latn, grek, cyrl, armn, hebr, arab, deva, beng, guru, gujr, orya, taml, thai, laoo, tibt, hang, kana, han. *Das Schlüsselwort* _auto *wählt das Schriftsystem, zu dem die meisten Zeichen im Text gehören, wobei* _latn *und* _none *ignoriert werden. Eine vollständige Liste aller Schlüsselwörter finden Sie in Abschnitt 6.4.2, »Schrift und Sprache«, Seite 169.Standardwert:* _none
shaping	*(Boolean) Mit* true *wird der Text entsprechend der Optionen* script *und* language *formatiert (shaped). Die Option* script *muss einen Wert ungleich* _none *haben und die erforderlichen Shaping-Tabellen müssen im Font vorhanden sein. Standardwert:* false

12.6.5 Eigenschaften für die Textformatierung

Tabelle 12.9 gibt eine Übersicht über Eigenschaften, die nur mit Textflow-Blöcken verwendet werden können, mit Ausnahme der Eigenschaft *stamp*, die auch mit Textline-Blöcken verwendet werden kann. Anhand dieser Eigenschaften wird die anfängliche Optionsliste zur Textflow-Verarbeitung zusammengestellt (entsprechend dem Parameter *optlist* für *PDF_create_textflow( )*). Mit dem Block-Plugin können keine Inline-Optionslisten für Textflow festgelegt werden. Diese lassen sich jedoch auf dem Server als Bestandteil des Textinhalts übergeben, wenn der Block mit *PDF_fill_textblock( )* gefüllt wird. Auch der Vorgabetext in der Eigenschaft *defaulttext* kann Inline-Optionen enthalten.

Tabelle 12.9 Eigenschaften für die Textformatierung (fast alle nur für Textflow-Blöcke)

Schlüsselwort	Mögliche Werte und Erklärung
adjustmethod	*(Schlüsselwort) Methode zur Anpassung von Textteilen, die nach einer Vergrößerung oder Verkleinerung des Wortabstands* `minspacing` *und* `maxspacing` *nicht mehr in die Zeile passen. (Standardwert:* `auto`*):*
	auto *Folgende Methoden werden in der angeführten Reihenfolge angewandt:* `shrink`, `spread`, `nofit`, `split`.
	clip *Wie* `nofit`*, nur dass der längere Teil am rechten Rand der Fitbox (unter Berücksichtigung der Option* `rightindent`*) abgeschnitten wird.*
	nofit *Das letzte Wort wird in die nächste Zeile verschoben, sofern die verbleibende (kurze) Zeile nicht kürzer als der in der Option* `nofitlimit` *festgelegte Prozentwert ist. Auch Absätze im Blocksatz sehen bei dieser Methode leicht ausgefranst aus.*
	shrink *Passt ein Wort nicht in die Zeile, wird der Text so lange gestaucht, bis das Wort hineinpasst, sofern die Option* `shrinklimit` *dies zulässt. Anderenfalls kommt die Methode* `nofit` *zum Einsatz.*
	split *Das letzte Wort wird nicht in die nächste Zeile verschoben, sondern zwangsweise getrennt. Bei Textfonts (nicht aber bei Symbolfonts) wird ein Trennzeichen eingefügt.*
	spread *Das letzte Wort wird in die nächste Zeile verschoben. Der Rest der (kurzen) Zeile wird im Blocksatz ausgerichtet, indem der Zeichenabstand innerhalb der Wörter vergrößert wird, sofern die Option* `spreadlimit` *dies zulässt. Kann kein Blocksatz erreicht werden, kommt die Methode* `nofit` *zum Einsatz.*
advancedlinebreak	*(Boolean) Aktiviert den erweiterten Zeilenumbruch-Algorithmus, der für den Zeilenumbruch in komplexen Schriftsystemen erforderlich ist, die keine Leerzeichen zur Kennzeichnung von Wortgrenzen verwenden, z.B. Thai. Die Optionen* `locale` *und* `scripts` *werden berücksichtigt. Standardwert:* `false`
alignment	*(Schlüsselwort) Legt die Formatierung für die Zeilen eines Absatzes fest. Standardwert:* `left`.
	left *linksbündig, beginnend bei* `leftindent`
	center *mittig zwischen* `leftindent` *und* `rightindent`
	right *rechtsbündig, bei* `rightindent` *endend*
	justify *links- und rechtsbündig (Blocksatz)*
avoidemptybegin	*(Boolean) Mit* `true` *werden Leerzeilen am Anfang einer Fitbox gelöscht. Standardwert:* `false`
fixedleading	*(Boolean) Ist* `fixedleading` *gleich* `true`*, wird der am Zeilenanfang geltende Zeilenabstand verwendet. Andernfalls wird der größte Zeilenabstand verwendet. Standardwert:* `false`

Tabelle 12.9 Eigenschaften für die Textformatierung (fast alle nur für Textflow-Blöcke)

Schlüsselwort	Mögliche Werte und Erklärung
hortab-method	*(Schlüsselwort) Legt die Interpretation von horizontalen Tabulatoren im Text fest. Liegt die berechnete Position links der aktuellen Textposition, so wird der Tabulator ignoriert. (Standardwert:* relative*):* **relative** *Die Position wird um den in* hortabsize *festgelegten Betrag vorgerückt.* **typewriter** *Die Position wird auf das nächste Vielfache von* hortabsize *vorgerückt.* **ruler** *Die Position wird auf den n-ten in der Option* ruler *verfügbaren Tabulatorwert gesetzt, wobei* n *die Anzahl der bislang in der Textzeile vorgekommenen Tabs bezeichnet. Ist* n *größer als die Anzahl der in* ruler *verfügbaren Tabulatorpositionen, kommt die Methode* relative *zum Einsatz.*
hortabsize	*(Float oder Prozentwert) Legt die Breite eines horizontalen Tabulators fest[1]. Die Interpretation wird von der Option* hortabmethod *gesteuert. Standardwert: 7.5%*
last-alignment	*(Schlüsselwort) Bestimmt die Formatierung der letzten Zeile eines Absatzes. Neben allen Schlüsselwörtern der Option* alignment *gibt es folgendes Schlüsselwort (Standardwert:* auto*):* **auto** *Es wird der Wert der Option* alignment *verwendet. Nur bei* justify *wird* left *verwendet.*
leading	*(Float oder Prozentwert) Zeilenabstand in Benutzerkoordinaten oder als prozentualer Anteil der Schriftgröße. Standardwert: 100%*
locale	*(Schlüsselwort) Bestimmt die Spracheinstellung für lokalisierte Zeilenumbruch-Methoden, wenn* advancedlinebreak=true*. Das Schlüsselwort besteht aus einer oder mehreren Komponenten, wobei die optionalen Komponenten durch einen Unterstrich '_' getrennt sind. (Die Syntax unterscheidet sich geringfügig von NLS/POSIX locale IDs):* ► *Ein erforderlicher Sprachcode aus zwei oder drei Kleinbuchstaben nach ISO 639-2 (siehe* www.loc.gov/standards/iso639-2*), z. B.* en*, (englisch),* de *(Deutsch),* ja *(Japanisch). Dieser unterscheidet sich von der Option* language*.* ► *Eine optionaler vierbuchstabiger Schriftsystem-Code nach ISO 15924 (siehe* www.unicode.org/iso15924/iso15924-codes.html*), z. B.* Hira *(Hiragana),* Hebr *(Hebräisch),* Arab *(Arabisch),* Thai *(Thai).* ► *Ein optionaler Ländercode aus zwei Großbuchstaben nach ISO 3166 (siehe* www.iso.org/iso/country_codes/iso_3166_code_lists*), z. B.* DE *(Deutschland),* CH *(Schweiz),* GB *(Großbritannien)* *Für den erweiterten Zeilenumbruch ist keine Spracheinstellung erforderlich: mit dem Schlüsselwort* _none *wird keine Sprache eingestellt. Standardwert:* _none *Beispiele:* de_DE, en_US, en_GB
maxspacing minspacing	*(Float oder Prozentwert) Bestimmt den maximalen bzw. minimalen Abstand zwischen Wörtern (in Benutzerkoordinaten oder als prozentualer Anteil der Breite eines Leerzeichens). Der berechnete Wortabstand wird durch die hier übergebenen Werte begrenzt, aber der Wert der Option* wordspacing *wird noch addiert. Standardwerte:* minspacing=50%*,* maxspacing=500%
minlinecount	*(Integer) Mindestanzahl von Zeilen im letzten Absatz der Fitbox. Wenn es weniger Zeilen sind, werden sie in die nächste Fitbox verschoben. Mit dem Wert* 2 *werden einzelne Zeilen eines Absatzes am Ende einer Fitbox verhindert (»Schusterjunge«). Standard:* 1
nofitlimit	*(Float oder Prozentwert) Minimale Zeilenlänge bei der Methode* nofit *(in Benutzerkoordinaten oder als prozentualer Anteil der Fitbox-Breite). Standardwert:* 75%
parindent	*(Float oder Prozentwert) Legt den linken Einzug der ersten Zeile eines Absatzes fest[1]. Der Wert wird zu* leftindent *addiert. Wird diese Option innerhalb der Zeile angegeben, so wirkt sie wie ein Tabulator. Standardwert: 0*
rightindent leftindent	*(Float oder Prozentwert) Bestimmt den rechten bzw. linken Einzug aller Textzeilen[1]. Wird* leftindent *innerhalb der Zeile angegeben und befindet sich die definierte Position links der aktuellen Textposition, so wird die Option für die aktuelle Zeile ignoriert. Standardwert: 0*
ruler[2]	*(Liste von Floats oder Prozentwerten) Liste der absoluten Tabulatorpositionen für* hortabmethod=ruler*[1]. Die Liste darf maximal 32 nicht negative Einträge in aufsteigender Reihenfolge enthalten. Standardwert: Vielfache von* hortabsize *als Integers*

Tabelle 12.9 Eigenschaften für die Textformatierung (fast alle nur für Textflow-Blöcke)

Schlüsselwort	Mögliche Werte und Erklärung
shrinklimit	*(Prozentwert) Untere Grenze für das Stauchen von Text mit der Methode* `shrink`*. Der berechnete Stauchungsfaktor wird durch den hier übergebenen Wert begrenzt, aber noch mit dem Wert der Option* `horizscaling` *multipliziert. Standardwert:* `85%`
spreadlimit	*(Float oder Prozentwert) Obere Grenze für den Abstand zwischen zwei Zeichen bei der Methode* `spread` *(in Benutzerkoordinaten oder als prozentualer Anteil der Schriftgröße); der berechnete Zeichenabstand wird durch den hier übergebenen Wert begrenzt, aber der Wert der Option* `charspacing` *wird noch addiert. Standardwert:* `0`
stamp	*(Schlüsselwort; Textline- und Textflow-Blöcke) Mit dieser Option lässt sich ein diagonaler Stempel im Blockrechteck erzeugen. Der Stempeltext wird dabei so groß wie möglich gesetzt. Bei der Positionierung des Stempeltexts in der Box werden die Optionen* `position`*,* `fitmethod` *und* `orientate` *(nur* `north` *und* `south`*) berücksichtigt. (Standardwert:* `none`*):* **ll2ur** *Der Stempel verläuft diagonal von der linken unteren zur rechten oberen Ecke.* **ul2lr** *Der Stempel verläuft diagonal vom der linken oberen zur rechten unteren Ecke.* **none** *Es wird kein Stempel erzeugt.*
tabalignchar	*(Integer) Unicode-Wert des Zeichens, an dem dezimale Tabulatoren ausgerichtet werden. Standardwert: das Zeichen '.' (*`U+002E`*)*
tabalignment[2]	*(Liste aus Schlüsselwörtern) Ausrichtung für Tabulatoren. Jeder Listeneintrag definiert die Ausrichtung des entsprechenden Eintrags in der Option* `ruler`*. Standardwert:* `left`*.* **center** *Text wird mittig an der Tabulatorposition ausgerichtet.* **decimal** *Das erste* `tabalignchar`*-Zeichen wird linksbündig an der Tabulatorposition ausgerichtet. Ist kein* `tabalignchar`*-Zeichen vorhanden, wird rechtsbündig ausgerichtet.* **left** *Text wird linksbündig an der Tabulatorposition ausgerichtet.* **right** *Text wird rechtsbündig an der Tabulatorposition ausgerichtet.*

1. In Benutzerkoordinaten oder als prozentualer Anteil an der Breite der Fitbox
2. Die Tabulatorpositionen werden im Blockeigenschaften-Dialog in der Eigenschaftengruppe `Tabulatoren für Textflow für hortabmethod=ruler` *festgelegt.*

12.6.6 Eigenschaften für die Objekteinpassung

Eigenschaften für die Objekteinpassung stehen für alle Blocktypen zur Verfügung, obwohl einige Eigenschaften nur auf bestimmte Blocktypen angewendet werden können. Sie steuern, wie Inhalte in die Blöcke eingepasst werden.

- Tabelle 12.10 gibt eine Übersicht über Einpassungseigenschaften für Blöcke vom Typ Textline, Image, PDF und Graphics.
- Tabelle 12.11 gibt eine Übersicht über Einpassungseigenschaften für Textflow-Blöcke (die meisten beziehen sich auf vertikale Einpassung).

Der Algorithmus für die Objekteinpassung verwendet das Block-Rechteck als Fitbox. Außer bei *fitmethod=clip* wird der Text nicht beschnitten. Um sicherzugehen, dass der Blockinhalt nicht über das Blockrechteck hinausgeht, vermeiden Sie die Methode *fitmethod=nofit*.

Tabelle 12.10 Einpassungseigenschaften für Blöcke vom Typ Textline, Image, PDF und Graphics

Schlüsselwort	Mögliche Werte und Erklärung
alignchar	*(Unichar oder Schlüsselwort) Ist das hier definierte Zeichen im Text vorhanden, wird dessen linke untere Ecke am Referenzpunkt ausgerichtet. Bei horizontalem Text mit* `orientate=north` *oder* `south` *definiert der erste in der Option* `position` *übergebene Wert die Position. Bei horizontalem Text mit* `orientate=west` *oder* `east` *dagegen definiert der zweite in der Option* `position` *übergebene Wert die Position. Diese Option wird ignoriert, wenn das hier definierte Zeichen im Text nicht vorhanden ist. Der Wert* `0` *und das Schlüsselwort* `none` *bedeuten, dass kein Ausrichtungszeichen festgelegt wird. Es wird die definierte* `fitmethod` *angewandt, selbst wenn der Text wegen der Ausrichtung nach* `alignchar` *nicht innerhalb der Fitbox platziert werden kann. Standardwert:* `none`
dpi	*(Float-Liste; nur für Blocktyp Image) Einer oder zwei Werte, die die gewünschte Bildauflösung in Pixel pro Zoll in horizontaler und vertikaler Richtung angeben. Ist der Wert gleich* `0`*, wird die interne Bildauflösung verwendet. Ist diese nicht vorhanden, werden 72 dpi benutzt. Diese Eigenschaft wird ignoriert, wenn die Eigenschaft* `fitmethod` *mit einem der Schlüsselwörter* `auto`, `meet`, `slice` *oder* `entire` *übergeben wurde. Standardwert: 0*
fitmethod	*(Schlüsselwort) Strategie für den Fall, dass der übergebene Inhalt nicht in die Box passt. Mögliche Werte sind* `auto`, `nofit`, `clip`, `meet`, `slice` *und* `entire`*. Für Blöcke vom Typ Textline, Image, PDF oder Graphics wird diese Eigenschaft gemäß Standardinterpretation interpretiert. Standardwert:* `meet`*. Für Textflow-Blöcke, die zu schmal für den Text sind, lautet die Interpretation wie folgt:* **auto** `fontsize` *und* `leading` *werden so lange reduziert, bis der Text passt.* **nofit** *Der Text läuft aus dem unteren Blockrand hinaus.* **clip** *Der Text wird am Blockrand abgeschnitten.*
margin	*(Float-Liste; nur für Blocktyp Textline) Ein oder zwei Float-Werte für eine zusätzliche horizontale und vertikale Verkleinerung des Block-Rechtecks. Standardwert: 0*
orientate	*(Schlüsselwort) Legt fest, in welcher Ausrichtung der Inhalt platziert wird. Mögliche Werte sind* `north`, `east`, `south`, `west`*. Standardwert:* `north`
position	*(Float-Liste) Ein oder zwei Werte, die die Position des Referenzpunkts innerhalb des Inhalts festlegen. Die Position wird als Prozentwert innerhalb des Blocks angegeben. Nur für Blöcke vom Typ Textline: Das Schlüsselwort* `auto` *kann für den ersten Wert in der Liste verwendet werden. Es bedeutet* `rechts`*, wenn die Schreibrichtung von rechts nach links ist (z.B. in arabischen oder hebräischen Texten) und ansonsten* `links` *(z.B. für Texte in lateinischer Schrift).* *Standardwert:* `{0 0}`*, d.h. die linke untere Ecke*
rotate	*(Float) Drehwinkel in Grad, um den der Block gegen den Uhrzeigersinn gedreht wird, bevor die Verarbeitung beginnt. Das Rotationszentrum ist der Referenzpunkt. Standardwert: 0*

Tabelle 12.10 Einpassungseigenschaften für Blöcke vom Typ Textline, Image, PDF und Graphics

Schlüsselwort	Mögliche Werte und Erklärung
scale	*(Float-Liste) Einer oder zwei Werte, die den oder die gewünschten Skalierungsfaktoren in horizontaler und vertikaler Richtung festlegen. Diese Eigenschaft wird ignoriert, wenn die Eigenschaft* `fitmethod` *mit einem der Schlüsselwörter* `auto`, `meet`, `slice` *oder* `entire` *übergeben wurde. Standardwert:* `1`
shrinklimit	*(Float oder Prozentwert; nur für Blocktyp Textline) Untere Grenze für das Stauchen von Text mit der Methode* `fitmethod=auto`. *Standardwert: 0,75*

.

Tabelle 12.11 Einpassungseigenschaften für Textflow-Blöcke

Schlüsselwort	Mögliche Werte und Erklärung
firstlinedist	*(Float, Prozentwert oder Schlüsselwort) Abstand zwischen dem oberen Rand der Fitbox und der Grundlinie der ersten Textzeile. Angegeben wird er in Benutzerkoordinaten, als Prozentsatz der Schriftgröße (der Größe der ersten in der Zeile auftretenden Schrift, wenn* `fixedleading=true`*, oder der maximal auftretenden Schriftgröße andernfalls) oder als Schlüsselwort. (Standardwert:* `leading`*):* **leading** *Für die erste Zeile ermittelter Zeilenabstand; diakritische Zeichen wie À berühren den oberen Rand der Fitbox.* **ascender** *Für die erste Zeile ermittelte Oberlänge; Zeichen mit großer Oberlänge wie d oder h berühren den oberen Rand der Fitbox.* **capheight** *Für die erste Zeile ermittelte Versalhöhe; hohe Großbuchstaben wie H berühren den oberen Rand der Fitbox.* **xheight** *Für die erste Zeile ermittelte x-Höhe; Kleinbuchstaben wie x berühren den oberen Rand der Fitbox.* *Ist* `fixedleading=false`*, wird der größte Wert verwendet, der für Zeilenabstand, Oberlänge, x-Höhe und Versalhöhe in der ersten Zeile ermittelt wurde.*
fitmethod	*(Schlüsselwort) Strategie für den Fall, dass der übergebene Inhalt nicht in die Box passt. Mögliche Werte sind* `auto`, `nofit`, `clip`. *Standardwert:* `auto`. *Für Textflow-Blöcke, die zu klein für den Text sind, lautet die Interpretation wie folgt:* **auto** `fontsize` *und* `leading` *werden so lange reduziert, bis der Text passt.* **nofit** *Der Text läuft aus dem unteren Blockrand hinaus.* **clip** *Der Text wird am Blockrand abgeschnitten.*
lastlinedist	*(Float, Prozentwert oder Schlüsselwort; wird ignoriert bei* `fitmethod=nofit`*) Der kleinste Abstand zwischen der Grundlinie der letzten Textzeile und dem unteren Rand der Fitbox. Angegeben wird er in Benutzerkoordinaten, als Prozentsatz der Schriftgröße (der ersten in der Zeile auftretenden Schriftgröße, wenn* `fixedleading=true` *oder andernfalls der maximal in der Zeile auftretenden Schriftgröße) oder als Schlüsselwort. Standardwert:* `0`*, d.h. der untere Rand der Fitbox wird als Grundlinie verwendet und die üblichen Unterlängen reichen aus der Fitbox hinaus.* **descender** *Für die letzte Zeile ermittelte Unterlänge; Zeichen mit Unterlängen wie* `g` *oder* `j` *berühren dabei den unteren Rand der Fitbox.* *Ist* `fixedleading=false` *wird der größte Wert verwendet, der in der letzten Zeile für die Unterlänge ermittelt wurde.*
linespreadlimit	*(Float oder Prozentwert; nur für* `verticalalign=justify`*) Größter Wert in Benutzerkoordinaten oder als Prozentsatz des Zeilenabstands, um den der Zeilenabstand bei vertikaler Ausrichtung erhöht wird. Standardwert: 200%*
maxlines	*(Integer oder Schlüsselwort) Maximale Anzahl der Zeilen in der Fitbox oder das Schlüsselwort* `auto`*, bei dem möglichst viele Zeilen in der Fitbox platziert werden. Nach der Platzierung der maximalen Anzahl von Zeilen gibt PDF_fit_textflow( ) den String* `_boxfull` *zurück.*

Tabelle 12.11 Einpassungseigenschaften für Textflow-Blöcke

Schlüsselwort	Mögliche Werte und Erklärung
orientate	*(Schlüsselwort) Legt fest, in welcher Ausrichtung der Text platziert wird. Mögliche Werte sind* `north`, `east`, `south`, `west`. *Standardwert:* `north`
minfontsize	*(Float oder Prozentwert) Mindestschriftgröße, in der der Text mit* `fitmethod=auto` *bei überschrittenem* `shrinklimit` *erstellt werden darf. Der Grenzwert wird in Benutzerkoordinaten oder als Prozentsatz der Blockhöhe angegeben. Wenn der Grenzwert erreicht ist, wird der Text mit der Schriftgröße* `minfontsize` *erstellt. Standard: 0,1%*
rotate	*(Float) Dreht das Koordinatensystem, mit der linken unteren Ecke der Fitbox als Mittelpunkt. Der übergebene Wert legt den Drehwinkel in Grad fest. Text und Box werden gedreht. Die Drehung wird zurückgesetzt, nachdem der Text platziert wurde. Standardwert: 0*
verticalalign	*(Schlüsselwort) Vertikale Ausrichtung des Texts in der Fitbox (Standardwert:* `top`*):* **top** *Die Formatierung beginnt in der ersten Zeile und setzt sich nach unten fort. Füllt der Text die Fitbox nicht vollständig aus, bleibt Weißraum unter dem Text.* **center** *Der Text wird vertikal in der Fitbox zentriert. Füllt der Text die Fitbox nicht vollständig aus, bleibt Weißraum über und unter dem Text.* **bottom** *Die Formatierung beginnt in der letzten Zeile und setzt sich nach oben fort. Füllt der Text die Fitbox nicht vollständig aus, bleibt Weißraum über dem Text.* **justify** *Der Text wird am oberen und unteren Rand der Fitbox ausgerichtet. Dazu wird der Zeilenabstand bis zur durch* `linespreadlimit` *festgelegten Grenze erhöht. Die Höhe der ersten Zeile wird nur bei* `firstlinedist=leading` *vergrößert.*

12.6.7 Vorgabewerte

Eigenschaften für Vorgabewerte definieren den Blockinhalt, wenn keine spezifischen Inhalte bereitgestellt werden. Sie sind besonders nützlich für die Vorschau-Funktion, da die Blöcke dann mit ihren Vorgabewerten gefüllt werden. Tabelle 12.12 gibt eine Übersicht.

Tabelle 12.12 Blockeigenschaften für Vorgabewerte

Schlüsselwort	Mögliche Werte und Erklärung
defaultgraphics	*(String; nur für Blocktyp Graphics) Pfadname der Grafikdatei, die verwendet wird, wenn vom Client keine Grafik übergeben wird.*[1]
defaultimage	*(String; nur für Blocktyp Image) Pfadname des Bildes, das verwendet wird, wenn vom Client kein Bild übergeben wird.*[1]
defaultpdf	*(String; nur für Blocktyp PDF) Pfadname des PDF-Dokuments, das verwendet wird.*[1]
defaultpdfpage	*(Integer; nur für Blocktyp PDF) Nummer der Seite im Vorgabe-PDF-Dokument. Standardwert: 1*
defaulttext	*(String; nur für Blocktyp Textline und Textflow) Text, der verwendet wird, wenn vom Client kein Ersatztext übergeben wird*[2]*.*

1. Dateinamen sollten ohne absolute Pfadangaben verwendet werden; stattdessen sollte in der PPS-Client-Anwendung mit der Funktionalität `SearchPath` *gearbeitet werden. Dies macht die Blockverarbeitung unabhängig von der Plattform und den dateisystemspezifischen Eigenheiten.*
2. Text wird im Zeichensatz `winansi` *oder Unicode interpretiert.*

12.6.8 Benutzerdefinierte Eigenschaften

Benutzerdefinierte Eigenschaften können für Blöcke beliebigen Typs definiert werden und werden von PPS und der Vorschau-Funktion ignoriert. Tabelle 12.13 gibt eine Übersicht über die Namensregeln für benutzerdefinierte Eigenschaften.

Table 12.13 Benutzerdefinierte Blockeigenschaften für alle Blocktypen

Schlüsselwort	***Mögliche Werte und Erklärung***
beliebiger Name, der die drei Zeichen [ ] / nicht enthalten darf	*(String, Name, Float oder Float-Liste) Die Interpretation der Werte benutzerdefinierter Eigenschaften liegt vollständig bei der Client-Applikation; sie werden von PPS ignoriert.*

12.7 Abfragen von Blocknamen und -eigenschaften mit pCOS

Neben automatischer Blockverarbeitung mit PPS kann die integrierte pCOS-Schnittstelle genutzt werden, um Blocknamen aufzulisten sowie Standard- und benutzerdefinierte Eigenschaften abzufragen.

Cookbook *Ein vollständiges Codebeispiel zur Abfrage der Eigenschaften von Blöcken, die in einem importierten PDF-Dokument enthalten sind, finden Sie im Cookbook-Topic* `blocks/query_block_properties`.

Ermitteln der Anzahl und Namen von Blöcken. Die Namen und die Anzahl der Blöcke auf einer importierten Seite brauchen dem Clientcode nicht bekannt zu sein, da sie abgefragt werden können. Die folgende Anweisung gibt die Anzahl der Blöcke auf der Seite *pagenum* zurück:

```
blockcount = (int) p.pcos_get_number(doc, "length:pages[" + pagenum + "]/blocks");
```

Die folgende Anweisung gibt den Namen von Block Nummer *blocknum* auf der Seite *pagenum* (die Zählung von Seiten und Blöcken beginnt bei 0):

```
blockname = p.pcos_get_string(doc,
        "pages[" + pagenum + "]/blocks[" + blocknum + "]/Name");
```

Der zurückgegebene Blockname kann im weiteren Verlauf zur Abfrage von Blockeigenschaften oder zum Füllen des Blocks mit Text, Bild, PDF- oder Grafik-Inhalt verwendet werden. Existiert der angegebene Block nicht, wird eine Exception ausgelöst. Um dies zu vermeiden, können Sie die Anzahl der Blöcke und damit den höchsten Index im Array *blocks* mit dem Präfix *length* ermitteln. Beachten Sie, dass die Blockanzahl um eins höher ist als der größtmögliche Index, da die Array-Indizierung bei 0 beginnt.

In der Syntax für den Pfad zur Adressierung der Blockeigenschaft sind folgende Ausdrücke gleichbedeutend, wobei davon ausgegangen wird, dass der Block mit der Nummer <*nummer*> seine Eigenschaft *Name* auf <*blockname*> gesetzt hat:

```
pages[...]/blocks[<nummer>]
pages[...]/blocks/<blockname>
```

Ermitteln von Blockkoordinaten. Die beiden Koordinatenpaare *(llx, lly)* und *(urx, ury)*, die die linke untere und die rechte obere Ecke eines Blocks namens *foo* beschreiben, lassen sich wie folgt abfragen:

```
llx = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/Rect[0]");
lly = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/Rect[1]");
urx = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/Rect[2]");
ury = p.pcos_get_number(doc, "pages[" + pagenum + "]/blocks/foo/Rect[3]");
```

Beachten Sie, dass diese Koordinaten im Standard-Koordinatensystem (mit dem Ursprung in der linken unteren Ecke, eventuell modifiziert durch die CropBox der Seite) übergeben werden, wohingegen das Block-Plugin die Koordinaten gemäß des Koordinatensystems der Acrobat-Benutzeroberfläche mit dem Ursprung in der linken oberen Ecke der Seite anzeigt. Da die Option *Rect* zum Überschreiben von Blockkoordinaten keine durch den CropBox-Eintrag gegebenen Änderungen berücksichtigt, können die vom

Originalblock abgefragten Koordinaten nicht direkt als neue Koordinaten verwendet werden, falls eine CropBox vorhanden ist. Als Abhilfe können Sie mit den Optionen *refpoint* und *boxsize* arbeiten.

Beachten Sie zudem, dass die Option *topdown* bei der Abfrage der Blockkoordinaten nicht berücksichtigt wird.

Abfrage von benutzerdefinierten Eigenschaften. Benutzerdefinierte Eigenschaften lassen sich wie in folgendem Beispiel abfragen, in dem die Eigenschaft *zipcode* von einem Block namens *b1* auf der Seite *pagenum* abgefragt wird.

```
zip = p.pcos_get_string(doc, "pages[" + pagenum + "]/blocks/b1/Custom/zipcode");
```

Wenn Sie nicht wissen, welche benutzerdefinierten Eigenschaften in einem Block eigentlich vorhanden sind, können Sie deren Namen zur Laufzeit abfragen. Mit folgendem Codefragment ermitteln Sie zum Beispiel den Namen der ersten benutzerdefinierten Eigenschaft des Blocks *b1:*

```
propname = p.pcos_get_string(doc, "pages[" + pagenum + "]/blocks/b1/Custom[0].key");
```

Wenn Sie den Index 0 erhöhen, erhalten Sie sukzessive die Namen aller weiteren benutzerdefinierten Eigenschaften. Mit dem Präfix *length* erhalten Sie die Anzahl der benutzerdefinierten Eigenschaften.

Nicht vorhandene Blockeigenschaften und Standardwerte. Mit dem Präfix *type* ermitteln Sie, ob ein Block oder eine Eigenschaft tatsächlich vorhanden sind. Ist der Typ eines Pfads gleich 0 oder *null*, so fehlt das entsprechende Objekt im PDF-Dokument. Beachten Sie, dass dann bei vordefinierten Eigenschaften der jeweilige Standardwert verwendet wird.

Namensraum für benutzerdefinierte Eigenschaften. Zur Vermeidung von Namenskonflikten beim Austausch von PDF-Dokumenten aus verschiedenen Quellen sollten die Namen benutzerdefinierter Eigenschaften aus einem firmeneigenen Präfix, gefolgt von einem Doppelpunkt ':' und dem eigentlichen Eigenschaftsnamen bestehen. Als firmenspezifisches Präfix empfehlen wir den jeweiligen Internet-Domainnamen. Die Property-Namen des Unternehmens ACME sähen dann zum Beispiel wie folgt aus:

```
acme.com:digits
acme.com:refnumber
```

Da Standard- und benutzerdefinierte Eigenschaften im Block unterschiedlich gespeichert werden, können PPS-Standardnamen (wie in Abschnitt 12.6, »Blockeigenschaften«, Seite 390 definiert) nie mit benutzerdefinierten Namen in Konflikt geraten.

12.8 Blocks per Programm erzeugen und importieren

12.8.1 Erzeugen von PDFlib-Blöcken mit POCA

PDFlib-Blöcke können per Programm mit der POCA-Schnittstelle erstellt werden, die in PPS enthalten ist. Mit POCA können die erforderlichen PDF-Datenstrukturen für Blöcke vorbereitet werden und dann an die Option *blocks* von *PDF_begin/end_page_ext( )* übergeben werden. Bei der Erstellung der Blockdefinitionen müssen Sie bestimmte Bedingungen befolgen, siehe Abschnitt 12.9, »Spezifikation für PDFlib-Blöcke«, Seite 406. Die Blockeigenschaften müssen gemäß der aufgeführten Datentypen erstellt werden, siehe Abschnitt 12.6, »Blockeigenschaften«, Seite 390.

Cookbook *Ein vollständiges Codebeispiel zur Erstellung der PDFlib-Blöcke mit PPS finden Sie in der Cookbook-Kategorie* `block-handling-and-pps`.

In der PDFlib Block-Spezifikation sind die Blöcke leider doppelt enthalten: einmal im Haupt-Block-Dictionary einer Seite, und erneut unter dem Eintrag *Name* innerhalb eines bestimmten Block-Dictionary. Diese beiden Namen müssen identisch sein, um Probleme beim Füllen des Blocks mit PPS oder beim Anzeigen der Blockvorschau mit dem Block-Plugin zu vermeiden. Durch *PDF_begin/end_page_ext( )* wird daher eine Exception ausgelöst, wenn das mit der Option *blocks* bereitgestellte Dictionary eine Blockdefinition enthält, die diese Regel verletzt. Im folgenden Codebeispiel sind die entsprechenden Paare in blau hervorgehoben.

Der folgende Code-Auszug zeigt die Verwendung der POCA-Funktionen zur Erzeugung der Blockdefinition gemäß Abschnitt , »Schlüssel in Block-Dictionaries«, Seite 407.

```
/* Block-Dictionary erzeugen */
blockdict = p.poca_new("containertype=dict usage=blocks");

/* ------------------------------------------------------
 * Textblock erzeugen
 * ------------------------------------------------------
 */
textblock = p.poca_new("containertype=dict usage=blocks type=name key=Type value=Block");

container1 = p.poca_new("containertype=array usage=blocks " +
        "type=integer values={70 640 300 700}");

p.poca_insert(textblock, "type=array key=Rect value=" + container1);
p.poca_insert(textblock, "type=name  key=Name value=job_title");
p.poca_insert(textblock, "type=name  key=Subtype value=Text");
p.poca_insert(textblock, "type=name  key=fitmethod value=auto");
p.poca_insert(textblock, "type=name  key=fontname value=Helvetica");
p.poca_insert(textblock, "type=float key=fontsize value=12");

/* Block im Block-Dictionary der Seite aufnehmen */
p.poca_insert(blockdict, "type=dict key=job_title direct=false value=" + textblock);

/* ------------------------------------------------------
 * Block vom Typ Image erzeugen
 * ------------------------------------------------------
 */
imageblock = p.poca_new("containertype=dict usage=blocks " +
        "type=name key=Type value=Block");
```

```
container2 = p.poca_new("containertype=array usage=blocks " +
        "type=integer values={70 440 300 600}");

p.poca_insert(imageblock, "type=array key=Rect value=" + container2);
p.poca_insert(imageblock, "type=name  key=Name value=logo");
p.poca_insert(imageblock, "type=name  key=Subtype value=Image");
p.poca_insert(imageblock, "type=name  key=fitmethod value=auto");

/* Block im Block-Dictionary der Seite aufnehmen */
p.poca_insert(blockdict, "type=dict key=logo direct=false value=" + imageblock);

/* ---------------------------------------------------------
 * Block vom Typ PDF erzeugen
 * ---------------------------------------------------------
 */
pdfblock = p.poca_new("containertype=dict usage=blocks " +
        "type=name key=Type value=Block");

container3 = p.poca_new("containertype=array usage=blocks " +
        "type=integer values={70 240 300 400}");

p.poca_insert(pdfblock, "type=array key=Rect value=" + container3);
p.poca_insert(pdfblock, "type=name key=Name value=pdflogo");
p.poca_insert(pdfblock, "type=name key=Subtype value=PDF");
p.poca_insert(pdfblock, "type=name key=fitmethod value=meet");

/* Block im Block-Dictionary der Seite aufnehmen */
p.poca_insert(blockdict, "type=dict key=pdflogo direct=false " + "value=" + pdfblock);


/* ---------------------------------------------------------
 * Block-Dictionary auf der aktuellen Seite aufnehmen
 * ---------------------------------------------------------
 */
p.end_page_ext("blocks=" + blockdict);


/* Bereinigen */
p.poca_delete(blockdict, "recursive");
```

12.8.2 Importieren von PDFlib-Blöcken

Sie können PDFlib-Blöcke aus dem Input-Dokument auf die aktuelle Ausgabeseite kopieren. Verwenden Sie dazu *PDF_process_pdi( )* und *action=copyallblocks* oder *action=copyblock* auf die folgende Weise:

```
if (p.process_pdi(p, doc, 0, "action=copyallblocks block={pagenumber=1}") != 1)
{
        /* Fehler */
}
```

So können Sie mehrstufige Block-Füllvorgänge implementieren. Beachten Sie, dass Blocknamen auf jeder Seite eindeutig sein müssen, d.h. Sie können nicht mehrere Blöcke mit dem gleichen Namen auf derselben Seite importieren. Verwenden Sie die Unteroption *outputblockname*, um Blöcke beim Kopieren umzubenennen.

12.9 Spezifikation für PDFlib-Blöcke

Die Blocksyntax geht konform mit der PDF-Referenz, die einen Erweiterungsmechanismus definiert, mit dem eine Applikation private Daten als Anhang an die Datenstrukturen einer PDF-Seite speichern kann. In diesem Abschnitt finden Sie eine Beschreibung des PDFlib Block-Syntax. Benutzer, die Blöcke mit dem Block-Plugin oder mit PDFlib erstellen, benötigen diese Informationen nicht.

PDF-Objektstruktur für PDFlib-Blöcke. Das Page-Dictionary enthält einen Eintrag *PieceInfo*, der als Wert ein weiteres Dictionary enthält. Das Page-Dictionary sollte auch den Schlüssel *LastModified* mit einem Zeitstempel für die Erstellung oder letzte Änderung an den Blockstrukturen enthalten. Dieses Dictionary enthält den Schlüssel *PDFlib*, der als Wert wiederum ein Dictionary mit Applikationsdaten enthält. Das Applikationsdaten-Dictionary enthält die beiden in Tabelle 12.14 aufgeführten Standardschlüssel.

Tabelle 12.14 Einträge in einem PDFlib-Applikationsdaten-Dictionary

Schlüssel	Wert
LastModified	*(Datum; erforderlich) Datum und Zeit der Erstellung oder letzten Änderung der Blöcke auf der Seite. Dieser Eintrag wird bei der Erstellung von Blöcken mit der POCA-Schnittstelle von PDFlib erzeugt.*
Private	*(Dictionary; erforderlich) Blockliste gemäß Tabelle 12.15*

Eine Blockliste ist ein Dictionary mit allgemeinen Information zur Blockverarbeitung sowie einer Liste aller Blöcke auf der Seite. Tabelle 12.15 enthält alle Schlüssel in einem Blocklisten-Dictionary.

Tabelle 12.15 Einträge in einem Blocklisten-Dictionary

Schlüssel	Wert
Blocks	*(Dictionary; erforderlich) Jeder Schlüssel ist ein Namensobjekt mit dem Namen eines Blocks; der zugehörige Wert ist das Block-Dictionary des Blocks (siehe Tabelle 12.17). Der Wert von* `Name` *im Block-Dictionary muss identisch mit dem Namen des Blocks in diesem Dictionary sein.*
BlockProducer[1]	*(String) Name der Software, mit der die Blöcke programmiert wurden. Dieser Eintrag wird bei der Erstellung von Blöcken mit der POCA-Schnittstelle von PDFlib erzeugt.*
PluginVersion[1]	*(String) Versionsnummer des Block-Plugins, mit dem die Blöcke erstellt wurden.*
pdfmark[1]	*(Boolean) Muss den Wert* `true` *haben, falls die Blockliste mit Hilfe von pdfmarks erstellt wurde.*
Version	*(Zahl; erforderlich) Die Versionsnummer der Blockspezifikation, gemäß der die Datei erstellt wurde. Dieses Dokument beschreibt Version 10 der Blockspezifikation.*

1. Genau einer der Schlüssel `BlockProducer`, `PluginVersion` *und* `pdfmark` *muss vorhanden sein.*

Datentypen von Blockeigenschaften. Für Blockeigenschaften werden mit Ausnahme von Handles und speziellen Listen wie Aktionslisten dieselben Datentypen wie für Optionslisten unterstützt. Tabelle 12.16 zeigt, wie diese Typen auf PDF-Datentypen abgebildet werden.

Tabelle 12.16 Datentypen für Blockeigenschaften

Datentyp	PDF-Typ und Anmerkungen
Boolean	*(Boolean)*
String	*(String)*
Schlüsselwort (Name)	*(Name) Es führt zu einem Fehler, wenn Schlüsselwörter übergeben werden, die nicht in der Liste der Schlüsselwörter enthalten sind, die von einer bestimmten Eigenschaft unterstützt werden.*
Float, Integer	*(Zahl) Optionslisten akzeptieren zwar Punkt und Komma als Dezimalzeichen, bei PDF-Zahlen ist aber nur ein Punkt erlaubt.*
Prozentwert	*(Array mit zwei Elementen) Das erste Array-Element enthält die Zahl, das zweite Element einen String mit einem Prozentzeichen.*
Liste	*(Array)*
Farbe	*(Array mit zwei oder drei Elementen) Das erste Array-Element legt einen Farbraum fest und das zweite einen Farbwert. Um die Abwesenheit von Farben festzulegen, muss die entsprechende Eigenschaft weggelassen werden. Für das erste Array-Element werden folgende Einträge unterstützt:* ***/DeviceGray*** *Das zweite Element ist ein einzelner Graustufenwert.* ***/DeviceRGB*** *Das zweite Element ist ein Array mit drei RGB-Werten.* ***/DeviceCMYK*** *Das zweite Element ist ein Array mit vier CMYK-Werten.* ***[/Separation/spotname]*** *Das erste Element ist ein Array mit dem Schlüsselwort* `/Separation` *und einem Schmuckfarbnamen. Das zweite Element ist ein Wert für den Farbauftrag.* *Das optionale dritte Element im Array legt eine Alternativfarbe für die Schmuckfarbe fest, die selbst ein Farbarray in einem der Farbräume* `DeviceGray`, `DeviceRGB`, `DeviceCMYK` *oder* `Lab` *ist. Fehlt die Alternativfarbe, muss der Schmuckfarbname eine Farbe sein, die PPS intern bekannt ist oder von der Anwendung zur Laufzeit definiert wurde.* ***[/Lab]*** *Das erste Element ist ein Array mit dem Schlüsselwort* `Lab`*. Das zweite Element ist ein Array aus drei Lab-Werten.*
unichar	*(Text-String) Unicode-Strings im Format* `utf16be`*, der mit BOM U+FEFF beginnen.*

Schlüssel in Block-Dictionaries. Block-Dictionaries können die Schlüssel in Tabelle 12.17 enthalten.

Tabelle 12.17 Einträge in einem Block-Dictionary

Eigenschaftsgruppe	Wert
Administrative Eigenschaften	*(Manche Schlüssel sind erforderlich) Administrative Blockeigenschaften gemäß Tabelle 12.4.*
Rechtecke	*(Manche Schlüssel sind erforderlich) Blockeigenschaften für Rechtecke gemäß Tabelle 12.5.*
Darstellung	*(Manche Schlüssel sind erforderlich) Darstellungsspezifische Eigenschaften für alle Blöcke gemäß Tabelle 12.6 und für Textline- und Textflow-Blöcke gemäß Tabelle 12.7.*
Textvorbereitung	*(Optional) Textvorbereitende Eigenschaften für Textflow- und Textline-Blöcke gemäß Tabelle 12.8.*
Textformatierung	*(Optional) Textformat-spezifische Eigenschaften für Textflow- und Textline-Blöcke gemäß Tabelle 12.9*
Objekteinpassung	*(Optional) Einpassungseigenschaften für Blöcke vom Typ Textline, Raster, PDF und Graphics gemäß Tabelle 12.10 und Einpassungseigenschaften für Textflow-Blöcke gemäß Tabelle 12.11.*
Vorgabewerte	*(Optional) Standard-Vorgabewerte für Blockeigenschaften gemäß Tabelle 12.12.*

Tabelle 12.17 Einträge in einem Block-Dictionary

Eigenschaftsgruppe	*Wert*
Benutzerdefiniert	*(Dictionary; optional) Dictionary mit Schlüssel/Wert-Paaren für benutzerdefinierte Eigenschaften gemäß Tabelle 12.13*

Beispiel. Das folgende Fragment zeigt den PDF-Code für zwei Blöcke, einen Textblock namens *job_title* und einen Image-Block namens *logo*. Der Textblock enthält eine selbst definierte Eigenschaft namens *format:*

```
<<
        /Contents 12 0 R
        /Type /Page
        /Parent 1 0 R
        /MediaBox [ 0 0 595 842 ]
        /LastModified (D:20120813200730)
        /PieceInfo << /PDFlib 13 0 R >>
>>

13 0 obj
<<
        /Private <<
            /Blocks <<
                /job_titel 14 0 R
                /logo 15 0 R
            >>
            /Version 10
            /PluginVersion (5.0)
        >>
        /LastModified (D:20120813200730)
>>
endobj

14 0 obj
<<
        /Type /Block
        /Rect [ 70 740 200 800 ]
        /Name /job_title
        /Subtype /Text
        /fitmethod /auto
        /fontname (Helvetica)
        /fontsize 12
        /Custom << /format 5 >>
>>
endobj

15 0 obj
<<
        /Type /Block
        /Rect [ 250 700 400 800 ]
        /Name /logo
        /Subtype /Image
        /fitmethod /auto
>>
```

A Änderungen an diesem Handbuch

Datum	*Änderungen*
21. Juni 2013	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 9.0.1*
13. Februar 2008	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 7.0.3*
9. März 2007	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 7.0.1*
1. Februar 2007	▸ *Deutsche Übersetzung und Umstrukturierung des Handbuchs für PDFlib 7.0.0*
13. März 2006	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 6.0.3 mit kleineren Erweiterungen und Korrekturen und einem neuen Abschnitt über Ruby*
14. September 2005	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 6.0.2 mit kleineren Erweiterungen und Korrekturen*
19. November 2004	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 6.0.1 mit kleineren Erweiterungen und Korrekturen* ▸ *Sprachabhängige Funktionsprototypen in Kapitel 8* ▸ *Beispiele für Hypertext-Erstellung in Kapitel 3*
19. August 2004	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 6.0.0*
18. Juni 2004	▸ *Erweiterungen für PDFlib 6*
6. Oktober 2003	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 5.0.2*
15. September 2003	▸ *Kleinere Änderungen für PDFlib 5.0.2, Ergänzung der Blockspezifikation*
30. Mai 2003	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 5.0.1*
6. August 2002	▸ *Deutsche Übersetzung des Handbuchs für PDFlib 4.0.3, Ergänzungen für .NET*
14. Juni 2002	▸ *Kleinere Änderungen für PDFlib 4.0.3 und Erweiterungen für die .NET-Bindung*
26. Januar 2002	▸ *Kleinere Änderungen für PDFlib 4.0.2 und Erweiterungen für die IBM-eServer-Edition*
17. Mai 2001	▸ *Kleinere Änderungen für PDFlib 4.0.1*
1. April 2001	▸ *Dokumentiert PDI- und andere Funktionen von PDFlib 4.0.0*
5. Februar 2001	▸ *Dokumentiert Template- und CMYK-Funktionen in PDFlib 3.5.0*
22. Dezember 2000	▸ *ColdFusion-Dokumentation und Erweiterungen für PDFlib 3.03; separate ActiveX-Edition des Handbuchs*
8. August 2000	▸ *Delphi-Dokumentation und kleinere Erweiterungen für PDFlib 3.02*
1. Juli 2000	▸ *Erweiterungen und Klarstellungen für PDFlib 3.01*
20. Februar 2000	▸ *Änderungen für PDFlib 3.0*
2. August 1999	▸ *Kleinere Änderungen und Erweiterungen für PDFlib 2.01*
29. Juni 1999	▸ *Eigene Abschnitte für die jeweiligen Sprachbindungen* ▸ *Erweiterungen für PDFlib 2.0*
1. Februar 1999	▸ *Kleinere Änderungen für PDFlib 1.0 (keine öffentliche Freigabe)*
10. August 1998	▸ *Erweiterungen für PDFlib 0.7 (nur für einen Kunden)*

Datum	*Änderungen*
8. Juli 1998	► *Erste Beschreibung der PDFlib-Skriptunterstützung in PDFlib 0.6*
25. Februar 1998	► *Kleine Erweiterungen am Handbuch für PDFlib 0.5*
22. September 1997	► *Erste öffentliche Version von PDFlib 0.4 und dieses Handbuchs*

Index

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

PDFlib GmbH

PDFlib GmbH
Franziska-Bilek-Weg 9
D-80339 München
www.pdflib.com
Tel. +49 • 89 • 452 33 84-0
Fax +49 • 89 • 452 33 84-99

Bei Fragen können Sie die PDFlib-Mailing-Liste abonnieren
und sich deren Archive ansehen unter tech.groups.yahoo.com/group/pdflib

Vertriebsinformationen
sales@pdflib.com

Support
support@pdflib.com *(geben Sie bitte immer Ihre Lizenznummer an)*