4 Metadata Formats

Abstract *Chapter 4 is about metadata formats in the printing industry. First, XMP (Extensible Metadata Platform) is presented. Subsequently, the Job Definition Format (JDF), the Exchange JDF (XJDF) and PrintTalk are explained in detail. The last section of this chapter is about PDF metadata relevant for print production. The basic concepts of these metadata formats are explained, not so much the actual coding.*

Keywords *JDF, XJDF, XMP, PrintTalk, PDF Metadata, Job Ticket.*

Metadata is structured data that describes the characteristics of other data (such as files). Properties of individual objects are also called metadata. Such objects could be resources or processes and much more. In this sense, the job tickets we talked about in Section 2.1.1 are examples of metadata. Other examples may describe a photo, the copyright status of a text, ISBN numbers, part numbers, serial numbers, drawing numbers, codes, and so forth. Here are four typical examples of metadata in the printing industry:

- Object descriptions (“The photo has a resolution of 350 ppi”)
- Product descriptions (“Adhesive binding is required”)
- Instructions (“Fold signature 1 according to F-16-6 in the fold catalog”)
- Operational data (“A warning occurred during the color conversion of some content file”)

Definition

Strictly speaking, metadata describes properties of other data. For example, if the data represents a document that includes text and formatting rules, the corresponding metadata might contain properties such as the author’s name, a copyright notice, and the creation date..

In the graphic arts industry, metadata is often seen in contrast to content data (also called print data or assets). The latter describes the artworks that can be looked at on the printing substrate after printing. They are typically stored as image data, graphic data, text data, layout data, or coded in a page description language.

Metadata is either embedded in content data or stored separately from the content data, i.e. in a separate file or in a database. Most photos do not only contain the color values of pixels, but also a lot of metadata that the camera automatically generates (such as information about the camera; camera settings while the picture was taken; date, time, and GPS location). This kind of metadata is embedded, but a database might extract and store it internally in a later stage. Most content data formats allow storing metadata internally. We will cover XMP as an example of such metadata in Section 4.1 because it is occasionally used for workflow automation. If the metadata is stored externally, the data must include a reference to the content. In Sections 4.2 to 4.5, we will discuss metadata that is stored separately from the content data. In Section 4.6, I will present some less known metadata inside PDF files.

T. Hoffmann-Walbeck, *Workflow Automation*, SpringerBriefs in Applied Sciences and Technology, https://doi.org/10.1007/978-3-030-84782-1_4

Figure 4.1 shows a snippet of a sample JDF file that refers to a PDF file.

```
...
<RunList Run="1" Status="Draft">
   <LayoutElement>
      <FileSpec MimeType="application/pdf"
         URL="file://jobs/...sample.pdf" />
   </LayoutElement>
</RunList>
...
```

Figure 4.1: Sample JDF data which refers to PDF data

The circle in Figure 4.2 represents the metadata "hub". This is not a physical hub like those in network technology, but rather a metaphorical one. The sketch shows merely that different instances use various metadata. The print provider's production devices use metadata, but so do the print provider's business software and his partners. In a production environment for printed products, different metadata formats may be used. The data is frequently converted from one format to some other.

The most important metadata formats, of course, are JDF, XJDF, and PrintTalk. We will cover these formats in this chapter. However, as mentioned before, there are others as well. The good thing is that regular operators in a print shop don't need to worry about them. Graphical user interfaces of a workflow management system, a controller or the like obscure these technical data formats. You need to learn about these formats only if you want to extend an existing workflow, integrate a new device into the workflow

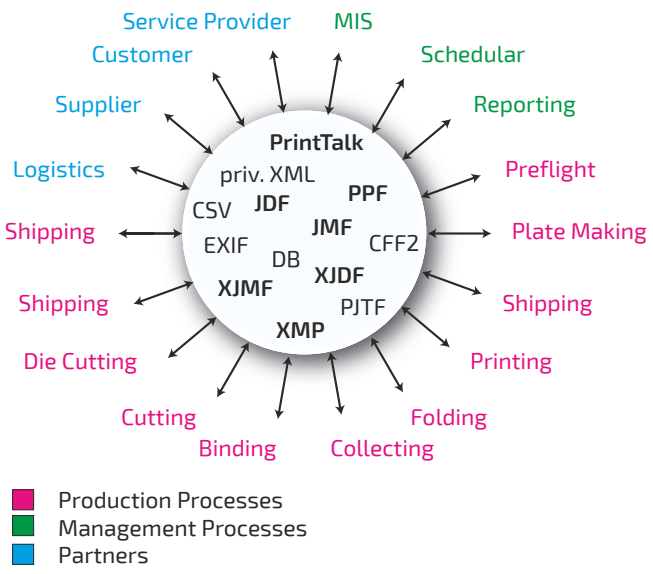

Figure 4.2: A print provider's metadata "hub"

automation network, or want to find out what is happening under the hood.

Nowadays, metadata is mostly encoded in XML or JSON. However, PPF is coded in PostScript, PJTF in PDF, CFF2 is a text format, and EXIF has its own specific binary structure. All examples in this chapter are encoded in XML; other encodings are ignored.

Figure 4.2 shows a selection of common data formats. Color profiles or preflight profiles, for instance, are not mentioned. Moreover, many companies specify private formats on their own. Why are there so many different formats? One reason is the evolution of these formats. PPF or PJTF are a bit outdated already, but they are still around, especially PPF, which is better known as the CIP3 format. Prepress often encodes ink zone presetting values in PPF/CIP3 before forwarding to press.

Another reason for the large number of metadata formats is the fact that they inhabit different ecosystems (see Figure 4.3). Some of them are prepress formats only, such as EXIF and XMP. JDF and XJDF have the widest habitat. The chart gives the impression that these formats are not being deployed between print buyer and print provider, but that is not quite true. A PrintTalk element describes business objects, which relate to products, of course. The product description inside the PrintTalk element can be either in JDF or XJDF.

The most important feature of a metadata format is what exactly it can describe. CFF2 is used for structural design in packaging, EXIF data is dedicated to image and audio data only, and XMP mainly

Note

PJTF stands for **Portable Job Ticket Format**. Adobe published the specification in 1999.

CFF2 stands for **Common File Format Version 2**. It is a text format for CAD data.

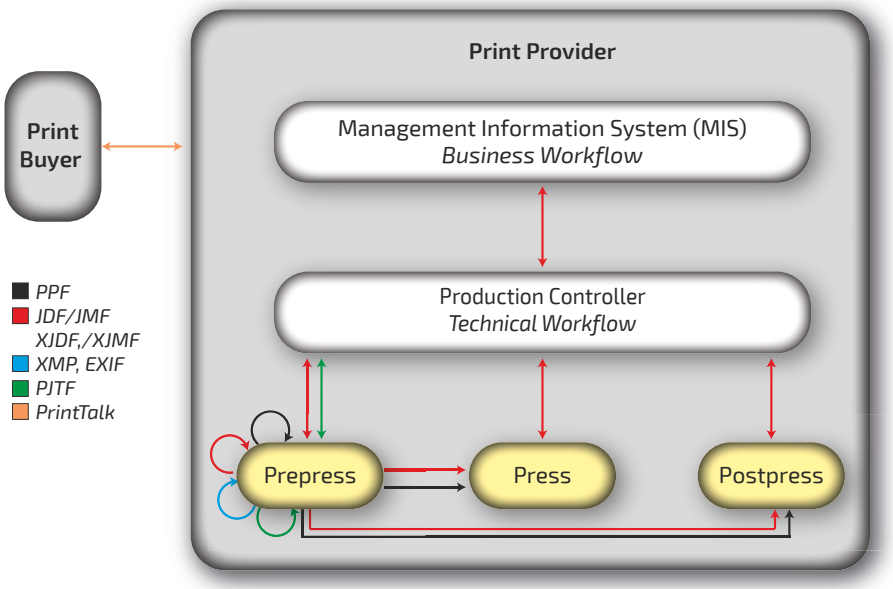

Figure 4.3: Metadata formats and their ecosystems

describes (elements of) a layout. Only JDF, XJDF and their associated messaging formats JMF and XJMF cover (almost) the entire print production scope.

In the following, we will look at the most prominent standard metadata formats only.

4.1 Extensible Metadata Platform (XMP)

Adobe Systems announced a new metadata format called *Extensible Metadata Platform (XMP)* in 2001 and released a specification in 2004. In 2008 and 2010, they split the specification into three parts. The currently valid specifications for Part 1, Part 2, and Part 3 can be found in (Adobe 2012), (Adobe 2016) and (Adobe 2020), respectively. In 2012, XMP Part 1 became also an ISO standard. The updated standard is (ISO 2019a).

Content elements such as images or graphics as well as layout files and PDF files often contain metadata entries such as copyright notices, the author's name and contact details, and keywords. They are often stored in the XMP data structure.

One application of an XMP workflow involves retrieving a caption that is registered within an image or a graphic. Often, an author delivers text and images/graphics to an agency or publisher, which then performs the layout. Traditionally, the author supplies the captions in a separate text file, and the layout artist copies them to the suitable figure. This is inefficient and, above all, error-prone. It is much more efficient if the author enters the captions in the XMP structure of the images/graphics, and the layout artist retrieves it from there.

The basic idea of XMP is that metadata persists when embedded in or linked to another file and during format conversions. This is the reason why in our previous example the layout program can extract the caption from the image/graphic. Figure 4.4 shows the concept.

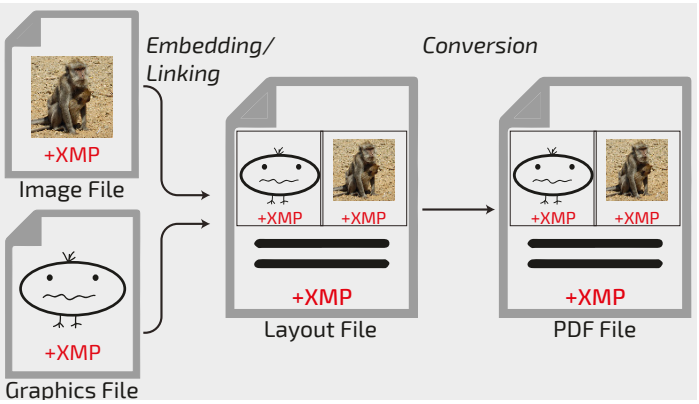

Figure 4.4: XMP metadata of a file often persist when embedded or converted into or another file.

It should be noted that not only document files can be tagged with metadata, but also certain components of a document. This includes, for instance, images that are placed in the layout or embedded in a PDF document. However, individual pages, paragraphs, words or even letters are not document components to which you can attach your own metadata. The

XMP structures of the components are still "attached" to the individual components. That is, they are not merged into one common data structure. In addition, the document itself may have XMP metadata, as indicated in Figure 4.4.

As already mentioned, format conversions should preserve XMP metadata. An important application for this is the copyright entry in images. For example, when reprinting a brochure, an advertising agency may have only the PDF data that was used for printing, but no longer the image data embedded in it. Nevertheless, the agency can still extract from the PDF the XMP metadata and thus the copyright status of the images.

Unfortunately, you cannot rely on the XMP data to be maintained during a format conversion. There are two reasons for this:

- Not all data formats can store XMP. The metadata can be embedded, for example, in PDF, PostScript, TIFF, JEPG (2000), GIF, PNG and SVG files. A somewhat more complete list can be found in (Adobe 2020). The XMP data structure is an addition to a file. This means that the extensibility of a data format is a pre-condition for embedding XMP. Since the BMP image format does not have this property, for instance, it is not capable of having XMP metadata embedded.
- An XMP structure is an uncompressed XML text block within a mostly binary-encoded and compressed file (see Figure 4.6). This allows XMP reading software to easily extract the XMP data without knowing the internal structure of the file. Of course, when including an XMP structure in a binary file, the structure of the file format must still be respected. The text cannot be arbitrarily incorporated into the binary file. Figure 4.5 shows the basic integration of an XMP record into PDF. In this format, everything is strictly enclosed by PDF objects. The first object with ID 13 contains the image data. Moreover, there is a reference to some metadata object 14. Object 14 holds the corresponding XMP metadata

  Accordingly, applications must support the XMP metadata format, especially when writing the structure into some data. Since Adobe originally specified XMP, it is not surprising that Adobe's applications allow XMP entries. However, many applications do not support XMP even if they generate data formats that can embed XMP.

When in doubt, test in advance if a workflow is to be based on XMP.

Note

The BMP raster data format introduced by Microsoft in 1990 does not support XMP metadata up to and including version 3.

Metadata in PDF

```
13 0 obj
<<
/Subtype/Image
Metadata 14 0 R
...
>>
stream
...(Image Data)
endstream
endobj
```

Object 12: Image Data

```
14 0 obj
<<
/Subtype/XML/
/Type/Metadata
...(XMP-Metadata)
>>
stream
...
endstream
endobj
```

Object 14: XMP Data

Figure 4.5: Two internal PDF objects. Object 12 is an image; object 14 contains the corresponding XMP structure.

Where do the XML metadata entries actually come from? There are various options:

- The most obvious way is to enter values (for example, author's name, copyright, keywords, etc.) into a specific file manually via in an XMP-supporting application.
- Applying XMP entries in batches, for example, in all files in a specific folder. A few applications allow this approach.
- The XMP entries come from another metadata format. For example, digital cameras produce metadata about the camera and camera settings while taking the picture. The values are stored in the *EXchangeable Image File* (EXIF) format (CIPA 2016), a well-known and widely used metadata format for digital cameras. These entries are converted to XMP automatically when the photo is imported into certain image processing software, provided the software supports it.
- Photos are often stored in a database, such as in a *Digital Asset Management* (DAM) or a *Media Asset Management* (MAM) system. The database reads XMP or EXIF data from the pictures and stores the data internally in order to characterize the images. Conversely, data that is added in the database (e.g., generated automatically by means of AI image analysis) can be stored in the XMP structure of the images. Note that XMP does not have to be embedded in the files but can be saved as a separate text file. These files can also be used as a backup.
- XMP is extensible, after all the "X" in the name stands for "eXtensible". This means that companies can easily extend the XMP structure with their own metadata. For instance, image processing software can store correction steps in XMP. Most importantly, a prepress workflow management system can store internal production details.

Definition

A **namespace** and an associated **prefix** is specified by an *xmlns* attribute. XML elements can then be placed in any namespace by a prefix, which is separated from the element name by a colon.

As mentioned earlier, what makes the XMP special is its XML structure, which can be embedded in different data formats. As known in XML, the *xmlns* attribute defines XML *namespaces* and their *prefixes*. This allows new XML elements to be defined without having to worry about already existing element names. The prefixes allow identical element names as long as the prefixes are different. In Figure 4.6, the prefixes are marked in red. In the example, only reserved prefix names occur (tiff, exif, photoshop, Iptc4xmpcore). There are a few more of them. For these, the element names are fixed.

A software vendor can define its own namespace and thus its own XML substructure. Other manufacturers will usually ignore these

```
<rdf:Description rdf:about=""
   xmlns:tiff="http://ns.adobe.com/tiff/1.0/"
   xmlns:exif="http://ns.adobe.com/exif/1.0/"
   xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/"
   xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/"
   ...
   tiff:XResolution="3140000/10000"
   tiff:YResolution="3140000/10000"
   ...
   exif:PixelXDimension="3072"
   exif:PixelYDimension="2304"
   exif:DateTimeOriginal="2007-07-28T13:39:44+02:00"
   ...
   photoshop:CaptionWriter="Thomas Hoffmann-Walbeck"
   photoshop:Headline="Brooklin"
   ...
   Iptc4xmpCore:CiTelWork="+711-7801714"
   Iptc4xmpCore:CiEmailWork="hofmann@hdm-stuttgart.de"
</rdf:Description>
```

Figure 4.6
XMP is coded in XML

entries, but the vendor's own software can of course access the data at any time. This is a proven means of data communication within a workflow management system. As an example of such entries, I would like to mention an EAN's code, size, color, and position. For a software developer, it is easy to read XMP data from a file or to write XMP data into a file.

The XMP structure is enclosed by an XML element called *rdf*. The term stands for *Resource Description Framework*. This XML schema was adopted by the W3C in 1999 as a recommendation for storing resource metadata.

Let us wrap up: In most cases, XMP data is created to store information about a file and/or its components across applications. This supports search engines as well as contact information and copyright notices. That is, XMP is a descriptive metadata format for prepress. In the context of this booklet, however, the temporary storage of private process data is even more important. It is not standardized and therefore incomprehensible for software from different manufacturers. However, in the end, one could convert the most relevant of the private XMP data into a standard format such as JDF (after several internal process phases have been executed using XMP).

For testing purposes, you can use Adobe software to create an XMP extension without programming.

Definition

An **XML schema** is a description of the structure of an XML document type. In particular, a schema defines the names of elements and attributes, the hierarchy of elements and the data types of attribute values.

4.2 Print Production Format (PPF)

We now come to the standard job ticket metadata formats, which are specifically specified for the description of print products and for controlling print production.

The Print Production Format (PPF) is considered the predecessor format of JDF. It was developed by the CIP3 organization, the predecessor organization of CIP4. Therefore, the PPF is often referred to as the *CIP3* format. It is encoded in the *PostScript* computer language. New structured elements have been defined to describe PPF content. They all start with "CIP3" as shown in Figure 4.7.

```
CIP3BeginPreviewImage
%%Page: 1
%%PlateColor: Cyan
CIP3BeginSeparation
/CIP3PreviewImageWidth 1490 def
/CIP3PreviewImageHeight 1210 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [1490 0 0 -1210 0 1210] def
/CIP3PreviewImageResolution [ 50.800 50.800 ] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageDataSize 515348 def
CIP3BeginPreviewImage …pixels of image
CIP3EndPreviewImage
CIP3EndSeparation
      …analog for all separations…
CIP3EndPreviewImage
```

Figure 4.7: PPF snippet defining a preview image in PPF (PostScript)

The reasons why it was decided to specify JDF and JMF formats and not to try to supplement the print production format are many and manifold. In particular:

- A major shortcoming of the Print Production Format is its lack of support for order processing (MIS/ERP). PPF is used almost exclusively for technical production control.
- The PPF does not support shop floor data collection.
- There is no message format for dynamic interaction (such as the Job Messaging Format).
- PPF is coded in PostScript and therefore relatively difficult to interpret and edit.
- There is no defined mechanism for separating and merging parts of a PPF. This makes central PPF storage for different workflow components cumbersome. Instead, several differ-

ent PPF files are exchanged from and to different workflow components via various hot folders.

In essence, JDF has replaced PPF. There is only one application in which the PPF is still very popular, namely in the transfer of prepress data for the ink zone presetting in offset printing. Prepress provides the low-res image preview and transfer curves. From this, a press application can calculate the color zone preset by first applying the transfer curve to the preview, then counting the percentage of color pixels in each ink zone and for each separation. From the percentage of ink zones, the positions of the servomotors for the ink feed of the press must then be determined, depending on the paper.

Definition

A **transfer curve** is a curve/table that defines modifications of tonal values of the artwork during RIPing to match desired tonal value increases in print – for example, to reach an offset standard.

Figure 4.7 shows a PPF file that describes a low-res preview image, which is suitable for calculating the ink zone presetting values. The example describes the following:

- A preview image with separations (*CIP3PreviewImageComponents*).
- The data is run-length compressed (*CIP3PreviewImageCompression*) and binary-encoded (*CIP3PreviewImageEncoding*).
- The width (*CIP3PreviewImageWidth*) and the height (*CIP3PreviewImageHeight*) are given in pixels.
- The resolution (*CIP3PreviewImageResolution*) is given in pixels per inch (ppi).

The resolution is 50.8 ppi. The *CIP3PreviewImageMatrix* specifies the pixel direction in the image; here it is defined from left to right and next from top to bottom. The pixel values are stored inside the PPF after the *CIP3PreviewImage* element.

4.3 Job Definition Format (JDF)

The introduction of JDF and the associated *Job Messaging Format* (JMF) by the *Cooperation for the Integration of Processes in Prepress, Press and Postpress* (CIP4) in the year 2000 was a major innovation. Through the subsequent JDF implementations, workflow automation in the print production has increased significantly.

First, let's take a look at the simple example of JDF integration in Figure 4.8. The MIS initially records quite a lot of data about the print product and its production during the estimating of a quote. If this estimation converts to a print job, some of the data will eventually be sent to prepress, press, and postpress, such as details about the intended product and some business data (such as customer details, delivery date, etc.). First, prepress will process

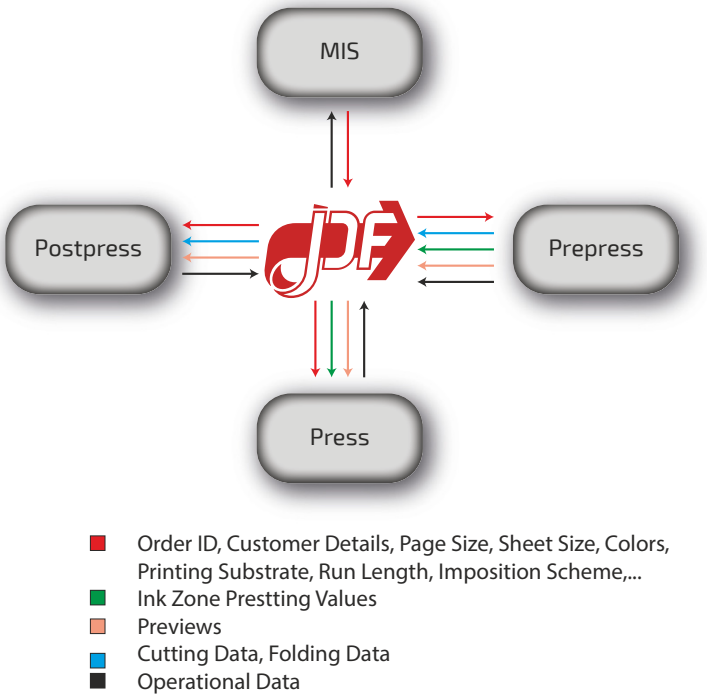

Figure 4.8: Example of information exchange with JDF

the JDF and add new entries, such as ink zone presetting values for the press. The RIP might also generate composite previews for press and postpress, for example, for quality assurance purposes. Moreover, the imposition software might forward cutting and folding positions to the guillotine and the folder. Finally, all these devices should send operational data back to the MIS.

We'll talk about the distribution of JDF later. For now, it should be sufficient to have a JDF pool of some sort as a hub.

What do the arrows in the Figure 4.8 are present? Note that the information content can vary regarding its details. For instance, if only the order number is transmitted, the level of detail is low. If, however, many production details are transferred so that a subsequent process can run fully automatically, the level of detail is high. All JDF interfaces usually operate between these two extreme positions.

This circumstance also explains why workflow managers in print shops and at manufacturers should be well versed in interface details. The benefit of JDF integration depends strongly on the level of detail in the JDF data.

Before discussing the details of JDF and JMF data structures, it should be mentioned that JDF workflow systems are usually configured very individually for a print shop. Since JDF workflow systems tend to affect many areas of a print shop, existing components are usually integrated.

However, JDF and JMF are merely standardized data formats and not specifications for workflow systems. This means that components from different manufacturers communicate with each other via JDF/JMF but otherwise actually act independently of each other. This can lead to incompatibilities that must be solved case by case (see Section 4.6). In particular, this means that JDF/JMF integration is not an off-the-shelf software package that can be purchased and simply installed, but a project that must be pursued in several stages and over the long term.

Consequently, setting up a JDF workflow system requires a great deal of commitment, expertise and project management skills.

In a JDF-based workflow, a specific JDF file is created for each print job, sometimes in multiple versions as production progresses. As Figure 4.8 suggests, the MIS often writes the first JDF file. Usually, it contains a description of the intended product and rough production definitions without much detail. As processes add additional data to the JDF during production, the file size will consequently increase.

4.3.1 JDF Nodes

JDF and JMF are XML-encoded. That is, each JDF or JMF record consists of a tree of XML elements and their attributes and values. The permitted XML element names are defined in the JDF specification (CIP4 2020). Since JSON has become increasingly popular in the last decade, CIP4 is currently working on a JSON representation of JDF/XJDF.

First, let us talk about the XML elements for JDF nodes. They define the product to be manufactured and, if applicable, its sub-products, as well as the production processes needed to manufacture this product.

Each JDF node has the attribute *Type*. The value of this attribute can be either *Product*, *ProcessGroup*, *CombinedProcess*, or some specific process name such as *ImageSetting, ConventionalPrinting*, or *Folding*. CIP4 defines over 100 different processes.

For a standard JDF file, the root JDF node has the value *Product*

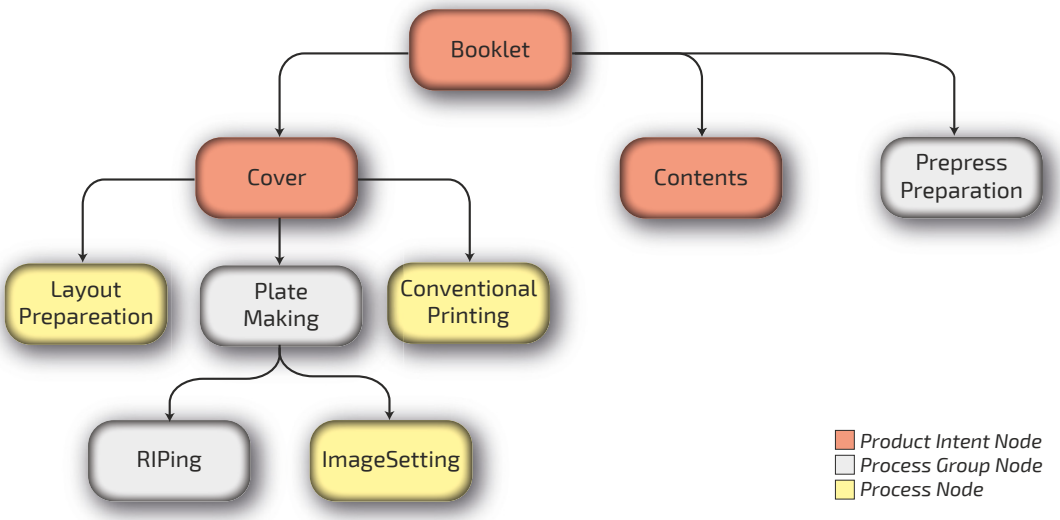

Figure 4.9 Example of a JDF Node tree for the production of a booklet

for the attribute *Type*. This node is called a *Product Intent Node*. It represents the final product. Underneath the root, there can be more *Product Intent Nodes*. These represent sub-products such as cover, content, and the like. Below the *Product Intent Nodes* there are *Process Group Nodes* and *Process Nodes*. They describe for each sub-product the processes that are required for its production. Please note:

- The node hierarchy is not limited. Figure 4.9 shows merely an example.
- As usual, the hierarchy of an XML tree is defined by creating subnodes. That is, the root node in our example contains two JDF child nodes of type *Product*, which in turn contain further child nodes. A *Process Node* is always a leaf of the tree (i.e., it does not contain any further subnodes), while *Process Group Nodes* can optionally contain further subnodes (further *Process Group Nodes* or *Process Nodes*).
- The *Process Nodes* and the *Process Group Node*s below some *Product Intent Node*, which represents a product part, refer only to the production of this product part.
- Not all processes that are needed to produce a print product need to be listed in the JDF file. In general, only those processes which are included in the JDF workflow will be defined.
- The JDF file reflects only the current situation at a given point in production. In general, more nodes may be added as production progresses.
- In a real production environment, the node tree is often much more complex than in this example.
- The resources are not shown in the JDF node tree.

```
<FoldingParams
   FoldCatalog="F16-6"
   SheetLay="Left" />
</FoldingParams>
```

Figure 4.10
Resource FoldingParams

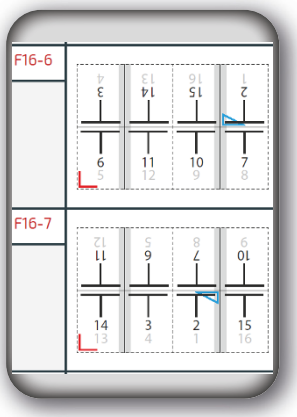

Figure 4.11: Two folding schemes from the CIP4 fold catalog. The catalog contains almost 100 schemes.

4.3.2 JDF Resources

Each JDF node can contain an XML element with the name *ResourcePool*, in which resources are stored. All resources of a JDF file are incorporated in a ResourcePool of some node.

CIP4 defines 170 different resources in the latest specification (CIP4 2020). Each resource is defined in detail with all possible options. The generally valid descriptions of these processes and resources are in themselves a valuable source of information. At the same time, such definitions are not easy to create. For example, how can you uniquely define a sheet layout? The code example in 4.10 shows the resource *FoldingParams*. This resource looks very simple. It merely defines that the reference edge where the paper is placed on the folder is on the left-hand side and the folding scheme is according to catalog number F16-6. The real

achievement was to define a unique fold catalog. This catalog can be found in the JDF specification (CIP4 2010). It includes all common folding schemes, almost 100 different ones in total. Figure 4.11 shows the folding schemes F16-6 and F16-7.

The specification of the folding catalog number does not suffice for an automatic folding machine preset. In the *FoldingParams* resource, the exact folding position can optionally be specified. Only then can a folding machine preset itself accurately according to the specifications in prepress (ignoring the paper distortion during printing). Again, this shows how important it is to know the information details of interfaces.

Input resources of a Process Node or a Process Group Node are the physical items such as plates, electronic items such as files, or conceptional items such as a parameter set that a process or process group needs to observe. An output resource is an item that the process or process group generates during execution. That is, a process consumes input resources and produces output resources. Thus, JDF is based on the process-resource model that we dealt with in Chapter 3. But what about *Product Intent Nodes*? Do they have input and output resources as well?

The answer to this is yes. The interpretations of those resources differ, though. The output resource of a *Product Intent Node* is always called *Component* and represents the final product or a product part - similar to the node itself. A Component can actually be an input resource for another Product Intent Node one level up in the hierarchy. For example, the Component that represents a cover is an input resource of the Product Intent Node representing the final booklet. The input resources of a Product Intent Node, which is not transitional (see Section 3.3, i.e., is not an output node of another Product Intent Node) is called *Intent Resource*. An Intent

Definition

A JDF **Component** describes the various versions of (semi-) finished goods. The final product, product parts, but also a set of printed sheets or a pile of folded sheets are considered components.

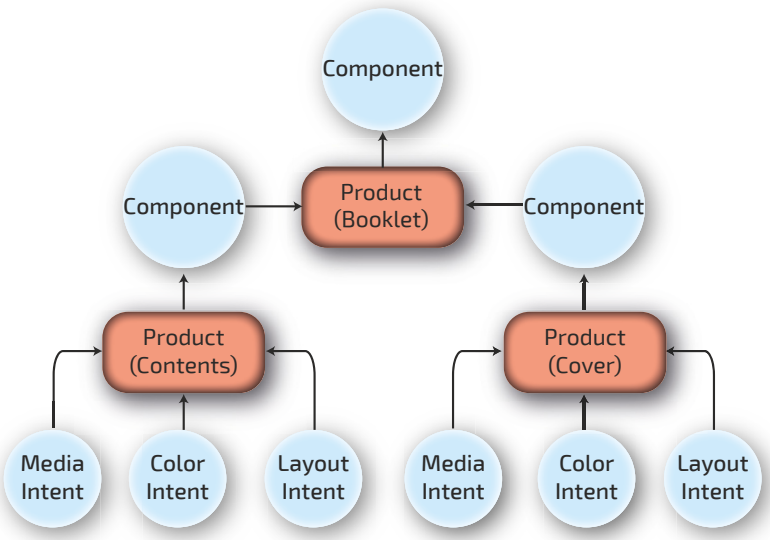

Figure 4.12: Example of Product Intent Nodes and their input and output resources

Resource specifies the print buyer's intention for the product (part), i.e., details of the product (parts) without defining the processes needed to make them. Figure 4.12 explains the terminology. Note that here the arrows do not represent relationships between JDF nodes as before, but the relationships between resources and nodes. The *MediaIntent* defines the printing substrate, the *ColorIntent* the separations for this job part, and *LayoutIntent* the number and dimensions of the pages.

In the remainder of this section, we will talk about *Partitionable Resources*. In general, one can talk about things in their entirety, as certain subsets, or as singularities. For example, one can talk about books in general, about certain categories of books, or about a particular book. This is similar to some resources for a print job. For example, the resource *ExposedMedia* specifies exposed printing plates (among other things). You may want to specify all plates for a print job or only a subset, i.e.:

- All plates of the job
- All plates of a signature (of the job)
- All plates of a sheet (of a signature and the job)
- all plates of a side (of a sheet, a signature, and the job)
- A specific separation (of a side, a sheet, a signature, and the job)

These kinds of resources are called *Partitionable Resources* in JDF terminology. The specification details how certain resources can be partitioned. Note that the partitioning may not always be identical for different resources. For instance, print sheets cannot be partitioned up to the separation, as was previously the case with the printing plates.

Another example is the printing substrate, which can be the same for the entire job, but often differs based on cover and content. Of course, the content can also be printed on differing substrates. With the help of partitions, all this can be defined easily and flexibly.

4.3.3 Structure of a JDF File

Product descriptions and processes are coded via JDF nodes. We already learned that resources are pooled in XML elements called *ResourcePools,* and that those elements are sub-elements of JDF nodes. Thus, every JDF node, no matter what type, can optionally contain a *ResourcePool*. However, not every resource that is either input or output of a JDF node necessarily resides in the *ResourcePool* of that node. The reason for this is that resources can have relationships with multiple JDF nodes, in particular transitional resources. In order to avoid having to keep (and update) resources

in multiple copies in different JDF nodes, JDF nodes may also have input or output resources which lie outside of the particular JDF node.

The logical consequence of this is that there must be some sepa-

```
<?xml.....?>
<JDF Type="..." ...> ...
      <ResourcePool>...
      </ResourcePool>
      ...
      <ResourceLinkPool>...
      </ResourceLinkPool>
      ...
      <JDF Type="..." ...>
            ...    other JDF sub node(s)
      </JDF> ...
</JDF>
```

Figure 4.13: Structure of a JDF file

rate information that specifies which resources a JDF node is linked to. This information is called *ResourceLink*, where all *ResourceLinks* of a JDF node are collected in a *ResourceLinkPool*. A *ResourceLinkPool* is a subelement of a JDF node. This results in the JDF file structure described in Figure 4.13.

4.3.4 Audit Pool

In Figure 4.13, an important pool in the JDF node is missing. In addition to the *ResourcePool* and the *ResourceLinkPool*, a process node can also contain an *AuditPool*. *Audit Elements* are written to this pool so that the process results can be recorded after execution. Typical contents of an *Audit Element* are:

- Generation, modification or deletion of a JDF node
- Process times (Start, End, etc.)
- End status (Completed, Aborted, Stopped, etc.)

```
<AuditPool>
      …
   <ProcessRun
      AgentName=    "CIP4 JDF Writer Java"
      End=          "2020-12-05T13:53:52+01:00"
      EndStatus=    "Completed"
      Start=        "2020-12-05T13:53:43+01:00"/>
      …
</AuditPool>
```

Figure 4.14: Code snippet of a ProcessRun Audit Element in an AuditPool

- Errors (Warning, Fatal, Error, etc.)
- Reaching of a milestone
- Consumed or missing resources

With the contents of the *AuditPool*, an MIS can post-calculate the print job, for example. Figure 4.14 shows a small code excerpt of an *AuditPool*.

The attribute *AgentName* holds the name of the application that added the Audit Element to the *AuditPool.* Beside the shown *Audit Element* of type *ProcessRun*, there are others, such as:

- *PhaseTime* for logging start and end times of any process states and process phases
- *Notification* for logging events (such as errors) during process execution
- *ResourceAudit* for logging the usage of resources during execution
- *Created, Modified,* and *Deleted* for logging the creation, modification, and deletion of a JDF node or a resource

4.3.5 Gray Boxes

As mentioned before, an MIS normally cannot describe details of the production, only a rough framework. To describe such framework, special Process Group Nodes have been specified. They are called *Gray Boxes*. A Gray Box does not define all processes or all resources for the embraced processes, except for the (final) output resources. The information contained in a Gray Box does not suffice to execute the processes that the Gray Box holds, which is why a Gray Box is called non-executable. The missing data must be added as the production progresses. If all necessary data is available, the Gray Box is dissolved and may become a normal Process Group, for example.

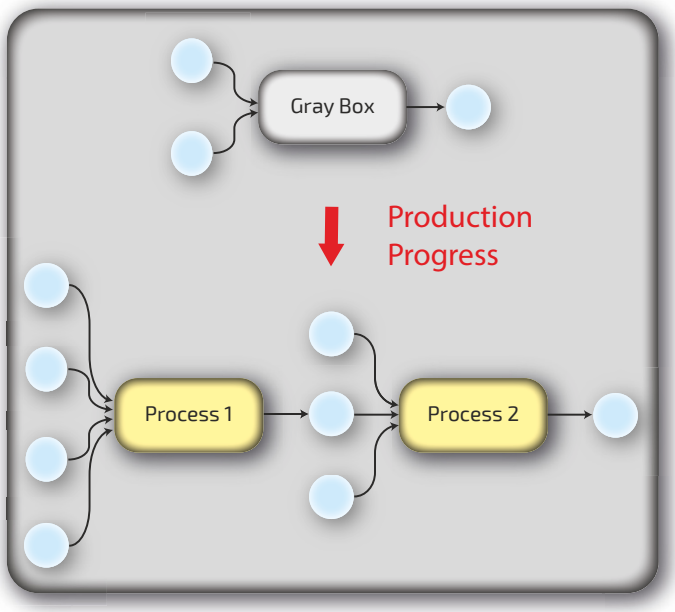

Figure 4.15: A Gray Box must become (a group of) processes with all mandatory entries and resources before it can be executed.

You can recognize a Gray Box by the fact that it

a) is a process group, i.e. *Type*="*ProcessGroup*", and
b) has the attribute *Types* (note the "s"!).

The value in the *Types* attribute is normally an (incomplete) list of processes or sometimes a predefined name. The process list gives an indication of which processes (among others) the Gray Box should be resolved to. The predefined names are specified in the *ICS* papers (see Section 4.6). The Gray Box name *PrePressPreparation* would be such an example. Further options and details are specified in the *ICS* papers.

Note

ICS stands for Interoperability Conformance Specification.

4.3.6 Spawning and Merging

In a print shop, products are produced by parallel or overlapping processes. For example, layout elements such as text, images and graphics are created in parallel (in JDF terms: *LayoutElementProduction*). An example of overlapping execution is that while the set of plates for a print product is still being imaged, some of the previously exposed plates are already used for printing.

This implies that the JDF structures (nodes and resources) of a JDF file must be sent to different controllers/devices simultaneously. This would not be a problem in itself, but keep in mind that the devices (should) provide the *AuditPools* with fresh audits. Resources might also be updated. For instance, only after RIPing can values for the ink zone presetting be calculated and stored in the corresponding resource. However, this can cause conflicts when the modified JDF structures are merged back into the original JDF file. This is JDF technology has a process called *"Spawning* and *Merging*". This mechanism requires that it must be recorded in the original JDF file if some part of the JDF structure is copied for a controller/device (*Spawning*). Whether the recipient has read-only or also write permissions is also important. A JDF structure may be checked out with write permissions only once at any given time. If the modified structure is merged back into the original file, it should be recorded once again. The JDF structure may then be spawned once more with full write permissions. This way, there are no more conflicts when merging. Figure 4.16 shows the principle of this mechanism.

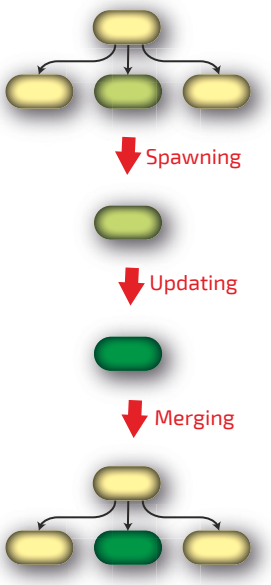

Figure 4.16: Spawning and merging of a JDF node

However, if two production lines operate in parallel, the processes are more likely to be modeled completely separately. For example, the cover and the contents of a brochure can be printed simultaneously on two different offset presses. This situation will be modeled via two different *ConventionalPrinting* processes.

4.3.7 Job Messaging Format (JMF)

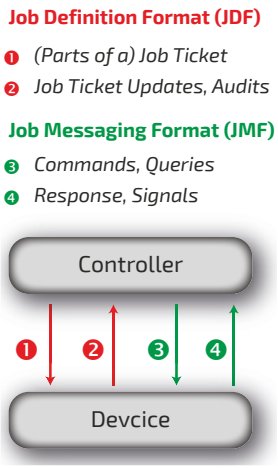

Figure 4.17: Key data exchange between controller and device

Now that we have discussed what kind of information can be stored in a JDF file, the question naturally arises: How do workflow instances exchange this data? In the simplest case, JDF job tickets are written by a JDF producer (see Figure 4.17-❶) and placed into the hot folder of a JDF consumer, who scans the folder for new data in certain intervals. This unidirectional interface via file transfer is, of course, quite static, slow, and does not allow real-time interaction. It also can become complex to maintain if many different workflow instances are involved. Real-time interaction, however, is necessary, if you think about Job tracking, job changes, instant error messages, and the display of actual material consumption or the degree of machine utilization.

Moreover, one should give up on the idea that a JDF file is copied to the input queue (or hot folder) of a controller/device via file transfer. The system sends a message containing the location of the JDF file instead (see Figure 4.17-❸). But how should one imagine such a message?

The Job Messaging Format (JMF) is part of the JDF specification. Like JDF, JMF is also XML-encoded. In the case of JMF, the root element is no longer JDF but rather JMF. JMF messages use the HTTP and HTTPS protocols, i.e., the JMF is the body data of an HTTP(S)

```
<JMF TimeStamp="..." SenderID="ID4711">
  <Command ID="M1" Type="SubmitQueueEntry">
    <QueueSubmissionParams>
      URL="File://Computer/Directory/job.jdf"/>
  </Command>
</JMF>
```

Figure 4.18: JMF command informing the recipient where to retrieve a JDF file

packet. The JMF code in Figure 4.18 shows an example of a JMF message. It contains a *Command* telling the recipient the location of the submitted JDF file *job.jdf*. Analogously, the recipient can also be told how to reach the JDF data via the HTTP protocol instead of via file transfer. More generally, numerous control messages concerning queues and their entries are important JMF applications. Others are:

- System bootstrapping and setup – for example, shutting down or waking up a device (which is in standby mode).
- Dynamic status, resource usage and error tracking for jobs and devices.
- Pipe control for overlapping processes. One process produces

resources while at the same time some other process consumes portions of these resources that are already available (see Section 2.4.5).
- Device setup and job changes – for example, creating new JDF nodes, which might be necessary if a workflow controller receives the artwork for a job but the MIS has not yet defined a JDF job ticket. The workflow controller will then send a *NewJDF* command to the MIS to initiate a new job. (See Section 2.2.1)
- *Device Capability* description for sending technical capabilities of a device to a controller, such as the minimum and maximum sheet size of a sheet-fed press.

Describing device capabilities can be quite tricky, though. The device capability might not be a fixed set of properties. Some machines (for example, a folding machine) can be optionally equipped with additional units, which changes the properties. In addition, properties can vary due to other factors. For example, the possible number of folds that a folding machine can perform depends on the paper thickness.

From a more abstract point of view, six different kinds of messages are available (called *JMF Message Families*):

- *Query*
- *Command*
- *Response*
- *Acknowledge*
- *Signal*
- *Registration*

A *Query* sends an inquiry to a recipient without changing its state, for instance, to retrieve information about a device's status or material consumption. A *Command*, however, changes the state of the addressee. An example is the deletion of a job in a queue or the submission of a JDF job ticket (see Figure 4.18). Both *Queries* and *Commands* require the receiver to send a *Response* message back to the sender. If a command (or query) takes a while to execute, it may be necessary to send a separate and asynchronous *Acknowledgment* message back to the sender after completion. A *Signal* is a unidirectional message, mostly from a device to a controller, usually about a status change. Signals are often forwarded automatically as "fire and forget" messages; that is, no responses are required. With a *Registration,* a controller can subscribe to certain signals from a device or another controller automatically, for instance, in certain time intervals. A controller may also request a

subscription on behalf of others. For example, an MIS might send a *Registration* to a prepress WMS in order to make sure that signals are sent to a press controller whenever a plate has been produced.

JMF messages are not only quite diverse but can also occur in large quantities. Therefore, let us conclude this section with the statement that JMF messages are mostly used for dashboards that visualize the current state of the shop floor. An MIS, on the other hand, uses the already summarized results in the *AuditPools* for post-calculation.

4.4 Exchange Job Definition Format (XJDF)

The JDF concept is over 20 years old already. Back then, the average run-lengths of jobs were much higher than today, and the average number of print jobs processed each day in a print shop was much lower. With a few dozens of print jobs per day, storing a separate JDF file for each print job is feasible. Today, however, a large online printer may run 10,000 jobs per day. Loading all these files into a workflow system in order to display the current jobs and their actual status would take much too long. Thus, a workflow controller such as an MIS operates a private database to speed up importing job data. This approach has been state-of the art for quite a while already. At the same time, the JDF handling has become more centralized. Either the MIS or a workflow controller is in charge of the JDF job tickets. This JDF master controller has many tasks. It must monitor hot folders and HTTP ports for incoming data. It must also send out data to other controllers and devices. To do this, it must monitor spawning and merging. It would be JDF compliant if a complete JDF file were transmitted to a device or another controller. Each device must be prepared to retrieve the required data from the file on its own, if necessary. However, this would be unfavorable since the complete JDF data set would be "blocked". Therefore, it is more efficient if the software sends only those JDF parts to a device that it needs. Thus, the master controller must be able to pick out the relevant parts for each device (usually one or more JDF nodes and the corresponding resources). In this case, JDF is a communication protocol between controller and devices. The workflow logic will be stored partly in the JDF files and partly in the controller's private database. Often, the workflow logic is merely part of the MIS and systems use JDF as a pure information interchange technology.

Definition

The **workflow logic** (also called **process logic**) is the description of rules and relationships between processes and resources that are needed for print production. In other words, it is the JDF's underlying process-resource model.

In 2018, the new Exchange JDF (XJDF) was published. The current version is from 2020 (CIP4 2020b). This new format reflects the changes in the JDF workflow implementations that we just discussed. XJDF is not an electronic job ticket like JDF but rather an interface specification. It is merely a protocol between two workflow components, primarily between controller and devices. It does not contain the workflow logic anymore, as JDF did. The workflow

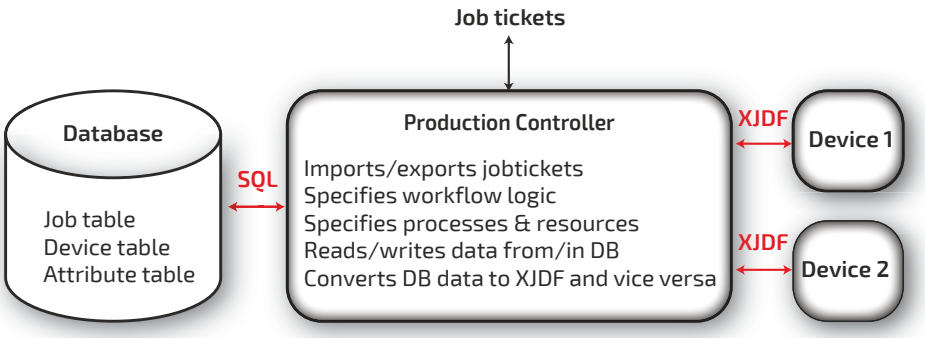

Figure 4.19: Simplified XJDF architecture

logic is now hidden in the internal data structure of the workflow system. This workflow system could be an MIS or some production control system. The internal data structure is undisclosed.

At first glance, it seems as if nothing of the "old" JDF has remained after the redesign. This is deceptive, though. The descriptions of the processes and resources have essentially remained the same. Only the connections between the processes (using the transitional resources) are no longer mentioned in XJDF. Moreover, there is no complete description for the print production at the end – at least not in XJDF. In fact, XJDF is only generated at runtime, then forwarded and read. Afterwards, it is discarded. There is no XJDF storage.

As a result, XJDF is much less complex:

- Since the XJDF addresses a single device only, an XJDF node has no children. Therefore, the nested node tree we saw in JDF no longer exists (see Section 4.3.1).
- All resources for a XJDF node are embedded in the node. There is no *ResourcePool* anymore. Similar resources are compiled in a so-called *ResourceSet*. There might be more than one *ResourceSet* in an XJDF node. Since all resources are inside the XJDF node, there is no longer the need to define *ResourceLinks*, and more searching for a resource in the node tree using *ResourceLinks* (see 4.3.2).
- Change orders can be implemented by simply updating a XJDF node and sending it once again.
- *Spawning* and *merging* has been removed (see 4.3.6). In JDF, *Process Nodes* and *Process Group Nodes* are subnodes of a *Product Intent Node*. Moreover, a JDF file is only valid for one product (and its sub-products). In XJDF, the product description is not directly connected with the production details anymore. Thus, changing the product structure (adding another product part, for instance) does not affect the

production description. In addition, different products can be written to a *Product List* if they are all to be processed in the same way by a single device.

Unlike a JDF node, a XJDF node is not differentiated according to the attribute *Type*. It does not even have this attribute at all. There is, for instance, no difference between a *Process* and a *Process Group*. A XJDF node is, in fact, something like a JDF Gray Box, that is, it can be incomplete. Consequently, XJDF has the *Types* attribute, whose values can hold either one or more process names or the word *Product* (or both). Unlike a JDF Gray Box, however, an XJDF node does not need to be expanded with additional information before it may be submitted to a device for execution. If an incomplete XJDF node is submitted, the device is expected to fill in the missing information on its own with default values. In fact, almost all XJDF attributes are optional.

```
<?xml.....?>
<XJDF Types="Process Name(s)" ...>
       ...
       <AuditPool>...</AuditPool>
       <ProductList>...</ProductList>
       <ResourceSet>...</ResourceSet>
             ...
       <ResourceSet>...</ResourceSet>
 </XJDF>
```

Figure 4.20: Structure of XJDF data

Figure 4.20 shows the structure of XJDF data. As mentioned above, it contains only a single XJDF node. Since JDF and XJDF are structurally different, XJDF is not backward-compatible with JDF. However, an XJDF element can be automatically converted to a JDF element and vice versa. Of course, XJDF data cannot be converted into a JDF job ticket containing the complete workflow logic.

XJDF coexist with JDF. The parallel development of both versions by CIP4 offers options to industry stakeholders wishing to incorporate JDF, XJDF or both into their product offerings and production environments. However, several print automation experts believe that XJDF is the format of the future and will gradually replace JDF.

Figure 4.21 shows a code snippet from an XJDF node that describes the product only from the print buyer's point of view, not the production. The value of the Types attribute is therefore Product. The code could, for example, come from a web-to-print system (B2C) or from any other external site (a subsidiary, for instance)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XJDF Category="Web2Print" JobID="4711" Types="Product">
  <ProductList>
    <Product Amount="300">
      <Intent Name="MediaIntent">
        <MediaIntent MediaQuality="Lumisilk" MediaType="Paper"/>
      </Intent>
      <Intent Name="LayoutIntent">
        <LayoutIntent FinishedDimensions="240.0 155.0 0.0"
         Pages="1" Sides="OneSided"/>
      </Intent>
      ... more Intents
    </Product>
  </ProductList>
  <ResourceSet Name="RunList">
    <Resource>
      <RunList>
        <FileSpec URL="asset/Cool.pdf"/>
      </RunList>
    </Resource>
  </ResourceSet>
    ... more ResourceSets
</XJDF>
```

Figure 4.21: XJDF Node of Types="Product"

and be addressed to the MIS. In the following section 4.5 we will learn that such XJDF codes can also be embedded into a PrintTalk element.

The XJDF node includes a product list, but it contains only a single product in our example. 300 copies are requested. Intent sub-elements describe the technical details of the product, similar to the Intent Resources in JDF. Only *MediaIntent* and *LayoutIntent* are listed here. *MediaIntent* defines the substrate while *LayoutIntent* defines the size (*FinishedDimension*) of the final product, the number of pages (*Pages*), and the number of printed surfaces (*Sides*). The size is given in DTP points, where the first value specifies the width and the second value the height. A *ResourceSet* describes a set of one or more Resource elements of the same kind (see also Figure 4.25). In Figure 4.21 there is only one Resource *RunList* that specifies the storage location of the content file.

Definition

A **RunList** defines one or more printable logical documents, e.g., a PDF file.

The XJDF node in Figure 4.22 is a protocol that travels from a controller to a folding machine (*Types="Folding"*). This example is a flyer to be produced 4/4 in a run of 1,000. In *ResourceSet FoldingParams*, only the desired fold catalog number is specified. This catalog was already presented in 4.3.2.

If you want to learn more about XJDF, I would like to recommend

(Meissner 2017). I would also like to mention the *EasyXJDF* tool written by the same author (Meissner 2018), which lets you generate simple XJDF examples that could be used for W2P.

```
<XJDF JobID="Job1234" Types="Folding"...>
  <AuditPool>
    ...Audits...
  </AuditPool>
  <ProductList>
    <Product Amount="1000" DescriptiveName="Flyer">
      <Intent Name="ColorIntent">
          <ColorIntent>
              <SurfaceColor Surface="Front"
                ColorsUsed="Cyan Magenta Yellow Black"/>
              <SurfaceColor Surface="Back"
                ColorsUsed="Cyan Magenta Yellow Black"/>
           </ColorIntent>
       ... More Intents
    </Product>
  </ProductList>
  <ResourceSet Name="FoldingParams" Usage="Input">
    <Resource>
      <FoldingParams FoldCatalog="F2-1"/>
    </Resource>
  </ResourceSet>
  ... More ResourceSets
 </XJDF>
```

Figure 4.22: XJDF Node of Types="Folding"

4.5 PrintTalk

PrintTalk is an XML-based, open data format used to describe commercial/business activities in the graphic industry. It is used mainly as an interface between the print buyer and the print provider. As an example, let's look at the interface between an external W2P system and a print provider's MIS/ERP system. The second area of application is currently the interaction of print brokers, subcontractors and branch offices with a print provider. We discussed this in Section 2.2.

Note

Initially, 16 companies have announced a project named **PrintTalk**. Next , NPES (now *Association of Print Technologies* or *APTech*) organized and published the PrintTalk specification. In 2005, the PrintTalk development was transferred to CIP4 for its long-term maintenance and distribution.

The root element in a PrintTalk document has the name *PrintTalk*. There are two important sub-elements, the *Header* and the *Request* (see Figure 4.23). The *Header* identifies the original sender and the recipient of the PrintTalk transaction. The *Request* is just a container for a *Business Object*.

There are 15 different Business Objects altogether (ordered alphabetically):

Cancellation, Confirmation, ContentDelivery, ContentDeliveryResponse, Invoice, OrderStatusRequest, OrderStatusResponse, ProofApprovalRequest, ProofApprovalResponse, PurchaseOrder, Quotation, Refusal, RFQ, StockLevelRequest, StockLevelResponse

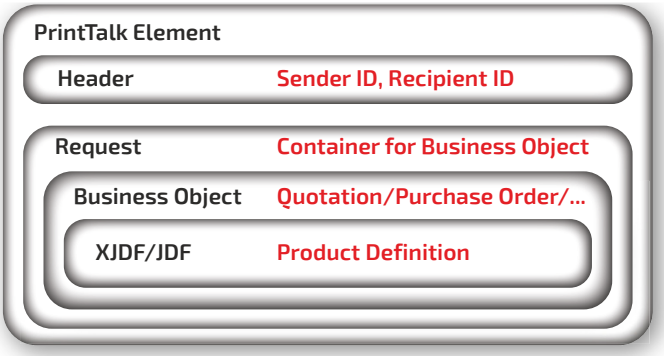

Figure 4.23: Structure of a PrintTalk Element

In Figure 2.3 we have already seen the classic business workflow, which starts on the print buyer's side with an *RFQ* (Request for Quote), followed by a *Quotation* (PP->PB), *Purchase* Order (PB->PP), *Confirmation* (PP->PB), *ContentDelivery* (PB->PP), *ContentDeliveryResponse* (PP->PB), *ProofApprovalRequest* (PP->PB), *ProofApprovalResponse* (PB->PP), and *Invoice* (PP->PB). This workflow is drawn in more detail in Figure 3.3, which also indicates possible refusals of business objects, for example, if the PP rejects an RFQ or the PB rejects a quotation.

In (PrintTalk 2020c) you can find PrintTalk workflows (Figures 6.1 to 6.8). Two of them are shown in Figure 4.24. In workflow ❶, the print buyer queries the print provider about status of his order using *OrderStatusRequest.* This business object can be defined either for one single or for multiple status requests. It can also specify if there should be a unique *OrderStatusResponse* or if the request is supposed to be interpreted as a subscription. In this case, an *OrderStatusResponse* message should be sent whenever a new milestone is reached.

Workflow ❷ represents a convenient workflow for B2B. If the PP maintains a warehouse for a PB, the PB can use the *StockLevelRequest* business object to query whether and how many products are still available in the warehouse. The PP should return a *StockLevelResponse* containing the number of items that are currently available. He

Note

PrintTalk is a protocol on top of HTTP. HTTP and protocols further down in the protocol stack (like TCP/IP) are responsible for the actual network routing of the PrintTalk element. The identification in the PrintTalk header is ensured by means of the Web URLs and the DUNS numbers of the companies involved. DUNS stands for Data Universal Numbering System. It is a unique nine-digit identifier for businesses.

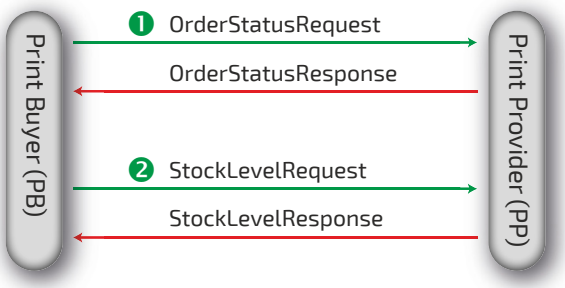

Figure 4.24: Two different PrintTalk workflows between PB and PP

can also return the price of the item. Moreover, he can communicate if new copies are planned for production in case no goods are currently available in the warehouse.

The very special feature of PrintTalk is that XJDF or JDF can be embedded (see Figure 4.23). This is what makes this protocol so powerful. The embedding of JDF and XJDF is described in the (CIP4 2015a) and (CIP4 2020c) specifications, respectively.

Let us take a closer look at the PrintTalk element example in Figure 4.25. For this purpose, the key terms are color-coded (element names in red, attribute names in green, and attribute values in blue). The PrintTalk element consists, as usual, of a *Header* and a *Request*. The *Request* contains the *PurchaseOrder* business object, which in turn encloses an XJDF object. This XJDF object consists of a *ProductList* and two *ResourceSets*. In the *ProductList*, the purchased product is described. The *ProductList*, in fact, contains three products. The first, with the product type *Booklet*, represents the final product. It includes two *Intents*, one for the *BindingIntent*, the other for the *LayoutIntent*. The *BindingIntent* defines, in particular, the *ChildRefs* attribute. The associated values are the IDs of the following two semi-products, representing the cover and the content of the booklet. Both include the *MediaIntent* and *LayoutIntent*, which determine the paper (*MediaQuality*) and the number of pages (*Pages*), respectively.

One *ResourceSet* defines the customer's *Contact* details, the other one specifies the PDF files containing the artwork. The latter information is located in two *RunList* resources. The first PDF file named *Cover.pdf* has two pages (*NPage*), the second named *Content.pdf* has eight pages.

The example in Figure 4.25 is a somewhat simplified version of the original file. For example, the prefix *ptk* for all PrintTalk element names is present, while I omitted the prefixes for the XJDF elements to make the example a bit easier to read. Moreover, I removed a few attributes, such as *Amount* in the Product elements, which specifies the desired number of copies.

Many PrintTalk Elements do not contain a *ProductList*. The business object *ContentDelivery*, for example, only needs to enclose a *RunLIst ResourceSet*, specifying the file path and name of the delivered content file. The delivery must relate to some purchase order, though. For this purpose, the attribute *BusinessRefID* is provided in the element *Request.*

On the next page - Figure 4.25: Example of an XJDF element

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ptk:PrintTalk Version="2.1" Timestamp="2019-09-09T16:27:38Z"
     xmlns:ptk=http://www.printtalk.org/schema_2_0
     xmlns="http://www.CIP4.org/JDFSchema_2_0">
  <ptk:Header>   Information about Sender and Recipient
  </ptk: Header>
  <ptk:Request BusinessID="ID4711">
    <ptk:PurchaseOrder Expires="2021-05-12T19:26:41Z">>
      <XJDF JobID="ID4712" Types="Product">
       <ProductList>
         <Product ID="ID_broschur" IsRoot="true" ProductType="Booklet">
           <Intent Name="BindingIntent">
             <BindingIntent BindingOrder="Collecting" BindingSide="Left"
              BindingType="SaddleStitch" ChildRefs ="ID_Cover ID_Content"/>
           </Intent>
           <Intent Name="LayoutIntent">
              <LayoutIntent NamedDimensions ="A4"/>
           </Intent>
         </Product>
         <Product ID="ID_Cover" IsRoot="false" ProductType="Cover">
           <Intent Name="MediaIntent">
             <MediaIntent MediaQuality="P1" MediaType="Paper"/>
           </Intent>
           <Intent Name="LayoutIntent">
             <LayoutIntent Pages="4"/>
           </Intent>
         </Product>
         <Product ID="ID_Content" IsRoot="false" ProductType="Content">
         <Intent Name="MediaIntent">
           <MediaIntent MediaQuality="P2" MediaType="Paper"/>
         </Intent>
         <Intent Name="LayoutIntent">
           <LayoutIntent Pages="8"/>
         </Intent>
         </Product>
       </ProductList>
       <ResourceSet Name="Contact" ProcessUsage="Input">
         <Resource ID="Contact_123">
           Contact ContactType="Customer">
              <Person FamilyName="Duck" FirstName="Donald"/>
           </Contact>
         </Resource>
       </ResourceSet>
       <ResourceSet Name="RunList" ProcessUsage="Input">
         <Resource>
           <Part Run="R_Cover"/>
           <RunList NPage="2" Pages="0 1">
             <FileSpec URL="File:///dir/Cover.pdf"/>/>
         </RunList>
         </Resource>
           <Resource>
             <Part Run="R_Content"/>
             <RunList NPage="8" Pages="0 7">
               <FileSpec URL="file:///dir/Content.pdf"/>/>
           </RunList>
         </Resource>
       </ResourceSet>
     </XJDF>
   </ptk:PurchaseOrder>
  </ptk:Request>
</ptk:PrintTalk>
```

4.6 Interoperability Conformance Specification (ICS)

Without a doubt, a JDF/XJDF-compatible system cannot be connected via plug-and-play. Manufacturers test interfaces in advance. An integration matrix, which can be found on the CIP4 website (cip4.org), provides an overview of products that have been tested and integrated with each other.

The reasons for incompatibilities are of different nature. First, let us look at the fact that JDF and XJDF element names, attribute names and attribute values can be privately extended. This is to allow a manufacturer to exchange private information between its system components based on JDF/XJDF. Such extensions are ignored by systems from third-party manufacturers. Of course, companies are urged to not replace standard JDF/XJDF keywords with their own, but at best to only supplement them. If this is not properly observed, it can lead to incompatibilities.

The main reason for incompatibilities is probably a different one, however. In general, languages are complex systems of communication. This is true not only for a language between humans, but also for computer languages and even data formats such as JDF. You can express an issue differently, even if you follow all the official rules, the grammar, and the official vocabulary. RIPs might render a PDF file differently. Similar is true for JDF. Some people say that different JDF dialects have emerged. This terminology, however, is a bit problematic, because a dialect is a modification of the grammar and/or vocabulary of a language. However, two JDF systems may not understand each other, even if they both comply with the JDF specifications. The reason for this is that the developers of a system do not implement the entire JDF functionality, but consciously or unconsciously expect certain prerequisites. A JDF reading device may simply ignore certain parts of the JDF information provided. For example, it is conceivable that a folding machine reads the resource *FoldingParams* properly, but then only interprets the information about the *FoldCatalog*, not the individual folding operations. After all, both attributes are marked as optional in the JDF specification.

Of course, there are also new versions of JDF/XJDF published from time to time. Certain new features are introduced, and old ones are discontinued. This alone might lead to incompatibilities.

The *Interoperability Conformance Specification (ICS)* is intended to remedy this situation. These papers describe which parts of the specification should be observed for certain production sectors. There are different papers for different sectors and interfaces. For example, there is an ICS paper for MIS-to-Prepress, another one for MIS-to-Finishing, and so on.

The ICS papers also define names of Gray Boxes. For example, you

will look in vain for information about Gray Box *PlateMaking* in the JDF specification. Instead, there is a description in *MIS to Prepress ICS Version 1.5* (CIP4 2015b).

The Gray Box name is noted in the *Category* attribute of the JDF node. The ICS papers define which process names must be entered in the *Types* attribute of the Gray Box (for example by the MIS) and which must be read (for example, by the prepress system). These two indications may be different, because sometimes a process may be written optionally but must be read if it is present. Moreover, certain conditions can be defined, for example, by saying that from two possible processes only one should be listed. The ICS papers also define for Gray Boxes which input and output resources must be written and read. Here again, conditions can be set. For example, the attribute *FoldCatalog* must be written in the *FoldingParams* resource if folding positions (*Fold*s) are not specified.

It is part of the nature of a Gray Box that not all required process resources are defined. For instance, a *RunList* holding images (bitmaps) is a necessary input for the *ImageSetting* process. Furthermore, *ImageSetting* is a necessary process of the *PlateMaking* Gray Box. However, there is no statement in the MIS-To-Prepress ICS about such a *RunList* resource in the *PlateMaking* Gray Box. This resource should be generated later during the expansion of the Gray Box.

Many ICS papers do not describe a single conformance specification but rather up to three. They are called *levels*, where Level 3 includes Level 2, and analogously Level 2 includes Level 1. These three levels represent different compatibility requirements. In simple terms, the levels identify low, medium, and higher compatibility conditions. For example, according to *MIS ICS Version 1.5* (CIP4 2015c), an MIS only needs to read the *ResourceAudit* resource element in an *AuditPool* if it is compatible with Level 2, and not if it is compatible with Level 1. The *ResourceAudit* element describes the usage of resources during execution (see Section 4.3.4).

Both JDF and XJDF optionally provide the *ICSVersion* attribute to state which ICS specification they comply with. The attribute should be entered in the root JDF Node but may also be entered in others. The entry *ICSVersions="Base_L2-1.5 MIS_L2-1.5"* in Figure 4.26 states, for example, that the JDF node meets the conformance requirements of both Level 2 of the *Base ICS Version 1.5* (CIP4 2015d) and Level 2 of the *MIS ICS Version 1.5* (CIP4 2015c).

```
<JDF xmlns="http://www.CIP4.org/JDFSchema_1_1"
Version="1.7" Type="Product" Status="InProgress"
JobID="07-0111" ID="4711" ICSVersions=
"Base_L2-1.5 MIS_L2-1.5">
```

Figure 4.26: Example of the opening tag of a JDF root element

If two systems have compatibility problems, it might be a good idea to check if they comply with the same ICS levels.

4.7 Portable Document Format (PDF)

Since the earliest days of PDF, the format could contain a *Document Information Dictionary*. This dictionary can only record a few values such as the document's title, author's name, and creation/modification dates. A *Tagged* PDF, which was introduced 2001, contains additional metadata information for the document structure. This means that PDF is no longer just an amorphous page description language, but that the content can be structured in headings, paragraphs, and the like. This logical structure of the page content allows, for example, automatic reflow of text, PDF conversion to XML/HTML documents, and defining the reading order for text-to-speech. In 2004, Adobe published the *Extended Metadata Platform (XMP)* specification (see Section 4.1). It is mainly used to store information about page components (such as images) or about the PDF document itself (such as the author's contact details). Because XMP metadata is much more flexible than the *Document Information Dictionary*, the latter was discontinued with the PDF 2.0 specification (ISO, 2017).

Both PDF tags and XMP metadata are rarely used for specifying the product or controlling the production processes. The XMP structure is not suitable for defining properties of product parts. We covered this in Section 4.1. Tagged PDF structures the layout of the content, but not in accordance with the requirements of manufacturing processes.

This changed in 2010 with the introduction of PDF/VT. This format was designed for document exchange concerning variable data and transactional (VT) printing. The idea was to reduce the amount of processing time for printing variable data. This is achieved by creating another branch for document parts (*DPart* tree) in parallel to the PDF page tree. This allows document parts to be identified within a PDF file by setting page starts and page ends referring to the classic page tree of the PDF file (see Figure 4.27). Each *DPart* object can also be associated with its own *Dictionary Document Part Metadata (DPM)*, which in turn can hold application-specific information about the document parts. The standard defines only the DPM structure that can be populated with basically anything; it does not define specific entries to be used within that structure. For a VT application, it may hold the address of every recipient, for instance. The PDF/VT file must comply with PDF/X-4 or PDF/X-5 specifications.

Note

PDF/X stands for PDF/Exchange. A PDF/X file is a PDF file with a subset of PDF features. It is geared to the printing industry.

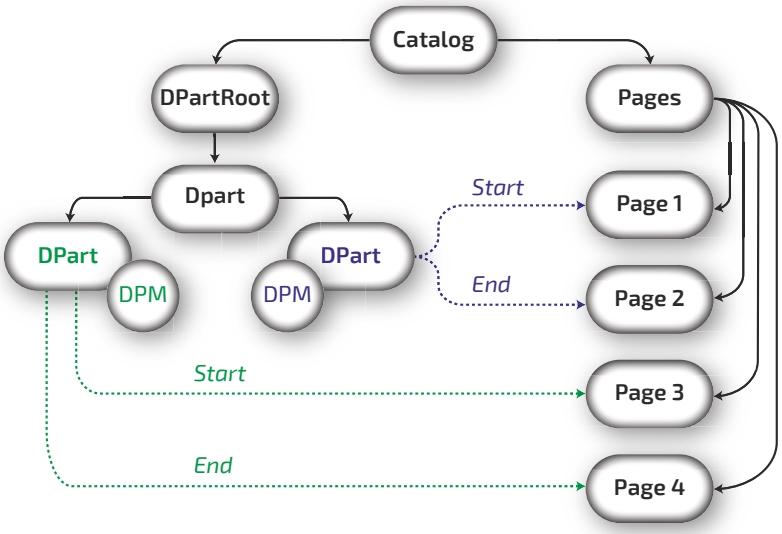

Figure 4.27:
The DPart tree specifies sub-documents (parts) of a PDF file..

4.7.1 Print Product Metadata

The product intent of a print buyer is normally described in JDF and XJDF. That is, both formats can hold properties of the requested product, such as the product type, printing substrate, binding intent, type of inks to be used, number and dimensions of pages, and delivery details. In JDF, the *Intent Resources* define the details of products to be produced, In XJDF, the *Product Intents* take over this role.

Since 2019, there has been third way to store this kind of data – inside PDF. The specification for this is in (ISO 2019b). It is based on (CIP4 2010). See also (Hoffmann-Walbeck, 2018).

Thus, a PDF file contains not only the artwork data but also the product description. This concept goes far beyond considering PDF as just a page description language (PDL).

The idea is that the print buyer stores the product intent inside the PDF file, either by entering the data directly in some layout program, or indirectly by using, for instance, a web-to-print systems that writes this metadata automatically into the PDF file. When such a PDF file reaches the print provider, it can be split up into two parts: PDF artwork data and JDF/XJDF product intent data. As a result, the print shop does not need to change its workflow automation. On the other hand, the advantage of such an approach is that both artwork data and product intent data are stored in one single file during transmission between print buyer and print provider. There is no need any more to link the artwork and the job ticket at the print provider's site. Figure 4.28 illustrates this scenario.

The difference between the classical and the new concept in Figure

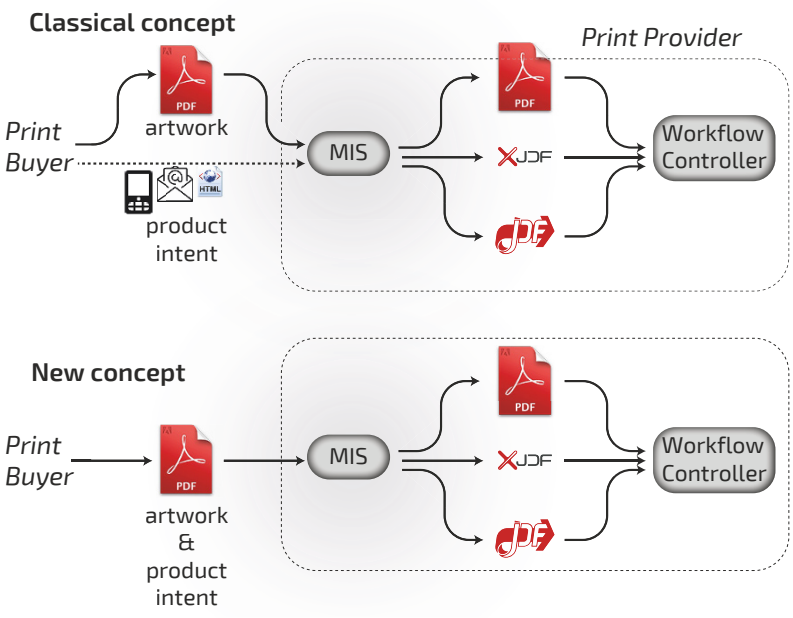

Figure 4.28: The Product Intent is transported via PDF

4.28 seems very subtle, but the new concept might boost the print provider's workflow automation. With the current standard production approach, the PP must link the job data with the artwork data. With the new concept, this is no longer necessary. Of course, even now it is not always a human operator in the print shop who connects the job data and the content data manually, although this is still common practice. By using a portal or a W2P system, this can also be automated. After all, it is still common for the PB to send the job data via e-mail and the content data via a file transfer service (see Section 2.2.1). In this context, the new concept might become useful.

Please note that Figure 4.28 does not show all possible conversions of the PDF file to a job ticket format at the PP's site. In particular, the data format CSV should be added that some prepress systems can process.

How is the product description data stored inside a PDF? It uses the *DPart* structure as shown in Figure 4.27 for PDF/VT. This mechanism allows to logically split a PDF file into several chunks, for instance into pages for the cover and others for the content. The product intents can then be stored in the associated *Document Part Metadata (DPM)* using standard keywords. A *DPM* is not coded in XML such as JDF and XJDF, but rather in a PDF dictionary. Figure 4.29 shows a snippet of *DPM* code. A dictionary in PDF is a table containing one or several key/value pair(s) – like a normal dictionary for languages. Each dictionary is wrapped by double angle brackets, that is <<key value key value…>>. Unlike a normal language dictionary, the value of a PDF dictionary can be an (almost) arbitrary object, for example another dictionary. That is why the structure in Figure 4.29 looks a bit wild. Each dictionary represents a level in the hierarchy for DPM structure. After all, the keywords are easy to recognize: they all start with the letters */CIP4_*.

Finally, I would like to emphasize that the job submission approach presented here is a possible concept for the future, but probably has not yet been implemented on a large scale so far.

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

Figure 4.29: Code snippet of a DCM structure

4.7.2 PDF Graphic Objects as Processing Steps

It is common to use graphical objects in PDF as information for production. Examples can be found especially in the packaging sector, such as die cutting, gluing, creasing, braille, etc. PDF paths are not used for drawing but rather for controlling some production processes. Normally, the data producer creates separate layers and/or special colors for this purpose. This works fine, but there is a catch: the names of the layers and colors are not standardized. Thus, the data creators must always send a separate note to the data consumer in order to explain the PDF. Furthermore, an automatic preflight program cannot check the PDF according to these special requirements. For example, if a color defines a structural design, it must be a spot color, and the object must be set on *overprint*. However, it can easily happen during PDF generation that a spot color is converted to CMYK or that layers are merged. Sometimes people simply forget to set the graphical object to the overprint mode. These kinds of data are then no longer useful.

The ISO 21812-1 (ISO 2019) standard intends to improve this situation. It defines how a graphic can be assigned to a process step. For this purpose, a so-called *Optional Content Group* (OCG) is created which contains a *GTS_Metadata Dictionary* (see Figure 4.30). The process step is defined in this *GTS_Metadata Dictionary*. In the example shown, this is *Cutting*. In addition, there are 22 other values defined in the ISO standard, such as creasing, drilling, gluing, foil stamping, embossing, folding, etc. It is up to the PDF processor to use this information appropriately.

Let us conclude this section with a brief summary of the PDF metadata that we discussed here:

```
7 0 obj
  <<
    /Name (Die cut)
    /GTS_Metadata <<
      /GTS_ProcStepsGroup /Structural
      /GTS_ProcStepsType /Cutting
    >>
  >>
endobj
```

Figure 4.30: The PDF snippet shows an OCG object with a GTS_Metadata key in the OCG dictionary

- *Document Information Dictionary* for storing a small set of fixed keywords.
- *XMP* defines metadata on the page and page element level.
- *Tags* define an HTML or XML kind of structure for PDF, such as header, section, paragraph, figure, etc.
- *Object Content Groups Metadata* for packaging and labeling.
- *PDF/VT* makes it possible to split PDF documents for different recipients.
- *DPart* Product Intent allows product descriptions to be defined in PDF. The print buyer can store suitable metadata in PDF, which is then used to convert it into a job ticket at the print provider's site.

References

Adobe Systems Incorporated (2012), XMP Specification Part 1, Data Model, Serialization, and Core Properties. Available at: https://wwwimages2.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMP%20SDK%20Release%20cc-2016-08/XMPSpecificationPart1.pdf (Accessed: 15 June 2021).

Adobe Systems Incorporated (2016), XMP Specification Part 2, Additional Properties. Available at: https://www.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMPSDKReleasecc-2020/XMPSpecificationPart2.pdf (Accessed: 15 June 2021).

Adobe Systems Incorporated (2020), XMP Specification Part 3, Storage in Files. https://www.adobe.com/content/dam/acom/en/devnet/xmp/pdfs/XMPSDKReleasecc-2020/XMPSpecificationPart3.pdf (Accessed: 15 June 2021).

CIP4 2010: ICS — Common Metadata for Document Production Workflows. Available at: https://confluence.cip4.org/display/PUB/Common+Metadata+for+Document+Production+Workflow+ICS (Accessed: 15 June 2021).

CIP4 2015a: PrintTalk Specification 1.5 (2015). Available at: https://confluence.cip4.org/display/PUB/PrintTalk (Accessed: 15 June 2021).

CIP4 2015b: MIS to Prepress ICS Version 1.5 (2015). Available at: https://confluence.cip4.org/display/PUB/MIS+to+PrePress+ICS (Accessed: 15 June 2021).

CIP4 2015c: MIS ICS Version 1.5 (2015). Available at: https://confluence.cip4.org/display/PUB/MIS+ICS (Accessed: 15 June 2021).

CIP4 2015d: Base ICS Version 1.5 (2015). Available at: https://confluence.cip4.org/display/PUB/Base+ICS (Accessed: 15 June 2021).

CIP4 2018: XJDF Specification 2.0 (2018). Available at: https://confluence.cip4.org/display/PUB/XJDF+2.0 (Accessed: 15 June 2021).

CIP4 2020a: JDF Specification 1.7 (2020). Available at: https://confluence.cip4.org/display/PUB/JDF (Accessed: 24 June 2021).

CIP4 2020b: XJDF Specification 2.1 (2020). Available at: https://confluence.cip4.org/display/PUB/XJDF (Accessed: 15 June 2021)

CIP4 2020c: PrintTalk Specification 2.1 (2020). Available at: https://confluence.cip4.org/display/PUB/PrintTalk (Accessed: 15 June 2021).

CIPA (2016), CIPA DC- 008-Translation- 2016: Exchangeable image file format for digital still cameras: Exif Version 2.31, Camera & Imaging Products Association, https://www.cipa.jp/std/documents/e/CIPA_DC-X008-Translation-2016-E.pdf (Accessed: 30 June 2021).

Consignor. Available at: https://www.consignor.com/carriers/ (Accessed: 15 June 2021).

DIN 66001 (1966), Deutsche Institut für Normung e.V., Sinnbilder für Datenfluß- und Programmblaufpläne, Beuth Verlag GmbH. Available at https://standards.globalspec.com/std/639497/DIN%2066001 (Accessed: 15 June 2021).

Hoffmann-Walbeck T (2018): PDF Metadata and Its Conversion to XJDF, GRID 2018 Proceeding, p. 445-453. Available at: https://www.grid.uns.ac.rs/symposium/download/2018/grid_18_p54.pdf (Accessed: 15 June 2021).

ISO 2010: 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). Available at: https://www.iso.org/standard/46428.html (Accessed: 15 June 2021).

ISO 2017: ISO 32000-2, 2017, Document Management - Portable Document Format - Part 2: PDF 2.0. Available at: https://www.iso.org/standard/63534.html (Accessed: 15 June 2021).

ISO 2018: ISO 19593-1:2018, Graphic technology — Use of PDF to associate processing steps and content data — Part 1: Processing steps for packaging and labels. Available at: https://infostore.saiglobal.com/en-us/Standards/ISO-19593-1-2018-1128856_SAIG_ISO_ISO_2618548/ (Accessed: 15 June 2021).

ISO 2019a: ISO 16684-1:2019, Graphic technology — Extensible metadata platform (XMP) — Part 1: Data model, serialization and core properties. Available at: https://webstore.ansi.org/Standards/ISO/ISO166842019 (Accessed: 15 June 2021).

ISO 2019b: ISO 21812-1:2019 Graphic technology — Print product metadata for PDF files — Part 1: Architecture and core requirements for metadata. Available at: https://www.iso.org/standard/74407.html (Accessed: 15 June 2021).

Meissner S., XJDF - Exchange Job Definition Format, ISBN 978-3-00-055604-3 (2017). Published at ricebean.net.

Meissner S., EasyXJDF (2018), available at: https://confluence.cip4.org/display/PUB/EasyXJDF?preview=%2F688397%2F29426299%2FCIP4+EasyXJDF+Bologna-18.03-bin.tar.bz2 (Accessed: 26 June 2021).