Mustang Project Entwicklerdokumentation

Jochen Stärk

Entwurf, 25.06.20014

http://www.mustangproject.org

Übersicht

	Plattform	Lizenz	Funktionsumfang			Geeignet für			Preis
			Lesen	XML Schreiben	PDF Schreiben	Kommerz. Sotware	Freeware	Open Source	
intarsys	Java	proprietär	✔	✔	✔	✔	✘	✘	a.A.
Konik	Java	AGPL	✔	✔	✔	✘	✘	✔	0 €
Mustang	Java	APL	✔	✔	✔	✔	✔	✔	0 €
https://github.com/stephanstapel/ZUGFeRD-csharp	C#	APL	✔	✔	✘	✔	✔	✔	0 €
https://github.com/opendatalab-de/zugferd	Java	APL	✔	✘	✘	✔	✔	✔	0 €

Mustang

Mit installiertem OpenOffice.org oder LibreOffice und Eclipse for Java.

1. Starten Sie Eclipse, Neues Java-Eclipse-Projekt erstellen, beispielsweise „sample“. Wechseln Sie in der Shell in den Ordner.
2. Download von
    1. Apache PDFBox
        1. Downloaden Sie http://apache.openmirror.de/pdfbox/1.8.5/pdfbox-1.8.5.jar
        2. Downloaden Sie http://apache.openmirror.de/pdfbox/1.8.5/preflight-app-1.8.5.jar
        3. Downloaden Sie http://apache.openmirror.de/pdfbox/1.8.5/xmpbox-1.8.5.jar
    2. Mustang
        1. Downloaden Sie https://github.com/Rayman2200/PDFA3/raw/master/mustang/target/mustang-1.0.jar
        2. Downloaden Sie https://raw.githubusercontent.com/Rayman2200/PDFA3/master/mustang/src/main/java/org/mustangproject/ZUGFeRD/NOTICE
    3. Laden Sie
        1. http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20140522_501.pdf
    4. Öffnen Sie OpenOffice.org. Laden Sie
        1. Die Quelldatei der Rechnung wget http://www.mustangproject.org/MustangGnuaccountingBeispielRE-20140522_501.odt
        2. Öffnen Sie die Datei in OpenOffice.org
        3. Datei|Exportieren als PDF: Wichtig ist hier, dass Sie die Checkbox PDF/A-1a setzen
        4. Speichern Sie die PDF-Datei beispielsweise als „blanko.pdf“ im sample-Ordner.
3. Wechseln Sie zurück zu Eclipse. Fügen Sie durch Rechtsklick auf das Projekt alle vier heruntergeladenen JAR-Dateien Projekteigenschaften als „externe Jars“ zum „Build Path“ hinzu.

Lesen

4. Neue Klasse unterhalb von src, beispielsweise Reader. Inkl. „Public static void main()“ generieren lassen.
5. Geben Sie innerhalb von Main „ZUGFeRDImporter zi=new ZUGFeRDImporter();“ ein und lassen Sie den Import durch STRG+SHIFT+O ergänzen
6. verwenden Sie zi.extract(PDF-Dateiname) und ggf. canParse() um festzustellen ob es sich um ZUGFeRD-Daten handelt.
7. Nach zi.parse() haben Sie Zugriff auf die getter wie getAmount()
8. Welche Daten enthalten sind, können Sie der XML-Datei entnehmen die im ZUGFeRD-Beispiel-PDF eingebettet ist

Komplettes Lesebeispiel

```
package sample;
```

```
import org.mustangproject.ZUGFeRD.ZUGFeRDImporter;

public class Read {

	public static void main(String[] args) {
		ZUGFeRDImporter zi=new ZUGFeRDImporter();
		zi.extract("./MustangGnuaccountingBeispielRE-20140522_501.pdf");
		System.out.println("Lese ZUGFeRD");
		if (zi.canParse()) {
			zi.parse();
			System.out.println("Fälliger Betrag:"+zi.getAmount());
			System.out.println("BIC:"+zi.getBIC());
			System.out.println("IBAN:"+zi.getIBAN());
			System.out.println("Kontoinhaber:"+zi.getHolder());

		}


	}

}
```

Schreiben

Ein Beispielprogramm zum Schreiben ist deshalb umfangreicher, weil erstens mehr Daten in einer differenzierteren Struktur geschrieben werden als derzeit beim Lesen benötigt und zweitens der Exporter sich seine Daten direkt per Interface, solzusagen in einer Art „pull-Ansatz“ holt. So wird zwar eine redundante Datenhaltung vermieden, ein Beispielprogramm muss jedoch Sorge tragen die Daten in geeigneter Weise zumindest im Arbeitsspeicher vorzuhalten. Bei der Einbindung in eine produktive Warenwirtschaft entfällt dieser Schritt, da die Warenwirtschaft die Daten ja bereits zu beliebigen Zeitpunkten zur Verfügung stellt.

Das derzeit alternative ZUGFeRD-Open-Source-Projekt Konik verfolgt einen konventionellen „push-Ansatz“ in dem – für Beispielprogramme einfacher – Daten redundant durch setter-Methoden gesetzt werden.

1. Neue Klasse unterhalb von src, beispielsweise MainClass. „Public static void main()“ generieren lassen.
2. Änderin Sie `public class MainClass  in public class MainClass implements IZUGFeRDExportableTransaction`
3. `class Contact implements IZUGFeRDExportableContact {}`
4. `class Item implements IZUGFeRDExportableItem {}`
5. `class Product implements IZUGFeRDExportableProduct {}`
6. Generieren Sie die Imports durch Drücken von STRG+SHIFT+O
7. Klicken Sie links auf MainClass und drücken Sie ALT+SHIFT+S, wählen Sie Override/Implement Methods und drücken return.
8. Klicken Sie auf Contact und Wiederholen Sie den letzten Schritt.

9. Klicken Sie auf Item und Wiederholen Sie den letzten Schritt.
10. Klicken Sie auf Products und Wiederholen Sie den letzten Schritt.
11. Folgende Methoden von Contact sollten Folgendes zurückgeben:
    1. getCountry(): "DE"
    2. getLocation(): "Spielkreis"
    3. getName(): "Theodor Est"
    4. getStreet(): "Bahnstr. 42"
    5. getVATID(): ""
    6. getZIP(): "88802";
12. Folgende Methoden der Hauptklasse sollten folgendes zurückgeben:
    1. getDueDate(): new GregorianCalendar(2014,Calendar.*JUNE*,12).getTime()
    2. getIssueDate(): new GregorianCalendar(2014,Calendar.*MAY*,22).getTime()
    3. getNumber(): "RE-20140522/501"
    4. getOwnBIC(): "COBADEFXXX"
    5. getOwnBankName(): ""
    6. getOwnIBAN(): "DE88 2008 0000 0970 3757 00"
    7. getOwnOrganisationName(): "Bei Spiel GmbH"
    8. getOwnTaxID(): "22/815/0815/4"
    9. getOwnVATID(): "DE136695976"
    10. getRecipient(): new Contact()
    11. getTotal(): new BigDecimal("496.00")
    12. getTotalGross(): new BigDecimal("571.04")
    13. getDeliveryDate() new Date();
    14. getOwnCountry() "DE"
    15. getOwnLocation() "Test city"
    16. getOwnStreet() "Test Street 22"
    17. getOwnZIP() "12345
    18. Sowohl die Item- wie auch die Product-Klasse geben in den überschriebenen Methoden Member-Variablen zurück, die durch den Konstruktor gesetzt werden.
    19. getZFItems() der Hauptklasse kann jetzt Produkte anlegen und diese als Array von Posten (Items) zurückliefern:

```
            Item[] allItems=new Item[3];
        Product designProduct=new Product("", "Künstlerische Gestaltung
(Stunde)", "HUR", new BigDecimal("7.000000"));
        Product balloonProduct=new Product("", "Luftballon", "C62", new
BigDecimal("19.000000"));
        Product airProduct=new Product("", "Heiße Luft pro Liter", "LTR", new
BigDecimal("19.000000"));

        allItems[0]=new Item(new BigDecimal("160"), new BigDecimal("171.20"),
new BigDecimal("1"), new BigDecimal("171.20"), designProduct);
        allItems[1]=new Item(new BigDecimal("0.79"), new BigDecimal("0.94"),
new BigDecimal("400"), new BigDecimal("376.04"), balloonProduct);
        allItems[2]=new Item(new BigDecimal("0.10"), new BigDecimal("0.12"),
new BigDecimal("200"), new BigDecimal("23.80"), airProduct);
        return allItems;
```

20. In der Main-Methode instantiiert man jetzt die Klasse und ruft eine neue, beispielsweise apply() genannte Methode auf.
21. In der apply-Methode kann man jetzt ein PDDocument
    1. laden,
    2. einen ZUGFeRDExporter instantiieren,
    3. dessen PDFmakeA3compliant (mit „Producer“, also Anwendungs- und „Creator“ also Autorenname) und
    4. PDFattachZugferdFile-Methoden (mit this als IZUGFeRDExportableTransaction) aufrufen und
    5. das PDDocument wieder speichern. Die apply-Methode sieht dann – mit entsprechenden try/catch-Blöcken- beispielsweise so aus:

```
PDDocument doc;
    try {
          doc = PDDocument.load("blanko.pdf");
          // automatically add Zugferd to all outgoing invoices
          ZUGFeRDExporter ze = new ZUGFeRDExporter();
          ze.PDFmakeA3compliant(doc, "My Application",
                      System.getProperty("user.name"), true);
          ze.PDFattachZugferdFile(doc, this);

          doc.save("unblanko.pdf");
    } catch (IOException e) {
          e.printStackTrace();
    } catch (TransformerException e) {
          e.printStackTrace();
    } catch (COSVisitorException e) {
          e.printStackTrace();
    }
```

Komplettes Schreibbeispiel

```
package sample;

import java.io.IOException;
import java.math.BigDecimal;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.HashMap;

import javax.xml.transform.TransformerException;

import org.apache.pdfbox.exceptions.COSVisitorException;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.mustangproject.ZUGFeRD.IZUGFeRDExportableContact;
import org.mustangproject.ZUGFeRD.IZUGFeRDExportableItem;
```

```
import org.mustangproject.ZUGFeRD.IZUGFeRDExportableProduct;
import org.mustangproject.ZUGFeRD.IZUGFeRDExportableTransaction;
import org.mustangproject.ZUGFeRD.ZUGFeRDExporter;

class Contact implements IZUGFeRDExportableContact {

	@Override
	public String getCountry() {
		return "DE";
	}

	@Override
	public String getLocation() {
		return "Spielkreis";
	}

	@Override
	public String getName() {
		return "Theodor Est";
	}

	@Override
	public String getStreet() {
		return "Bahnstr. 42";
	}

	@Override
	public String getVATID() {
		return "";
	}

	@Override
	public String getZIP() {
		return "88802";
	}
}
class Product implements IZUGFeRDExportableProduct {

	private String description, name, unit;
	private BigDecimal VatPercent;


	public Product (String description, String name, String unit, BigDecimal
VatPercent) {
		this.description=description;
		this.name=name;
		this.unit=unit;
		this.VatPercent=VatPercent;
	}
	@Override
	public String getDescription() {
		return description;
	}

	@Override
```

```
	public String getName() {
		return name;
	}

	@Override
	public String getUnit() {
		return unit;
	}

	@Override
	public BigDecimal getVATPercent() {
		return VatPercent;
	}
}
class Item implements IZUGFeRDExportableItem {

	private BigDecimal price, priceGross,quantity,totalGross;
	private Product product;

	public Item(BigDecimal price, BigDecimal priceGross,BigDecimal
quantity,BigDecimal totalGross, Product product) {
		this.price=price;
		this.priceGross=priceGross;
		this.quantity=quantity;
		this.totalGross=totalGross;
		this.product=product;
	}

	@Override
	public BigDecimal getPrice() {
		return price;
	}

	@Override
	public BigDecimal getPriceGross() {
		return priceGross;
	}

	@Override
	public IZUGFeRDExportableProduct getProduct() {
		return product;
	}

	@Override
	public BigDecimal getQuantity() {
		return quantity;
	}

	@Override
	public BigDecimal getTotalGross() {
		return totalGross;
	}


}
```

```
public class MainClass implements IZUGFeRDExportableTransaction{


	@Override
	public Date getDueDate() {
		return new GregorianCalendar(2014,Calendar.JUNE,12).getTime();
	}

	@Override
	public Date getIssueDate() {
		return new GregorianCalendar(2014,Calendar.MAY,22).getTime();
	}

	@Override
	public String getNumber() {
		return "RE-20140522/501";
	}

	@Override
	public String getOwnBIC() {
		return "COBADEFXXX";
	}

	@Override
	public String getOwnBankName() {
		return "";
	}

	@Override
	public String getOwnIBAN() {
		return "DE88 2008 0000 0970 3757 00";
	}

	@Override
	public String getOwnOrganisationName() {
		return "Bei Spiel GmbH";
	}

	@Override
	public String getOwnTaxID() {
		return "22/815/0815/4";
	}

	@Override
	public String getOwnVATID() {
		return "DE136695976";
	}

	@Override
	public IZUGFeRDExportableContact getRecipient() {
		return new Contact();
	}

	@Override
	public BigDecimal getTotal() {
```

```
		return new BigDecimal("496.00");
	}

	@Override
	public BigDecimal getTotalGross() {
		return new BigDecimal("571.04");
	}
	@Override
	public Date getDeliveryDate() {

		return new Date();
	}

	@Override
	public String getOwnCountry() {
		return "DE";
	}

	@Override
	public String getOwnLocation() {
		return "Test city";
	}

	@Override
	public String getOwnStreet() {
		return "Test Street 22";
	}

	@Override
	public String getOwnZIP() {
		return "12345";
	}

	@Override
	public HashMap<BigDecimal, BigDecimal> getVATPercentAmountMap() {
		HashMap<BigDecimal, BigDecimal> VATs=new HashMap<BigDecimal,
BigDecimal>();
		return VATs;
	}

	@Override
	public IZUGFeRDExportableItem[] getZFItems() {
		Item[] allItems=new Item[3];
		Product designProduct=new Product("", "Künstlerische Gestaltung
(Stunde)", "HUR", new BigDecimal("7.000000"));
		Product balloonProduct=new Product("", "Luftballon", "C62", new
BigDecimal("19.000000"));
		Product airProduct=new Product("", "Heiße Luft pro Liter", "LTR", new
BigDecimal("19.000000"));

		allItems[0]=new Item(new BigDecimal("160"), new BigDecimal("171.20"),
new BigDecimal("1"), new BigDecimal("171.20"), designProduct);
		allItems[1]=new Item(new BigDecimal("0.79"), new BigDecimal("0.94"),
new BigDecimal("400"), new BigDecimal("376.04"), balloonProduct);
		allItems[2]=new Item(new BigDecimal("0.10"), new BigDecimal("0.12"),
```

```
new BigDecimal("200"), new BigDecimal("23.80"), airProduct);
            return allItems;
      }



      public void apply()  {
            PDDocument doc;
            try {
                  doc =
PDDocument.load("/home/jstaerk/workspace/sample/blanko.pdf");
                  // automatically add Zugferd to all outgoing invoices
                  ZUGFeRDExporter ze = new ZUGFeRDExporter();
                  ze.PDFmakeA3compliant(doc, "My Application",
                              System.getProperty("user.name"), true);
                  ze.PDFattachZugferdFile(doc, this);

                  doc.save("unblanko.pdf");
            } catch (IOException e) {
                  e.printStackTrace();
            } catch (TransformerException e) {
                  e.printStackTrace();
            } catch (COSVisitorException e) {
                  e.printStackTrace();
            }
            System.out.println("Hello ZUGFeRD");

      }

      public static void main(String[] args) {
            MainClass write=new MainClass();
            write.apply();

      }

}
```