https://doi.org/10.24867/GRID-2018-p54 Professional paper

PDF METADATA AND ITS CONVERSION TO XJDF

Thomas Hoffman-Walbeck
Stuttgart Media University, Faculty Print and Media, Stuttgart, Germany

Abstract: *In this paper, we show an example of how a product description such as the specifications of the print substrate can be embedded as metadata in a PDF file using a well-documented technology for variable data printing. We demonstrate the corresponding data structures in PDF. Moreover, we are explaining how these structures can be integrated in a PDF file and retrieved afterwards with a JAVA program. In addition, this metadata is converted into XJDF, which can be passed on to a commercially available workflow management system. The basic structure of XJDF is explained as well as its generation with JAVA.*

Key words: Metadata, PDF, PDF/VT, XJDF, PrintTalk, JDF, CSV, Web-to-Print, Workflow Management System

1. INTRODUCTION

Typically, a production department of a print service provider (PSP) imports content data - mostly PDF - and metadata - in formats like JDF or private XML - into a Workflow Management System (WMS). PDF contains the layout definition of the pages that are supposed to be printed, while the metadata incorporates the overall product specification as well as production issues. Metadata inside PDF are rarely used for specifying the product or even controlling the production. Extended Metadata Platform (XMP) (ISO 16684-1:2012, 2012), for example, is mainly used for storing information about page components (like images) or about the PDF document itself (like contact details of the author). The XMP metadata structure is not suitable for defining a product and its product parts.
The PSP usually receives PDF documents from the print buyer (PB) or some agency via an email attachment, FTP-server, portal or some cloud storage. Metadata concerning the product definition, however, is generated by the MIS of the PSB or by some PB's Enterprise Resource Planning System (ERP). The MIS-employee has to learn the PB's product intentions and fill them in the system. Only in the case of Web-to-Print (W2P) the PDF and the metadata stem from the same source, i.e. the W2P-Server. In all other cases, the content data and the product specification are using different communication channels, which can be time-consuming to synchronize. Anyhow, even with W2P two different files need to be handled simultaneously. Thus, it would be easier if the product definition could be stored inside the PB's PDF, in particular since it describes the view of the PBs intention concerning the product in the first place. This metadata could be generated by a W2P or even by the customer himself, using a plug-in for a layout software for example.
The necessary technology became available with PDF/VT (ISO 16612-2:2010, 2010), an extension of PDF 1.6 for transactional and variable data printing. Now, however, this is part of PDF 2.0 (ISO 32000-2:2017, 2017). The technology allows to define different product parts and to assign individual pages of a PDF document to just these product parts. In addition, the final product as well as each product part may refer to their own metadata structure, in which one can specify detailed technical information. As an example, the desired printing substrates of the product parts or the overall binding intent may be given. While PDF 2.0 provides the general file structure, the CIP4 organization specified the necessary semantics in the papers "ICS-Common Metadata for Document Production Workflows" (Prosi, 2015) and "ICS-IntentMetadata.PDF.1.5 ISO Draft" (Meissner, 2015). In this context, please refer also to the draft ISO standard ISO/CD 21812-1 about print product metadata in PDF files (ISO/DIS 21812-1).
At first, we will outline in this paper the technical structure of the PDF metadata with the help of an example (a booklet with a cover and content). Moreover, we will explain how to implant such structure into a PDF file by a JAVA program. The second part of this paper shows how to extract this information from the PDF file and how to convert the data into a suitable input format for a WMS like JDF (Meissner, 2018), XJDF (Meissner, 2018) and CSV (Wikipedia, 2018). However, in this paper we will only describe in detail the conversion to XJDF through a JAVA program. Since the XJDF can be employed to generate a new job within the WMS it will be embedding into a PrintTalk (Confluence.cip, 2018;

Confluence.cip, 2015) business object of type “purchase order”. Finally, we will show what kind of results can be gained when importing the generated PrintTalk/XJDF file into a commercially available WMS.
Figure 1 shows the modules of the implementation. Software ① and ② are stand-alone prototype applications. The user enters details in a dialogue mask of program ① in order to provide the necessary metadata. Note that ① will be executed by a PB, while ② will precede the PSPs WMS.

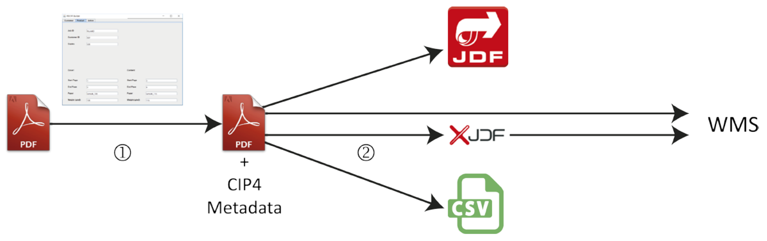


Figure 1: Structure of the software applications: ① writes CIP4 metadata into a PDF,
② reads the metadata and converts it to JDF, XJDF and CSV.

2. METHODS

For both ① and ② we are using the SDK IntelliJIDEA 2018.1 (Jetbrains, 2018) and the JAVA version 9.0.1. For reading and writing the PDF/VT-structures, we employ the libraries PDFlib und PDI 9.1.2 (Pdflib, 2018) of the company PDFlib GmbH. Furthermore we include different external libraries for the XJDF conversion in ②: istack-commons-runtime-2.16.jar, jaxb-core-2.2.7.jar, jaxb-impl-2.2.7.jar, FastInfoset-1.2.12.jar, commons-configuration-1.9.jar, commons-io-2.4.jar, commons-lang-2.6.jar, commons-logging-1.1.1.jar, activation-1.1.1.jar, jaxb-api-2.2.7.jar, jsr173_api-1.0.jar, junit-4.10.jar, commons-lang3-3.4.jar, xJdfLib-0.13.jar, xPrintTalkLib-0.13.jar, hamcrest-core-1.1.jar, annotations-13.0.jar, jaxb2-basics-runtime-0.9.1.jar. The generated PDF should not be compressed, in order to analyse the file with a standard text editor. Moreover the tool “Enfocus Browser Vers. 3.0” is very helpful to study the dictionary hierarchy in PDF.

3. RESULTS

In subsection 3.1, we will explain the document part (DPart) hierarchy inside PDF, which might reference to one or more pages (page objects). Each DPart can also reference to an object called Document Part Metadata (DPM), which in turn can contain technical details about the document part. We will present a particular example. Then we will show in subsection 3.2 how to generate DPart und DPM object using the PDFlib library for JAVA. The following two section cover details concerning ②. In section 3.3, we will present the basic structure of XJDF. Section 3.4 primarily covers the conversion of the metadata information into XJDF using the library xJDF of CIP4. Finally, in subsection 3.5 we will demonstrate the XJDF import into a WMS.

3.1 Metadata structure

Figure 2 shows an example of a simplified PDF/VT structure. In the right-hand side, there are the normal page objects, in the left-hand side the DPart objects.

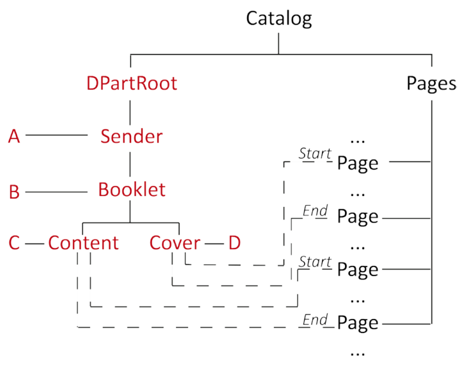


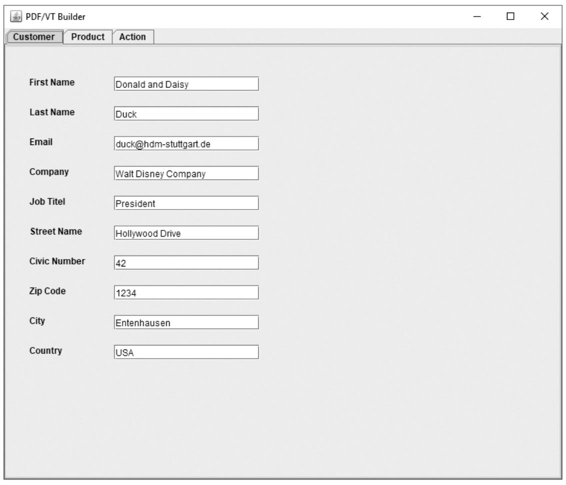
Figure 2: DPart hierarchy with DPMs A, B, C and D

The leaves of the Dpart-tree are the product parts. They contain the information about the first and the last page of the product part. The page number in the PDF document does not give the references, but rather as the page object number, which identifies the PDF page object.
The actual definitions of the four DPart objects inside PDF are to be seen in figure 3. Please note that all of them are contained on one stream object. Each DPart is a dictionary that begins with "<<" and ends with ">>". PDF dictionaries have two parts per entry. The first one is called *key* (e.g. /Type), the second one is called value (e.g. /Dpart).

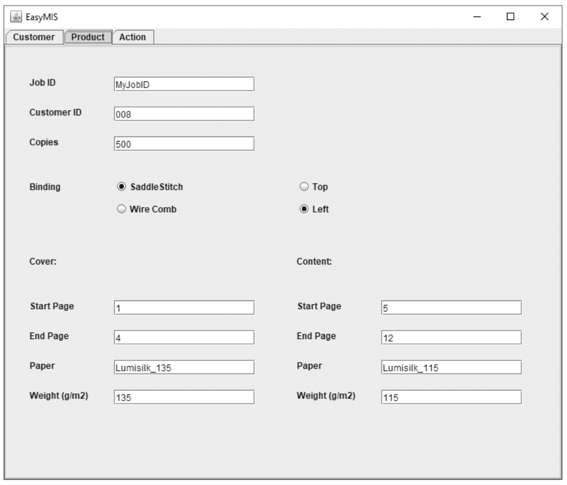
```
stream
11 0
34 63
9 126
6 181
<</Type/DPart/Parent 9 0 R/DPM 12 0 R/Start 22 0 R/End 29 0 R>>    ← Cover
<</Type/DPart/Parent 9 0 R/DPM 35 0 R/Start 39 0 R/End 74 0 R>>    ← Content
<</Type/DPart/Parent 6 0 R/DPM 10 0 R/DParts[ 78 0 R]>>            ← Sender
<</Type/DPart/Parent 80 0 R/DPM 7 0 R/DParts[ 81 0 R]>>            ← DpartRoot
endstream
```

Figure 3: DPart objects in side PDF

Each `DPart` in figure 2 and 3 refers to a specific `DMP`, which hold information like the intended print substrate or about the contact details of the PB. In application ① the user provides this data via a GUI, which is partially shown in figure 4. The entries in the text fields of the GUI are stored in a XML preferences file in order to make them available for the next launch of the program. The entries can, of course, alternatively edited directly in the XML file.



First Name	Donald and Daisy
Last Name	Duck
Email	duck@hdm-stuttgart.de
Company	Walt Disney Company
Job Titel	President
Street Name	Hollywood Drive
Civic Number	42
Zip Code	1234
City	Entenhausen
Country	USA



Job ID: MyJobID
Customer ID: 008
Copies: 500
Binding: [✓ SaddleStitch] Wire Comb — Top [✓ Left]

Cover:		Content:	
Start Page	1	Start Page	5
End Page	4	End Page	12
Paper	Lumisilk_135	Paper	Lumisilk_115
Weight (g/m2)	135	Weight (g/m2)	115

Figure 4: GUI of application ①

The structures of three (slightly text formatted) DPMs can be seen Figure 5. The first one contains information about the sender, the second one some characteristics about the final product, including the binding intent. The product properties concerning the cover of the brochure are given in the third DPM. In this example, only the dictionaries `CIP4_BindingIntent and CIP4_MediaIntent` have been embedded. In similar manner, one can add other resources like `CIP4_ColorIntent`, `CIP4_` or `CIP4_FoldingIntent`. The metadata for the content has been omitted in figure 5.

```
<</DPM<</CIP4_Root<</CIP4_Metadata<<
                           /CIP4_Conformance(base)
                           /CIP4_Creator(Disney)
                           /CIP4_JobID(MyJobID)
                           /CIP4_Sender<<
                                 /CIP4_Contact<<
                                        /CIP4_Person<<
                                               /CIP4_FirstName(Daisy and Donald)
                                               /CIP4_Lastname(Duck)
                                               /CIP4_JobTitel(Presedent)>>
                                        /CIP4_Address<<
                                               /CIP4_City(Entenhausen)
                                               /CIP4_Country(USA)
                                               /CIP4_PostalCode(1234)
                                               /CIP4_StreetName(Hollywood Drive)
                                               /CIP4_CivicNumber(42)>>
                                        /CIP4_ComChannel<<
                                               /CIP4_Email(duck@hdm-stuttgart.de)
                                               /CIP4_ChannelType(Email)>>
                                        /CIP4_Company<<
                                               /CIP4_OrganizationName(Walt Disney Company)>>
>>     >>     >>     >>     >>     >>

<</DPM<</CIP4_Root<<
                    /CIP4_Production<<
                           /CIP4_DescriptiveName(Cover)>>
                    /CIP4_Intent<<
                           /CIP4_ProductType(WrapAroundCover)
                           /CIP4_MediaIntent<<
                                 /CIP4_MediaQuality(lumisilk_135)
                                 /CIP4_MediaWeight(135)>>
>>     >>     >>     >>
```

Figure 5: DPM structures in PDF example

3.2 Writing metadata into PDF

Figure 6 shows a snippet of the code for writing the metadata into the PDF file. In the first two lines, the PDFLib is open and the PDF file is read. In the next two lines DPM dictionaries are generated, that are called `CIP4_Intent` and `CIP4_Media_Intent`. Here `POCA` stands for "PDF Object Creation API" and is a PDFlib set of methods for creating objects like DPMs.
In the following line the dictionary `CIP4_Media_Intent` is filled in with the key-value pair `CIP4_Media_Intent/coverPaperWeight`, whereas `CIP4_MediaWeight` is a metadata term defined in [1] and `coverPaperWeight` an internal variable that holds the corresponding string that the user has filled in via the GUI. The method "P.poca_insert" requires a String in the second parameter. The last line inserts the dictionary `CIP4_MediaIntent` as a key/value pair into the `CIP4_Intent` dictionary. Using this technology one can construct the tree of dictionaries in figure 5.

```
…
p=new pdflib();
…
docIn = p.open_pdi_document(inFile, “”);
…
p.begin_dpart(“dpm=” + dpm);
…
CIP4_Intent = p.poca_new(“containertype = dict usage=dpm”);
…
CIP4_MediaIntent = p.poca_new(“containertype = dict usage=dpm”);
…
p.poca_insert(CIP4_MediaIntent, “type=string key=CIP4_MediaWeight value=”+”{“+coverPaperWeight+”}”);
…
p.poca_insert(CIP4_Intent, “type=dict key=CIP4_MediaIntent value=” + CIP4_MediaIntent);
…
```

Figure 6: Writing a DPart and DPMs into PDF

3.3 XJDF

An introduction into XJDF technology can be found in (Meißner, 2017) and in (Meissner, 2018). Here we only want to describe the XJDF outcome of software ②. The XJDF element in figure 7 is embraced by a `PrintTalk` element.

```
<ptk:PrintTalk version=“2.0‟ payloadID=“5bf27d60-cc4f-4ed3-86a4-3a2a88b6900c‟ Timestamp=“2018-
06-07T12:52:16Z‟ xmlns:ptk=“http://www.printtalk.org/schema_2_0‟ xmlns:xjdf=“http://www.CIP4.org/
JDFSchema_2_0‟>
    <ptk:Request>
        <ptk:PurchaseOrder Currency=“Euro‟ BusinessID=“4711‟>
            <xjdf:XJDF Category=“Product‟ DescriptiveName=“Final Broschur‟ JobID=“MyJobID‟
                  Types=“Product‟>
                <xjdf:ProductList>
                    <xjdf:Product Amount=“500‟ DescriptiveName=“Broschur (root)‟ ID=“ID_broschur‟
                          IsRoot=“true‟
                        ProductType=“booklet‟>
                        <xjdf:Intent Name=“BindingIntent‟>
                            <xjdf:BindingIntent BindingOrder=“Collecting‟ BindingSide=“Left‟
                                  BindingType=“SaddleStitch‟/>
                        </xjdf:Intent>
                        <xjdf:Intent Name=“LayoutIntent‟>
                            <xjdf:LayoutIntent Pages=“12‟/>
                        </xjdf:Intent>
                    </xjdf:Product>
                    <xjdf:Product Amount=“500‟ DescriptiveName=“Cover‟ ID=“ID_Cover‟ IsRoot=“false‟
                        ProductType=“WrapAroundCover‟>
                        <xjdf:Intent Name=“MediaIntent‟>
                            <xjdf:MediaIntent MediaQuality=“Lumisilk_135‟ MediaType=“Paper‟
                               NamedWeight=“135‟/>
                        </xjdf:Intent>
                        <xjdf:Intent Name=“LayoutIntent‟>
                            <xjdf:LayoutIntent Pages=“4‟/>
                        </xjdf:Intent>
                    </xjdf:Product>

                  ...Analogue structure for the product part „Content“...

                </xjdf:ProductList>
                <xjdf:ResourceSet Name=“Contact‟ ProcessUsage=“Input‟>
                    <xjdf:Resource ID=“Contact_8fb6beef-0c72-43b5-a42a-cb3d53b4b262‟>
                        <xjdf:Contact ContactTypeDetails=“Customer‟>
                            <xjdf:Address City=“Entenhausen‟ CivicNumber=“42‟ Country=“USA‟
                                  PostalCode=“1234‟Street=“Hollywood
                              Drive‟/>
                            <xjdf:ComChannel ChannelType=“Email‟ Locator=“duck@hdm-stuttgart.de‟/>
                            <xjdf:Company OrganizationName=“Walt Disney Company‟/>
                            <xjdf:Person FamilyName=“Duck‟ FirstName=“Donald and Daisy‟
                                  JobTitle=“President‟/>
                        </xjdf:Contact>
                    </xjdf:Resource>
                </xjdf:ResourceSet>
                <xjdf:ResourceSet Name=“RunList‟ ProcessUsage=“Input‟>
                    <xjdf:Resource ID=“RunList_7a226b95-f194-4ae1-839f-b5888b8cc032‟>
                        <xjdf:Part ProductPart=“ID_RunList_Cover‟/>
                        <xjdf:RunList NPage=“4‟ Pages=“1 4‟>
                            <xjdf:FileSpec URL=“https://www.hdm-stuttgart.de/print40/Flyer.pdf‟/>
                        </xjdf:RunList>
                    </xjdf:Resource>
                    <xjdf:Resource ID=“RunList_6b34765a-fd03-49e4-b5db-798801b3b97a‟>
                        <xjdf:Part ProductPart=“ID_RunList_Content‟/>
                        <xjdf:RunList NPage=“8‟ Pages=“5 12‟>
                            <xjdf:FileSpec URL=“https://www.hdm-stuttgart.de/print40/Flyer.pdf‟/>
                        </xjdf:RunList>
                    </xjdf:Resource>
                </xjdf:ResourceSet>
            </xjdf:XJDF>
        </ptk:PurchaseOrder>
    </ptk:Request>
</ptk:PrintTalk>
```

Figure 7: PrintTalk element with XJDF sub-element

Sub-element of the PrintTalk is the business object Request, which in turn contains the PurchaseOrder element (see (Confluence.cip4, 2018)). XJDF is a child element of PurchaseOrder. Next, we find the ProductList in the hierarchy. In our example, there are three products in this list. First, there is the final product with the value of the attribute IsRoot equals true. Next, the product parts "cover" and "content" are also defined as product elements with attributes IsRoot equal false. Each those products contain one (or more) Intent element(s), which describe the product or the product parts from the PB's point of voiew. Each Intent resource is specialized further by a Name attribute and the according sub-element. Here, the final product contains a BindingIntent, the product parts MediaIntents, which define the printing substrates. In each of them specific LayoutIntents have been incorporated. Note that the attributes of these elements are not derived from a corresponding DPM dictionary in the PDF file but rather from the DPart structures (see figure 3).
ResourceSets succeed the ProductList. Each ResourseSet contain one or more Resources, which in turn can describe physical (like paper used by PSB, not defined in our example) or logical entities like RunLists. The two RunLists in Figure 7 define the PDF pages of the product parts cover and content. The attributes NPage denotes the "Number of Pages" and Pages the range of pages in the document. Note that here the sequential page numbers in the document are referred to, while in figure 3, the PDF internal page object numbers are stored.

3.4 Reading metadata from PDF and converting it into XJDF

Figure 8 and 9 show some code snippets for the extracting metadata out of a PDF/VT file. Reading the metadata is actually mostly straightforward: In the first line, the PDFlib is opened, in the next the PDF document is parsed in. After that, a path of a dictionary is specified from which entries would be determined. This is done in the last two lines. The individual entries in the dictionary are accessed in the notation of an array.

```
…
p = new pdflib();
…
docIn = p.open_pdi_document(inFileFullName, "");
…
path = "/Root/DPartRoot/DPartRootNode/DPM/DPM/CIP4_Root/CIP4_Metadata/
        CIP4_Sender/CIP4_Person";
firstName = p.pcos_get_string(docIn, path + "[" + 0 + "].val");
lastName = p.pcos_get_string(docIn, path + "[" + 1 + "].val");
…
```

Figure 8: Reading values from a PDF/VT dictionary

Figure 9 shows a snippet of the method concerning the computing of page numbers using PDF page object numbers. Here it is assumed that the page objects in the PDF file have the same order as the pages in the PDFD file. The page object numbers can be arbitrarily. This assumption, however, is not true in general (see F.4.2 in (ISO 32000-2:2017, 2017), p. 885)

```
double getPageNumber(String PDFPageObject) {
…
pageObjectNumber = p.pcos_get_number(docIn, "pcosid:" + PDFPageObject);
…
int dict_length = p.pcos_get_number(docIn, "length:" + "pages");
…
for (i = 0; i < (int) dict_length; i++) {
      if (pageObjectNumber == p.pcos_get_number(docIn,"pcosid:"+"pages"+"["+i+"]"))
            pageNumber = i;
…
      return (pageNumber + 1);
}
```

Figure 9: Calculating a page number from PDF page objects

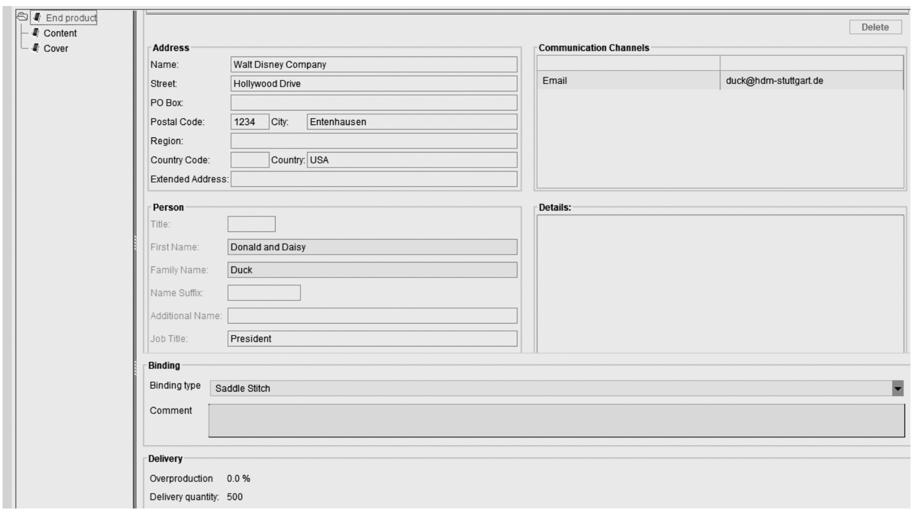
Figure 10 shows a snippet of the translation of the metadata to XJDF as shown in figure 7. Here the CIP4 library xJdfLib is used. In the first part, the resource MediaIntent is defined. The values CIP4_PaperCover and CIP4_PaperCoverWeight are derived from the PDF metadata and were originally put into the GUI of software ① by user. In the second section, the product part Cover is set and resource `MediaIntentCover` is added. The third paragraph determines the XJDF-structure with the `ProductList` (`broschur`, `cover` and `content`). The generation of the `ResourceSets` are skipped in figure 10. In the last part, finally, the PrintTalk envelope is built around the XJDF structure.

```
…
//Create MediaIntent for Cover
MediaIntent mediaIntentCover = new MediaIntent();
mediaIntentCover.setMediaType(MediaType.PAPER);
mediaIntentCover.setMediaQuality(CIP4_PaperCover);
mediaIntentCover.setNamedWeight(CIP4_PaperCoverWeight);
…
//create Cover and add mediaIntent for Cover
ProductBuilder productBuilderCover = new ProductBuilder(CIP4_Copy_Count);
Product cover = productBuilderCover.build();
cover.setDescriptiveName("Cover");
cover.setIsRoot(false);
cover.setProductType("Product");
cover.setID("ID_Cover");
cover.setProductType(CIP4_ProductTypeCover);
productBuilderCover.addIntent(mediaIntentCover);
…
// create xjdf and add product (parts)
XJdfBuilder xJdfBuilder = new XJdfBuilder(CIP4_JobID,"Product","Final Broschur");
xJdf = xJdfBuilder.build();
xJdfBuilder.addProduct(broschur);
xJdfBuilder.addProduct(cover);
xJdfBuilder.addProduct(content);
…
// create PrintTalk and embrace xJDF
PrintTalkNodeFactory ptkNf = new PrintTalkNodeFactory();
PrintTalkBuilder ptkBuilder = new PrintTalkBuilder();
PurchaseOrder purchaseOrder = ptkNf.createPurchaseOrder("4711", "Euro", xJdf);
ptkBuilder.addRequest(purchaseOrder);
PrintTalk printTalk = ptkBuilder.build();
…
```

Figure 10: Generating XJDF

3.5 Importing XJDF into a WMS

The XJDF file from figure 7 was put in a suitable hot-folder of the Heidelberger Prinect WMS (Heidelberg). Doing that a job with the two parts "content" and "cover" is automatically created. Figure 11 also shows that the customer details, the binding intent and the number of copies are already set.



Address

Name:	Walt Disney Company
Street:	Hollywood Drive
PO Box:	
Postal Code:	1234 City: Entenhausen
Region:	
Country Code:	Country: USA
Extended Address:	

Communication Channels

Email	duck@hdm-stuttgart.de

Person

Title:	
First Name:	Donald and Daisy
Family Name:	Duck
Name Suffix:	
Additional Name:	
Job Title:	President

Details:

Binding

Binding type: Saddle Stitch

Comment:

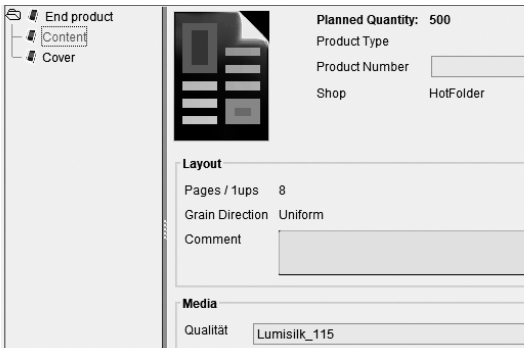
Delivery

Overproduction 0.0 %

Delivery quantity: 500

Figure 11: Prinect reads XJDF Resource "Contact"

One can observe in figure 12 that the printing substrate and the number of pages of the product part content has been recognized. The same applies for the cover.



Planned Quantity: 500
Product Type
Product Number
Shop HotFolder

Layout
Pages / 1ups 8
Grain Direction Uniform
Comment

Media
Qualität Lumisilk_115

Figure 12: Prinect reads XJDF Product Part "Content"

4. DISCUSSION

The specifications concerning the metadata used here are draft versions only (Meissner, 2018; ISO/DIS 21812-1) and thus subject to change. It is not advisable to start a product development based on this still unstable ground.
The outcome of the conversion has also been imported into a WMS. The metadata has been partly recognized.
The software ① and ② are not meant to become industry solutions. For that, the programming got to be much more error tolerant and more general. Reading the metadata from PDF/VT, for example, need to be implemented more universal and cannot assume a certain structure as it has been the case in our project. Thus, software ① and ② are working fine in conjunction, but ② will most likely fail if it reads PDF/VT with CIP4 metadata that has been generated by some other program and some other library. Moreover, we concentrated on a single product type, i.e. a booklet with cover and content. Even for this example, we did not address every aspect – for example, neither the colour properties of the product nor the actual formats are considered.

5. CONCLUSIONS

The feasibility of the interfaces is proven. The problem in practice will be the generation of the metadata. There are several obvious scenarios that are right for this new approach like Web-to-print or forwarding jobs to a subsidiary or partner of the PSP. It might become a bit more questionable, if the PB needs to fill out a form, as it is the case with this prototype software. In this scenario, a steady and firm collaboration between PB and PSP might be a necessary assumption for the start.

6. ACKNOWLEDGMENTS

I would like to thank PDFlib for providing me with a temporary license for their library.

7. REFERENCES

[1] Heidelberg, Prinect, Heidelberg,
URL: https://www.heidelberg.com/global/en/lifecycle/workflow/prinect_overview.jsp
(last request: 2018-07-18)
[2] International Organization for Standardization: ISO 16612-2:2010. "Graphic technology - Variable data exchange - Part 2: Using PDF/X-4 and PDF/X-5 (PDF/VT-1 and PDF/VT-2)", International Organization for Standardization, 2010.
[3] International Organization for Standardization: ISO 16684-1:2012. "Graphic Technology – Extensible Metadata Platform (XMP) specification", International Organization for Standardization, 2012.

[4] International Organization for Standardization: ISO 32000-2:2017. “Document Management - Portable Document Format - Part 2: PDF 2.0”, International Organization for Standardization, 2017.
[5] International Organization for Standardization: ISO/DIS 21812-1. “Graphic technology — Digital data exchange — Print product metadata for PDF files — Part 1: Architecture and core requirements for metadata”, International Organization for Standardization, Draft 2018.
[6] Jetbrains, IntelliJIDEA, Jetbrains, URL: https://www.jetbrains.com/idea/ (last request: 2018-07-19)
[7] Meissner, S.: “ICS-IntentMetadata.PDF.1,5”, confluence.cip4, URL: https://confluence.cip4.org/display/PUB/XJDF, (last request: 2018-05-13)
[8] Meissner, S.: “XJDF Specification 2.0 – final”, confluence.cip4, URL: https://confluence.cip4.org/display/PUB/XJDF (last request: 2018-07-09)
[9] Meissner, S.: “Print Talk”, confluence.cip4, URL: https://confluence.cip4.org/display/PUB/PrintTalk (last request: 2018-05-13)
[10] Meissner, S.: “JDF Specification 1.6-final”, confluence.cip4, URL: https://confluence.cip4.org/display/PUB/XJDF (last request: 2018-05-16)
[11] Prosi, R.: “ICS — Common Metadata for Document Production Workflows”, confluence.cip4, URL: https://confluence.cip4.org/display/PUB/ICS+Documents (last request: 2018-05-13)
[12] Wikipedia, CSV (Dateiformat), Wikipedia, URL: https://de.wikipedia.org/wiki/CSV_(Dateiformat) (last request: 2018-05-25)