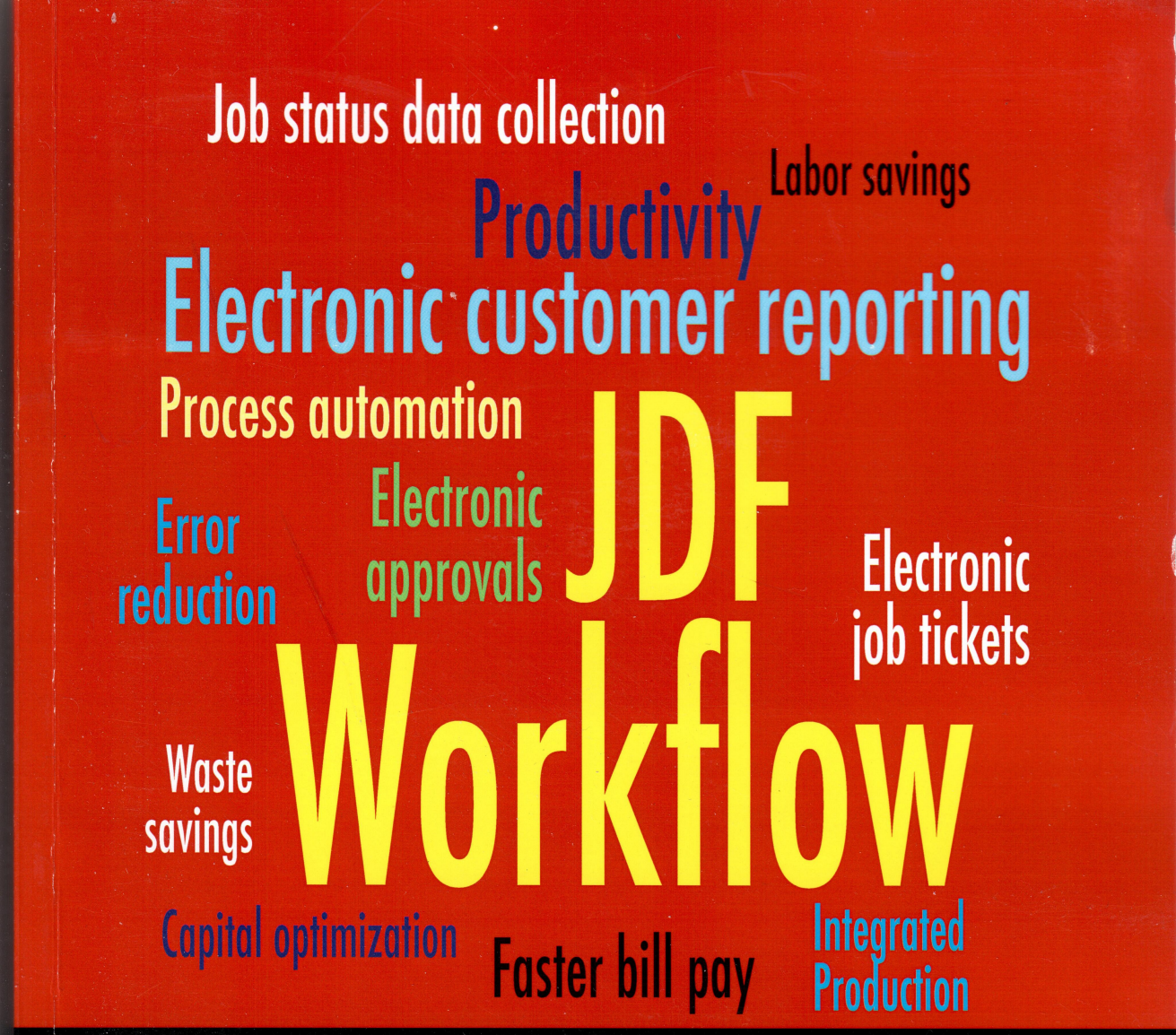


JDF Workflow

A Guide to Automation in the Graphic Communications Industry

Thomas Hoffmann-Walbeck
Sebastian Riegel

Printing Industries Press
PRINTING INDUSTRIES OF AMERICA

JDF Workflow

Thomas Hoffmann-Walbeck
Sebastian Riegel
translated by Joe Dobrowits

JDF Workflow

A Guide to Automation in the Graphic Communications Industry

Printing Industries Press
Pittsburgh

Library of Congress Catalog Card No. 2011932629
International Standard Book Number: 978-0-88362-718-1

Printed in the United States of America
Printing Industries of America Catalog No. 1791
First Printing, XXX 2011

Printing Industries Press
Printing Industries of America
200 Deer Run Road
Sewickley, PA 15143
Phone: 412-741-6860
Toll-Free: 800-910-4283 x770
Fax: 412-741-2311
Email: membercentral@printing.org
Online: www.printing.org

Foreword

Why a book about JDF workflow? Above all, why JDF? Isn't it simply a data format that, behind the scenes, ensures that machines in a print shop receive the correct data? When using the phone, who cares about how the signaling protocols run in the background? And can't you also drive your car without knowing the technical details of the piston profiles?

In fact, these arguments are not to be rejected out of hand. Nevertheless, we believe that the preoccupation with JDF can be interesting for media specialists because Job Definition Format (JDF) describes most of the processes in the print industry and represents an important model for the graphic industry.

The technology is still fairly new and the service does not function in a self-explanatory manner like a telephone or a car. To install, set up test, and debug a JDF workflow requires some basic knowledge of the format.

But we do not intend to go too far into the depths of the data format. More important, in fact, is the workflow itself which can be built with the help of JDF. Therefore we will describe the process in the first several chapters completely independently from JDF, in order to subsequently explain the modeling in JDF. We have tried to separate the two parts. So if you want, you can apply yourself to one of the two things apart from each other.

The intended audience for this book is apprentices in the graphic communications industry, students of printing and media technology, practitioners in print companies or manufacturers, and finally also computer scientists.

In this book we presuppose no knowledge of the JDF format but do expect that the basic production processes for manufacturing a print product are known. However, in the glossary we have defined some of these processes to facilitate the understanding of the reader if necessary.

Naturally, we could not describe all of the processing steps for producing a graphic product in microscopic detail in such a book. There are special textbooks about image processing, computer-to-plate, offset printing, etc. We can only go into technical details in part with examples. On the other hand, we absolutely wish to do this and we are not satisfied to simply philosophize over the general purpose of process networking.

Another warning: we want to illuminate the basic principles of JDF workflows and not give tips and tricks for various workflow products. We will not even allude to the workflow systems of various manufacturers—that is believed to be the responsibility of the trade magazines.

We have taught prepress workflow for many years at the University of the Media in Stuttgart, both theoretically in lectures and in practice as exercises. Here we have determined that there is generally little suitable literature in the area of workflow. On the one hand, there is the more than 1,000-page specification for the JDF format [13], which is primarily aimed at computer scientists and is rather daunting; on the other hand, there are published descriptions of the economic benefits of JDF automation (for example in [32] or [33]).

Of course, there are comprehensive handbooks about the use of their systems from the manufacturers of JDF workflow solutions. Vendor-independent books about "workflow" are quite rare (see, for example, [16]), and most are limited to a portion of production (for example, see [10] and [22]).

We strongly believe that with the JDF workflow a new production form has risen, and is still being developed, in the graphic communications industry—which is fascinating for some and frightening for others. And so in any case we believe that it is worth close examination of this issue. There is still always the quote from the *Klimsch's Year Book* from 1924/25 (page 109).

> Unstoppable and ruthless economic development goes their way. The old is crumbling, and new life bursts out of the ruins. A look back shows the almost sudden development that has taken over the last decade, the graphic art

In the introduction (Chapter 1) we discuss the general characteristics and expectations of a JDF workflow. The development which has led to this format is also briefly discussed. Anyone who has traced professional articles on the subject of JDF for a length of time can safely skip this chapter.

In the second chapter we describe the three scenarios of print production. All three print companies are state of the art, but only two employ JDF technology. We wish to clarify for the readers the differences that may accompany the inclusion of JDF. In addition, concepts such as workflow, job tickets, etc., are defined in Section 4 of this chapter, and in Section 5 general characteristics of workflows are presented.

The process-resource model, or rather the producer-consumer model, is the topic of Chapter 3. Here the basic features of this view are demonstrated with examples from the areas of order management, prepress, printing, and finishing.

In Chapter 5 a short introduction to XML, the Extensible Markup Language, is given, as it is necessary for the understanding of JDF code. Following, in Chapter 6, the most important JDF structures are presented.

Chapter 7 deals with the Job Messaging Format (JMF), the "SMS of the Print Industry." It is a data format and a protocol for communication in a JDF environment. Both form the basis for Chapters 8–12, which cover workflow details and their JDF equivalents from the areas of order management, prepress, printing, finishing, and package printing.

In Chapter 13 two possible JDF projects are discussed which might be of interest to the reader. The first section goes over the implementation of JDF workflows in print shops. The second section gives a short introduction to JAVA programming of JDF applications. It is only intended to provide initial guidance, such as on how to integrate libraries.

At the end of some chapters we have added exercises which should help you to better understand the material.

The authors wish to heartily thank the following people for their valuable support in writing this book:

- Mr. Dieter Adam (MB Bäuerle)
- Mr. Jan Breithold (HELL Gravure Systems)
- Mr. Ruben Cagnie (EskoArtwork)
- Ms. Anja Dannhorn (Fujifilm)
- Mr. Gottfried Grasl (Heidelberger Druckmaschinen)
- Mr. Stefan Kopec (Druckerei Mack)
- Ms. Ulrike Kurz (MBO)
- Mr. Bernd Laubengaier (Druckerei Laubengaier)
- Ms. Prof. Dr. Christa Neß (Hochschule der Medien)
- Mr. Lieven Plettnick (EskoArtwork)
- Ms. Ulrike Seethaler (Heidelberger Druckmaschinen)
- Mr. Matthias Siegel (MB Bäuerle)
- Mr. Klaus Stocklossa (MBO)

We would be very pleased to see suggestions, corrections, comments or additions to this book.

Thomas Hoffmann-Walbeck
Sebastian Riegel
c/o Hochschule der Medien
Nobelstraße 10
D-70569 Stuttgart
0711-89232128 or 0711-89232115
jdf@hdm-stuttgart.de

Editor's Note: Please note that the bracketed numbers found throughout the text refer to the Bibliography listings found on pages xxx–xxx.

Contents

1 Introduction

A "JDF workflow" is generally understood in the graphic communications industry to be an integrated process based on the twin, open-standard data formats of **Job Definition Format** (JDF) and **Job Messaging Format** (JMF). The stated goal is the automation of process steps through the integration of disparate applications and systems. The basic underlying idea is therefore quite simple: to summarize information about a print job and pass it on to the stakeholders who require it. Imagine, for instance, all of the places where sheet size must be entered: estimating, digital prepress, at the press controller, at the guillotine cutter... Obviously, costs can be reduced here.

However, a JDF workflow also has other goals, as we will see, namely, error reduction, time savings, and cost transparency.

1.1 The Development of JDF

Initiated by Heidelberger Druckmaschinen, manroland, Agfa, and Adobe, the data formats were announced shortly before DRUPA 2000 and then presented at DRUPA. In September of the same year, the development of these formats was handed over to the **International Cooperation for the Integration of Processes in Prepress, Press, and Postpress** (CIP4). The association currently has more than three hundred members, mostly suppliers, software users, consultants, and institutes of the graphic arts industries.

The CIP4 Organization emerged from the existing CIP3 consortium, founded in 1995. CIP3 is an abbreviation for **International Cooperation for Integration of Prepress, Press, and Postpress. Print Production Format** (PPF), which was published by CIP3, was also acquired by CIP4. CIP4 is headquartered in Zurich, Switzerland.

PPF and JDF/JMF are interface formats for networking solutions. Both formats serve the principle of the industrialized production of press products, which is in contrast to the former, more handwork-oriented production methods seen in the past. In this context, the commonly dropped buzzword is **computer integrated manufacturing** (CIM).

PPF, which is widely known in the industry as "CIP3 format," makes the transmission of technical data, mostly from prepress to the pressroom, or input to further processing possible (Figure 1.2). The most widely used application is the storage of a preview image (*Preview*) of a signature in a PPF file, a task completed by the RIP

Figure 1.1
History of JDF; CIP4 logo

1993 Concept development of PPF
1995 Public presentation of PPF Version 1.0; Foundation of the CIP3 Organization
1996 PPF Version 2.0
1998 PPF Version 3.0
2000 Public presentation of JDF; Foundation of the CIP4 consortium
2001 JDF Version 1.0
2002 JDF Version 1.1
2004 JDF Version 1.3
2008 JDF Version 1.4

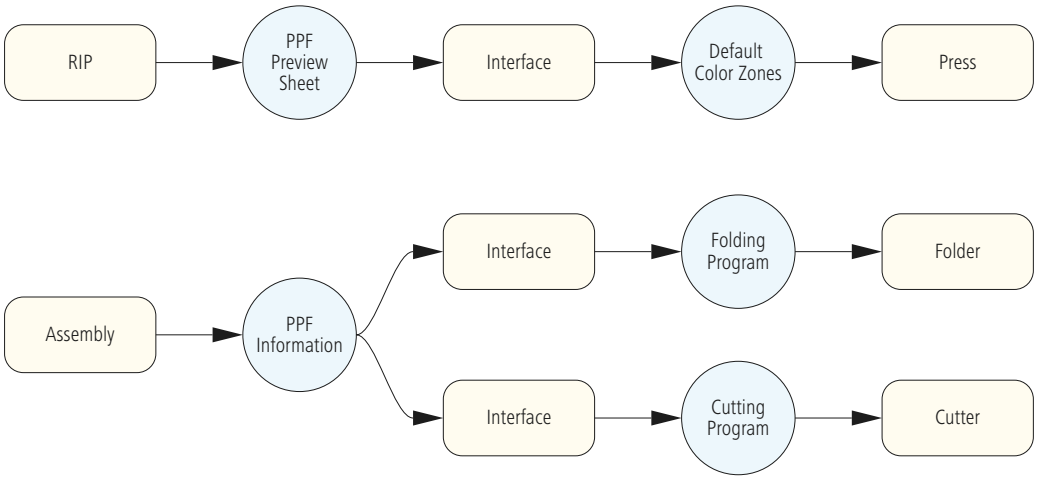
Figure 1.2
Distribution of PPF information

which precedes the plate burner. This preview image is then transferred to software which can calculate the resulting ink zone settings for an offset printing press (Figure 1.3). Another example of the PPF workflow is the passing of information on the cutting and folding of a sheet from the digital layout to finishing. The corresponding marks are set, and subsequently the information is written in a suitable manner in the PPF file. Software capable of interpreting PPF can then generate a cutting or folding program for a guillotine cutter and folding machine, respectively, in a proprietary data format. So in PPF no machine control data is passed on; instead abstract information is passed from which the machine control data can be generated.

In summary, the PPF workflow offers the following advantages over a production mode without PPF integration:

- Technical data, especially for machine settings, may be shared across departmental boundaries.
- Entering data (such as sheet size) in multiple areas can be eliminated.
- Certain process steps could be removed through the handoff of the technical data, for example an earlier standard flatbed scan to produce a color-key setting.

Figure 1.3
Detailed PPF workflow



- Through the publication and standardization of the **Print Production Format,** PPF-compliant modules from diverse manufacturers may communicate with each other.

Further details about Print Production Format may be found in Section 4.2.

1.2 Key Features of Job Definition Format

JDF is sometimes considered an "electronic job folder" (see Figure 1.4), but it is actually much more because it would be a job ticket that, for instance, could control automatic workflows, preset machines, and log the working data.

Most important, JDF offers support:

- In transmission of order data
- In transmission of setup data
- In collection of production and machine data
- For planning
- In order tracking

The JDF specification includes the functionality of PPF, but also goes far beyond. The following additional options are given for JDF/JMF workflows:

- Support of the interface between estimating programs, that is, management information systems and production. JDF also lends itself to defining "soft" information, which may be available at the time the product is specified, for example an approximate line screen. Naturally, at some point in later production processes, these parameters must be clearly defined.
- Machine and operating data that enable **job tracking** as well as the costing of each order.
- A protocol that, to a certain extent, can control jobs and production equipment.
- Support of the **e-commerce** links between print product buyers and print product manufacturers. "Business Objects" can be defined, such as requests, quotes, order confirmations, etc.

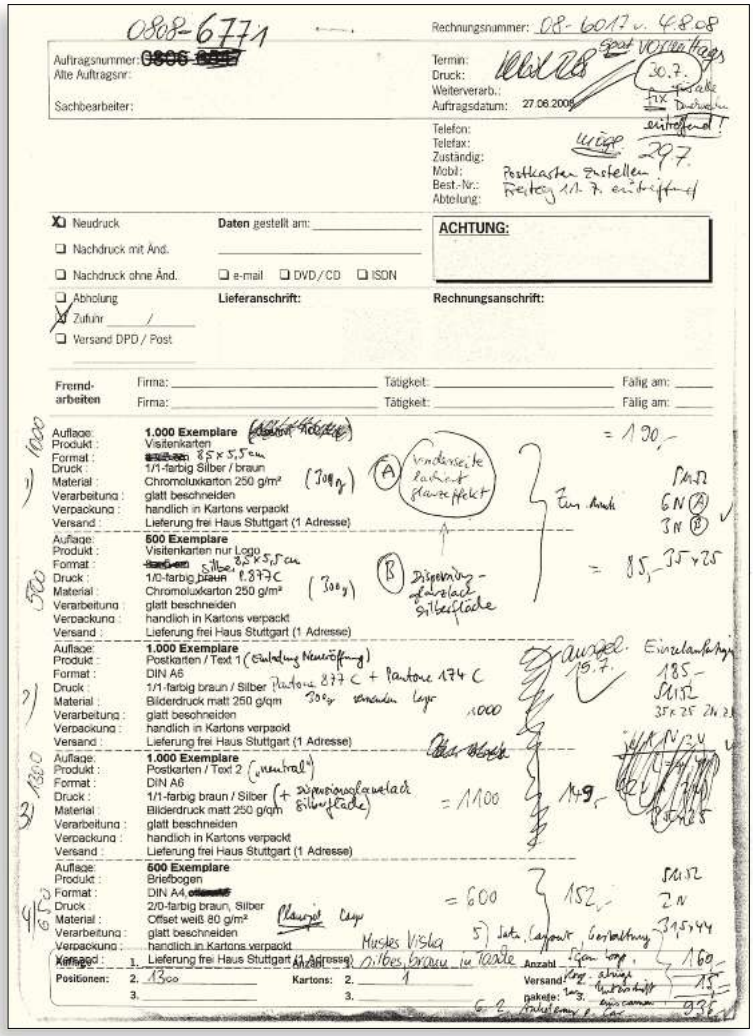


Figure 1.4
Annotated job ticket provided during normal production

- A basic approach for a **plug-and-play** method to integrate new JDF/JMF software into a corresponding production environment.
- The creation of a production log for individual print jobs.
- Methods which allow sections of the JDF data to be extracted and passed on, whether to a service provider, such as plate suppliers, or even different software modules in a print shop. Upon completion of the outsourcing, the altered JDF sections can be re-input into the original JDF. (Figure 1.5)

Figure 1.5
Modification of a JDF document when different service providers are involved with the tasks

The details of these points are discussed at length in later chapters and perhaps are only then fully understandable. Yet it may already be known that the JDF workflow, in comparison with JMF workflow, optimizes order handling (Figure 1.6).

But why is this facet more important? The answer is simple: the reduction in average run length and the increase in average production rate, and likewise the reduction of makeready times for print and finishing machines, allow the company to increase the number of jobs to be processed per hour. Therefore, the costs that are independent of the run length decrease more and more signif-

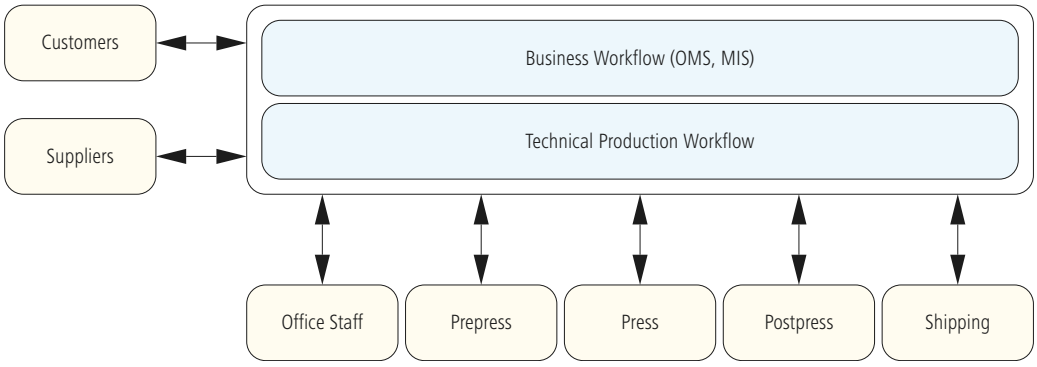


Figure 1.6
Communication channels in the JDF workflow

icantly. And that is now primarily the cost of order handling and prepress work.

1.3 Implementation of JDF Workflows

JDF is only a data format and not a workflow. The specification does not contain any guidance for workflow producers as to how they are to design their modules. Similar to Adobe's PDF specification, there are no rules indicating which functionality a layout program should have in order to export and import a PDF.

Nevertheless, there are some general things which apply to the JDF workflow.

JDF files do not migrate freely from one program to the next or from one machine to the next. That would indeed be theoretically conceivable but probably a nightmare for software developers and users alike. Instead there is a central point with a central database in which the JDF files for the individual jobs are collected. This nerve center may be a dedicated **management information system** (MIS) or even an overarching production system.

JDF/JMF implementations are not consistently present in the print production process, at least not currently. Often a few production machines are connected, especially in finishing, but not to the JDF/JMF network—or information is passed via other protocols, be it PPF or proprietary. The traditional workflow for computer-to-plate (CTP) systems were, in many cases, the seeds of broader JDF workflows. You can often find networking between MIS and the pressroom. However there are limitations on what information will be funneled into the JDF/JMF. One would normally search in vain for JDF-based operations in the graphics department of an advertising agency.

The JDF specification only defines the possible content which may be exchanged between the workflow components. It does not ask which information is available from which components and who must then analyze it. It's a bit like if there was an agreement between air traffic controllers and pilots that data will be exchanged about altitude, airspeed, and rate of descent, but not who is responsible for it, if anyone. And it also expands the JDF workflow that affects the communications between different module classes, such as the interface between MIS and prepress or between MIS and press. For this purpose the CIP4 consortium already offers a whole set of different papers [12] which go under the umbrella

term **Interoperability Conformance Specifications** (ICS). Further details can be found in Section 6.7.

It took several years after the publication of JDF specification 1.0 in 2001 until the first JDF-compatible applications hit the market. Since that time (since 2008) we are now at version 1.4 of the specification, but above all there are many JDF-compliant applications. The CIP4 consortium regularly publishes a catalog of JDF applications and services under the title "The JDF Marketplace" [14]. In the June 2008 catalog, more than one hundred products and services were listed.

Moreover, JDF software may be certified, which means that it will be reviewed based on the ICS requirements. A list of certified products can also be found on the website of the CIP4 Organization (www.cip4.org).

However, neither the mere existence of the ICS nor the certifications suggest that one may then indiscriminately or blindly plug in software modules from different manufacturers without testing. The devil lies in the details. The ICS provides only minimum standards, which for special print are absolutely not sufficient.

This is the reason that the creation or extension of JDF integration into a print shop is always a project; one must always clarify many things in advance with the manufacturers and users and also test. Checkpoints and tips to a practical implementation of such a project maybe be found in [35], and a more general discussion is presented in Chapter 13.

Exercise:

Familiarize yourself with the official website of the CIP4 Association (www.cip4.org). In particular, read the Introduction to the current specification [13].

2 Workflow Basics

In this chapter, two quite contrasting matters will be covered. Initially, the practices of three printers, which we will call X, Y, and Z, are introduced. Company X has a workflow that is completely up to date, but it contains no specific JDF features. Company Y, however, has implemented some components that use JDF. Company Z is largely JDF/JMF networked.

After these rather concrete descriptions it becomes somewhat more abstract. In Section 2.4, basic concepts such as workflow, workflow management systems, and job tickets are defined. Then we cover some general characteristics of workflow, and finally imagine a workflow classification which is based on specific printers.

2.1 Company without a JDF Workflow

The full-service Company X has only twelve employees. For the last several years a modern **workflow management system** (WMS) has been installed in the prepress department with a form proofer and a platesetter as output options. The WMS can also create PPF data for presetting the ink zones, but this functionality is not yet being used. The pressroom works on three web offset presses: a five-, a four-, and a two-color press with formats up to 50 × 70 cm. (19.7 × 27.5 in.). There is also a digital press with inline finishing. The finishing department has guillotine cutters and folding machines, as well as a stamping and embossing book printing cylinder. About fifty jobs are completed each week, and many jobs consist of multiple subproducts.

Requests for quotes mostly come in to the shop through email or telephone, sometimes by fax. A clerk or customer service representative (CSR) enters the most important details of the request into the order management system. The estimate is often taken over by the owner himself, for which he only uses a price table and pocket calculator. In about a third of the requests, information required to do the calculations is missing and the potential client must be called back. The number of estimate revisions that are requested by potential customers has grown dramatically in recent years. In particular, advertising agencies often wish to slightly modify estimates for their jobs.

The ratio of awarded contracts and quote offers is under 10%. As with the awarded contracts, the print job data is mostly sent as email attachments, of which 80% are PDFs; the rest are open data (InDesign, QuarkXPress, Microsoft Office Suite). The CSR then or-

ders the paper, determines the imposition, assigns the job to a press, and prints a run ticket out of the order management system (OMS) (see Figure 1.6). For orders over $700 USD (500 Euros), the CSR also sends a confirmation to the customer.

The prepress department of Company X then examines the supplied files with a preflight program, where about two-thirds (of the supplied files) have errors and must be corrected. Any errors that can be corrected within ten minutes are resolved immediately without contacting the client and without any charges levied. After digital prepress, a form proof is issued which is sent to the customer via mail or by carrier. Only after the proof has been signed off and approved by the customer are the plates output. If the order is a rush and the customer has been known for years, a PDF proof of the press sheet may be created and sent by email to the customer for inspection. If the client explicitly requires a color match proof, it will be created at another company.

The plates then go to the pressroom. A large wall calendar serves as the planning board on which the end dates of the orders are entered. Unfortunately, this does not always work out because the CSR sometimes forgets to copy the due dates to the hanging calendar in production.

During production there are often changes. If the customer calls to make a change to the order, the CSR searches the shop for the run ticket and inserts the changes within. The invoicing and dunning processes (pursuing accounts receivable) are supported by the order management system. There is also an online interface to the tax authority.

A costing of the order occurs only sporadically and then is done manually. The daily run sheets are only occasionally completed in order to verify the internal price table for estimating from time to time.

Order processing for digital printing is identical to the procedures for offset printing.

The owner believes that in his company, highly productive and automated production equipment could be used, but order handling has not reached the same level of automation.

In fact, this appears to be a very typical situation in smaller printing companies.

2.2 Company with a Partial JDF Integration

Print Service Provider Y primarily prints catalogs, magazines, and magazine inserts on its two large web presses and covers in IIIB format for these products on its sheetfed offset press. The firm receives requests for quotes either from existing customers or from sales representatives who acquire new customers. There is a paper-based quote request form for the sales representatives to enter details such as page count, circulation, paper, color, etc. Orders from the inside sales staff are recorded in an order management system (OMS); with new customers, the customer's basic information is also requested. Likewise, the costing is done with the order management system. The quotes are sent to the potential customers, and the sales staff follows up with the lead, if necessary, to renegotiate and issue amended quotes.

At the time of the order placement, information for the Production Planning and Control department (such as required press capacity, input date, and delivery date) is printed out of the order management system. Employees there plan the job and place the appropriate card in the planning table. The planning data is not passed back into the order management system, so scheduling conflicts are not reported automatically in the order management system.

The paper order is triggered by an inside staffer after consultation with the paper purchaser. The order is then automatically sent to the paper supplier.

Of course, the order management system sends an order confirmation to the customer in parallel to this.

The sheet plan (fold, press sheets, and pagination) are specified in the order management system. Production details that are not necessary for estimating are also entered, such as the specifications for the RIP. The OMS then generates an electronic order ticket for each job as an HTML document so that all of the information is available company-wide. A separate paper document is printed out for the prepress department in which the processes are listed and are signed off on after completion by the staff. Information is transferred using JDF from the prepress workflow system so that many of the values entered into the OMS can be automatically transferred. From there, data is passed to the printing presses, mainly color zone presets in PPF format (see Section 4.2), but also preview images for the color-accurate soft proof which is used for color matching. The press approval is made by the customer either via a classic hardcopy proof or even, especially in less substantial

products, via an Internet portal. There, the customers of the print service providers can make correction requests or grant approvals which flow directly into the prepress workflow system.

The plant data collection occurs via terminals into which the staff's working hours, cost centers, setup times, and the like are entered. The presses provide status information to a networked machine automation system; however, it has no connection back to the OMS. Here, the benchmarks must be transferred manually. The OMS then determines the final costing, meaning the variations between the estimated costs and the actual situation.

This example makes two things clear: first it shows the improvements in production that are achieved by an OMS and the corresponding JDF interfaces. It also shows the typical dilemma of JDF: many interfaces, such as online press approval or between the OMS and production planning, are partially achieved with proprietary data formats or are only manually operated.

2.3 Company with Extensive JDF/JMF Networking

The fully integrated Business Z, with one hundred fifty employees, is a typical full-service printer that not only prints books, brochures, and magazines, but also business documents and business cards. Orders are generally received via email and fax. The estimates are done in a management information system. The MIS has an online link to paper suppliers, and each night the current daily price is automatically updated in the system. Inventory for the paper order is verified against stock in the high-rack warehouse directly over the Internet to the paper supplier. The interface is conducted through a proprietary messaging protocol (XML).

If an estimate becomes an order, an HTML job jacket is available to anyone across the enterprise and is generated directly from the database at each request.

The MIS generates a single JDF file per order, which is passed on to a workflow server in the production area. All of the production data relevant to the instructions are stored there. With the help of the JDF data, not only can electronic production planning be carried out, but the jobs can also be handled in prepress. The latter means that the imposition, layout, and other production data can be transferred from the JDF file and, if necessary, can also be modified.

Customers often upload the print data to the print shop's server via the Internet. The print shop's production system receives the

message, and the data is received and processed. The results of the verification are recorded in the associated JDF file and also sent to the customer.

Four of the six printing presses receive the necessary data, such as sheet size, color zones, and colors, via JDF/JMF. Device settings and information about the order are reciprocally sent back to the workflow server. Two other older offline presses are connected via terminals. The workflow server, for its part, sends a portion of the operating and machine data, received from prepress and the presses (for example, plate consumption or specific job-related milestones like completion of the plates or the end of the press process) back to the order management system.

The postpress finishing machines are not currently directly networked; the shop floor data collection takes place exclusively via data-entry terminals.

After the completion of production, the OMS prints out a shipping order. However, a data connection to the logistics companies was not implemented because shipping is carried out by too many different companies, often by small local shippers which have no system connectivity to take orders.

Even with this company it is clear that, despite the very high degree of connectivity, many areas are still not integrated with JDF/JMF. This includes, for example, finishing, where there are long-standing, vendor-specific workflow solutions. Because of the usual long investment cycles in this area, it takes many years for new technologies to be employed.

2.4 Definitions

A workflow is a "production flow" or a "river of work," and a workflow management system is a system for managing this production flow. And the water, which in the year 1555 drove the river of work, is today replaced by a job ticket (see Figure 2.1). Basically that's it, but we still want the terms to be explained a little more precisely so we are not lost in the vastness of the definition.

We begin with the term **computer supported cooperative work** (CSCW). It is the interdisciplinary field of research that focuses on group work, especially with the communication technologies required for it. The different perspectives on CSCW are presented in detail as an example in [41]. **Groupware** is a CSCW application, in other words a software and hardware solution for CSCW. Groupware products are many and diverse. An email system is an

important example to name, but conferencing, planning, group editing, and information systems are also included in this category.

We understand a **workflow** to be a series of defined work steps, whereby sequences of activities are triggered through events that are controlled and terminated. Initially the workflow must not be computer assisted. Cooking, for example, would be a workflow, and so would printing on a printing press—its defined steps include setup, plate changing, adjustments, sheet pulls, registration and color proofing, press approval specification, pressrun monitoring, etc.

Workflow management includes the tasks of definition (modeling), management, execution, control, and sometimes the simulation of workflows. To stay with the previously discussed examples, the organization of a kitchen or also the scheduling of print jobs with a planning board can be described as workflow management.

Finally, a **workflow management system** (WMS) is groupware to support workflow management. Therefore, a WMS is a computer-based application that will help define, manage, and control sequences of operations.

If this issue is of specific interest to you, refer to [31] and particularly to the website of the Workflow Management Coalition (WfMC) [43], whose definition of workflow and workflow management systems we share here:

> ***Workflow:*** The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

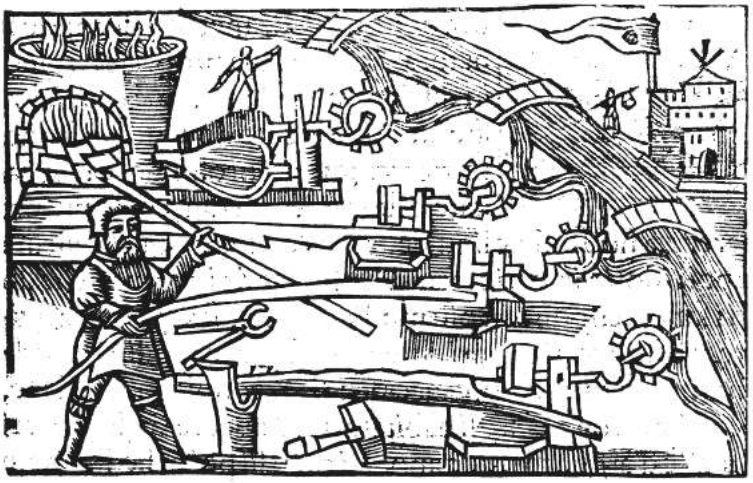


Figure 2.1
Olaus Magnus
Historia de gentibus septentrionalibus (A History of the Northern Peoples), 1555
Forges de Dalécarlie
Bibliothèque Sainte-Geneviève

Workflow Management System: A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

Therefore, the term **JDF workflow** is actually not totally correct; rather one should talk about a "JDF-based workflow management system." But because this is a mouthful, for practicality, the concept is generally referred to as "JDF workflow," and we conclude on that.

Job tickets are defined as electronic information packages supporting workflow management systems in the graphic arts industry. A job ticket, for example, can be a work instruction or a set of parameters to control the print job. In particular, job tickets can naturally be written in Job Definition Format. In other words, JDF is an example of a job ticket format.

A WMS exists out of multiple **workflow engines,** also called **job ticket processors.** These are software modules (like a trapping engine, color transformation modules, or the control station of a printing press) which are able to produce, interpret, and execute job tickets.

In prepress, job tickets provide a form of **metadata,** i.e., data that can contain information about other data. The metadata are in contrast and at the same time linked with the *content* data. The latter are the print data, such as open applications data from InDesign, Quark, etc., or the closed exchange formats like PDF. Metadata can be separated into "object descriptions" and "instructions." By way of example, an **object description** is the declaration that an image has a resolution of 400 ppi. An **instruction,** on the other hand, would be that this image should be scaled down to 300 ppi. Metadata and content data can be present together in a file (see XMP in Section 4.1) or separately, such as JDF. If the metadata lies outside of the content data, the content data must be referenced, for example, a referral to the file name.

Even though metadata and content data are separated, the boundary between the two is blurred frequently. In the mere file name of the content data lays meta information, which is certain to be used by the WMS for workflow control.

2.5 Workflow Classification

Now that the term **workflow management system** has been more precisely defined, we want to divide it into three stages for the graphic arts industry:

- WMS within a department
- Cross-departmental WMS within an organization
- Cross-enterprise WMS with Web-based links ("online portals") to customers and/or suppliers

The cross-departmental WMSs are again, for practical considerations, often divided into:

- WMS with a connection to the MIS
- WMS without a connection to the MIS

Because, in fact, MIS connectivity plays a crucial role. This is known also as **workflow integration.**

Obviously the categories described are, in reality, not always so clearly distinguishable from each other. A WMS generally is used to bind together only a part of all of the activities; that is to say there are activities within a department that are not supported by the WMS, or there are whole departments which are not integrated with a cross-departmental WMS. It is, therefore, always a question of the WMS scope. A RIP, which also provides data for calculating the color zone settings of the press, is already an example of a cross-departmental WMS, as is a printing company which has integrated JDF into several departments.

The three categories will be explained in greater detail.

The traditional workflow is sketched out in Figure 2.2. The customer or, as the case may be, the person placing the order, communicates metadata (green arrow) as well as content data (black arrow) to the job manager of the print shop. Metadata is commonly sent to the print shop by phone or by email; content data is commonly sent over the Internet as an email attachment, or occasionally on a CD or DVD. After the CSR enters the job in an OMS and the job ticket has been printed, the CSR forwards the data to production for validation and data processing. In the "individual components workflow" diagram, the print data are processed by various software components in which "Assembly" and "RIP" stand only as examples. If more people and more computers are

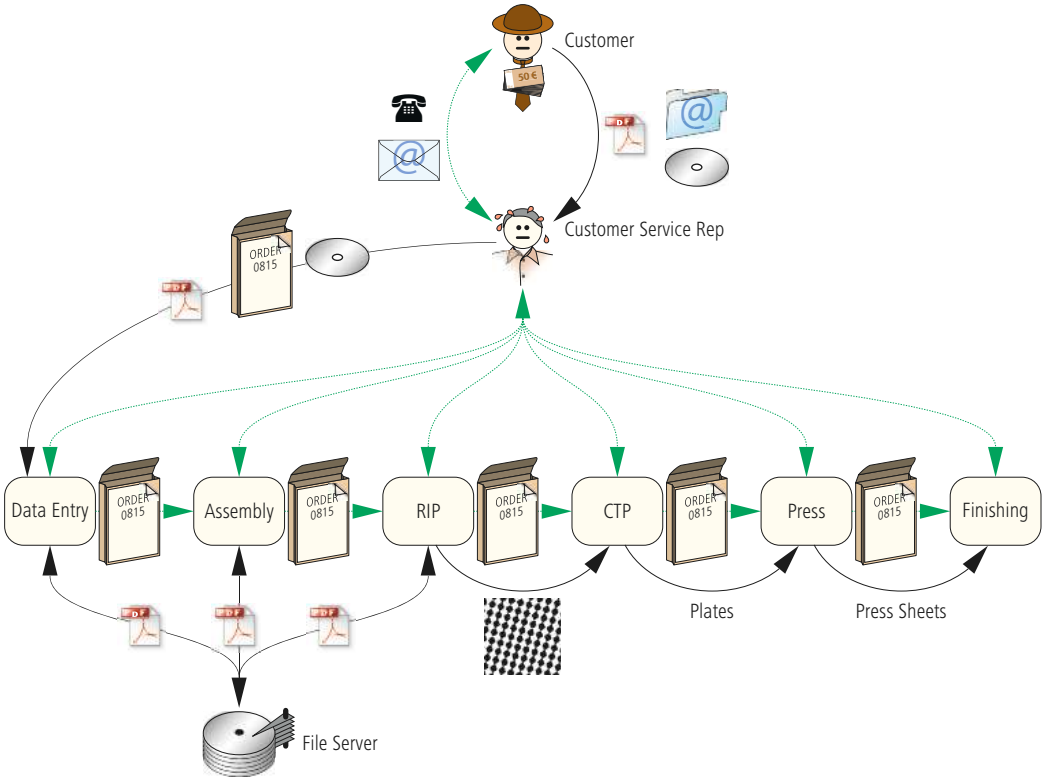


Figure 2.2
Traditional model of a prepress workflow

involved in the production process, the data is cached on a file server and the job ticket will nevertheless be passed manually. Finally, a press plate is output and sent to the pressroom along with the job ticket. For order tracking, the order manager must contact (by phone or in person) the individual stations of the production process.

What are the obvious weaknesses of this workflow? There are a number of things to mention:

- The content data and order data are transferred via different communication channels, which reduces production reliability.
- Data collection either occurs independently of the work process or does not occur at all.
- Order tracking (job tracking) is expensive.
- Preset information from machines must always be entered from scratch.

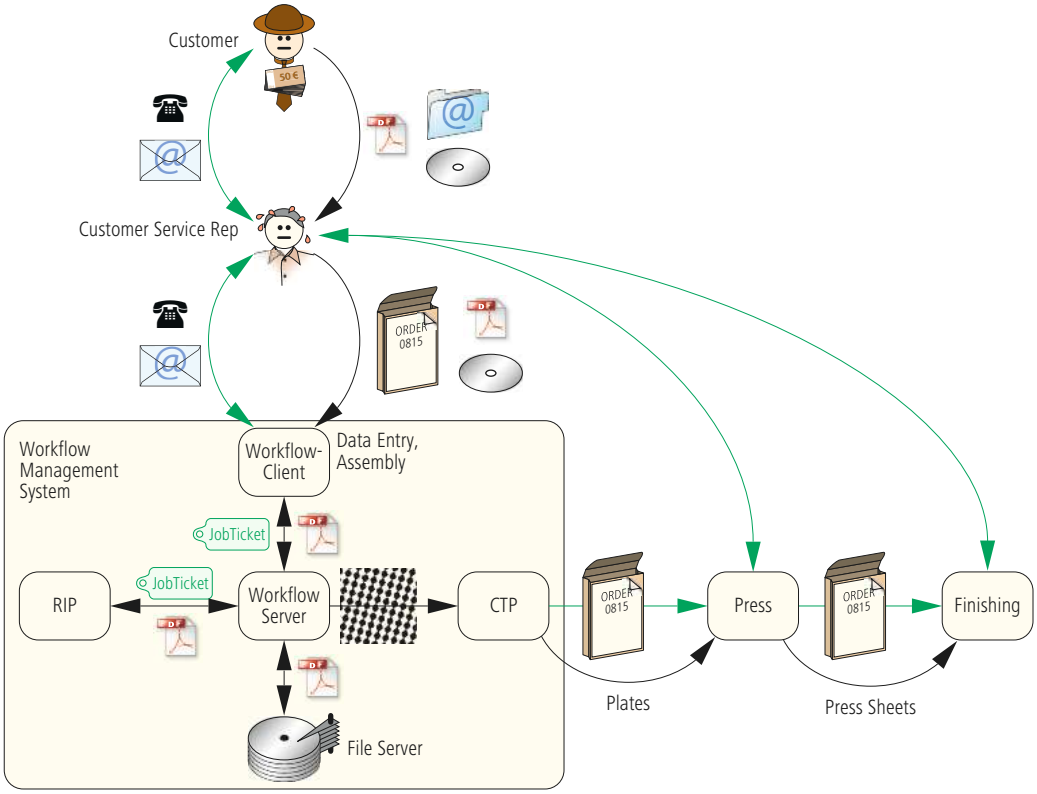


Figure 2.3
Client-Server-based WMS

This way of working, therefore, does not strictly adhere to the definition of a real workflow management system. One sees such a workflow management system in Figure 2.3. Using a client-server workflow system, production can be automated considerably better.

As a client, the user(s) call up the necessary job tickets which have been forwarded to the server-based job ticket processors. Generally, only the system administrator may edit the default job tickets, for example to set a 70 line screen default generated by the raster for output. A typical prepress operator may then only change the individual entry for his press order.

Through these firmly defined job tickets, which can still be adjusted individually, one gains a high level of production reliability and flexibility.

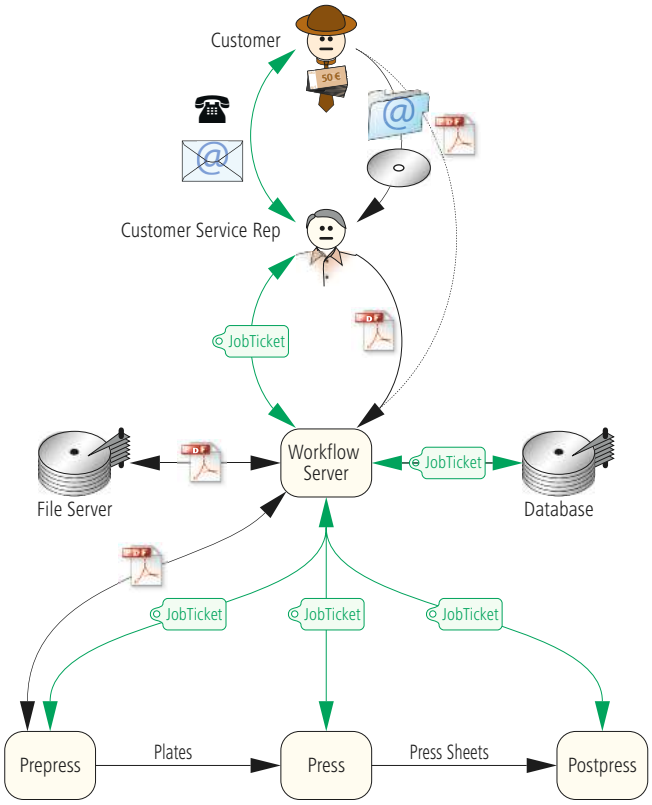
The black arrows which lie in the WMS with the yellow background transport content as well as metadata. Here, the strict separation of the two communications channels is overcome.

The concept represents a significant improvement over the individual-component workflow, but still has some limitations:

- No overall order and production management
- No overall data collection (performance data collection)
- No interdepartmental transfer of presets for machines

With such a WMS the cross-functional electronic transmission of information is already common. This data, for example, which is used as the basis for calculating color zone settings, is pulled up from prepress and passed to the pressroom.

Figure 2.4
Fully integrated WMS



A complete and integrated cross-departmental WMS within a company is outlined in Figure 2.4. This metadata is transported over the local network to the various departments and back again. The content data are managed by the client, of course, only up to the prepress department. It should again be noted that such a plan does not reflect reality, since the network is currently only partially set up in the firm.

In Figure 2.5, final data exchange occurs only through the Internet. Order management, prepress, and plate production may lie in any location in the world and be performed by different companies. Geographic proximity is only necessary when physical things, such as printing plates or press sheets, must be transported. Indeed such a plan seems to be unrealistic or exaggerated, but if you look at recent examples of Internet-based workflows between customer and print shop, it's clear that we are not so far from it.

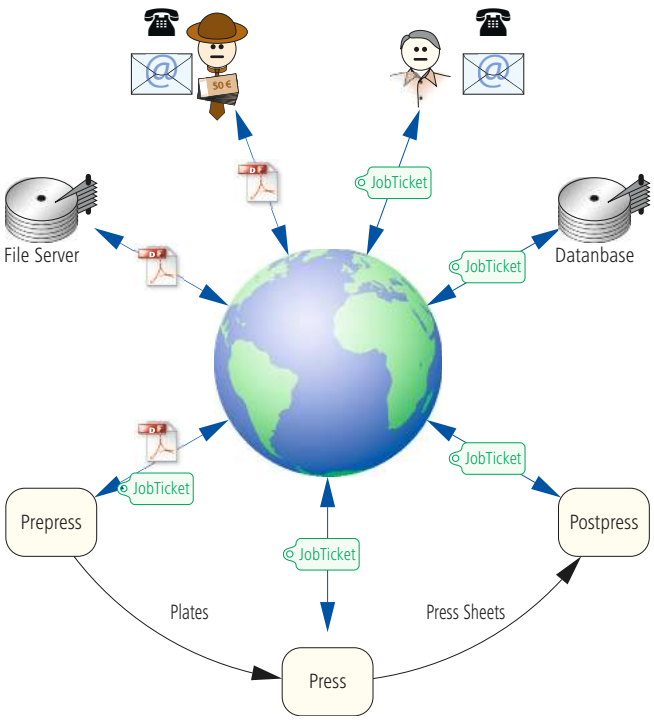


Figure 2.5
Internet-based WMS
(The locations of the participating "service providers" play almost no role)

2.6 Properties of WMS and Job Ticket Formats

We want to provide a more detailed discussion of the following three requirements of workflow management systems, or the underlying job ticket data formats: adaptability, standardization, and extensibility.

These keywords seem to be quite self-explanatory at first glance, but are not without problems on closer inspection.

Adaptability

The WMS model must adapt to the workflow of a business and not the other way around. This seemingly accurate statement has its limits: a totally chaotic workflow can not be modeled. Rather, a WMS with an arbitrary, but well-defined production method should be depicted. In fact, analyzing and defining the production method is an important and difficult task when introducing a JDF workflow in a company.

Business and production procedures generally run chronologically not just serially but also in parallel, or alternating, overlapping, or iterative and repetitive (see Figure 2.6). There are many examples of these situations in the production of printed products:

- **Serial:** First print, then fold…
- **Parallel:** Create text and images for a page layout
- **Overlapping:** Plates (several types), exposing and printing
- **Alternative:** Print the job on machine A or machine B
- **Iterative:** Proof, corrections, proof, approval

A WMS must be able to model these situations. This raises the question of what must be anchored in the job ticket file format and what is anchored in the WMS software the job ticket uses.

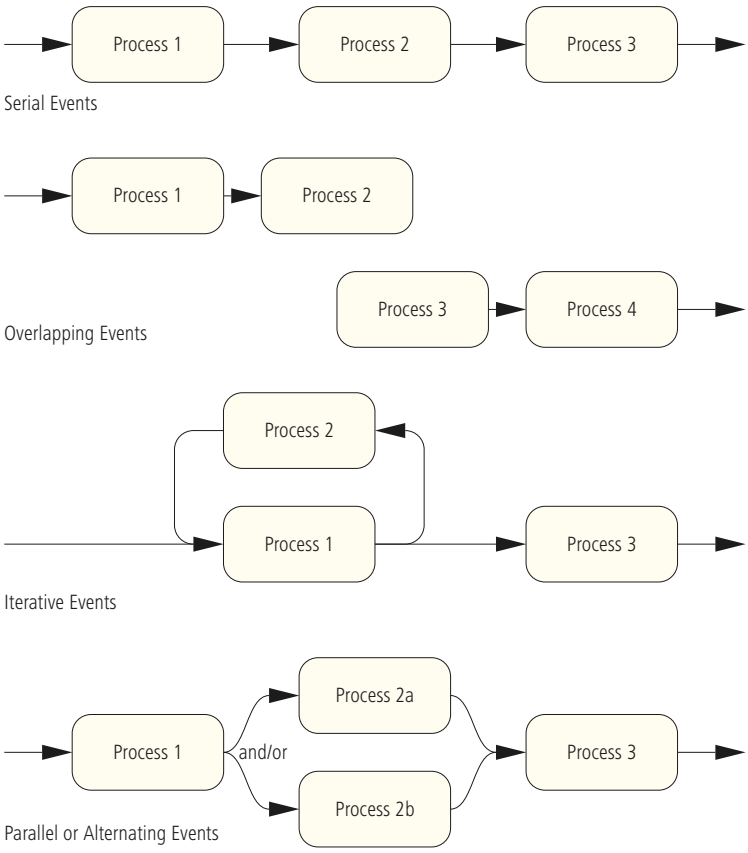


Figure 2.6
Different sequence possibilities of business and production processes

In JDF one can model overlapping procedures, for example. For this purpose, the concept of a "pipe," i.e., a tube, is used. The principle is that an operation produces an output (for example, creating printing plates), and this feeds into the pipe and overlaps a second process (printing), the output virtually obtained from the pipe is used as input for the next step (Figure 2.7). And while printing takes place, the plate burner can store more plates in the pipe. In other words, one could also describe the structure as a warehouse or a silo.

In the manufacture of a printed product, it is also quite typical that, during the course of production, many technical and organizational details are clarified. This means that the job tickets must be dynamic and able to constantly receive new information. The exact color zone presets, for example, are unknown when the order is accepted. Furthermore, the print product may have been specified, but not clearly enough, at the beginning of production. Perhaps at that point, as mentioned before, only the approximate line

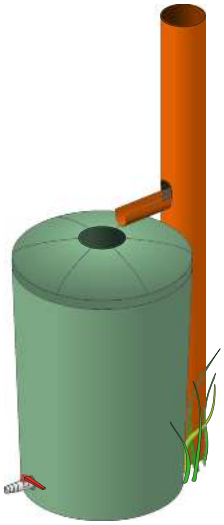


Figure 2.7
Any amateur gardener knows the pipe principle of the rainwater barrel:
1) The output of the rain is the input of the barrel
2) During a rainfall, one can simultaneously water plants as long as there is water in the barrel (pipe)

screen is known, but during the course of production the information is more precisely defined. The WMS and job ticket format must take these changes and updates into account.

Even changes in production parameters during job processing (e.g., triggered by the client) are quite common but, unfortunately, very difficult to model in a WMS. Much development work is necessary here.

Standardization

To allow the possibility of cross-vendor integration of **WMS engines,** it is necessary to standardize the underlying communication. However, standardization has its limits. After all, what would that ultimately mean?

First, a job ticket format must be complete, meaning it must be capable of handling all of the possible communications content for graphics products. If this were not so, some of the WMS vendors would not use the standard job ticket format because of the lack of functionality for their specific products. The alternative is that the WMS vendors must restrict the functionality of what may be defined in the job ticket format; as a result progress would be completely blocked and firms would not be able to differentiate their products. Finally, such a standard must be voluntarily agreed upon between firms and not by national regulation. Accordingly, a restriction is totally unrealistic. Standardization that is not fully fleshed out will be doomed to failure. On the other hand, their complete specification will never be achieved because production methods are much too complex.

So how does one solve this dilemma? The answer is that standardization can only be completed up to a certain point and job tickets will be supplemented with non-standardized information. This actually contradicts the idea of standardization and can also lead to incompatibilities between **workflow engines,** but it is the only remaining way out. We will go into the so-called "extensibility" of job ticket format shortly.

It is not enough to only define the content of the communication, i.e., the "what," within the specification of job ticket format. It also needs to establish the "who." A communication between many workflow engines can only function correctly if it is clear who provides certain information and, conversely, who can rely on receiving the information. Within JDF/JMF we are given

Interoperability Conformance Specifications (ICS), which are discussed in Section 6.7.

Yet, a very pragmatic viewpoint illustrates a limitation in standardization. Workflow management systems are not, or at least only rarely are, programmed from scratch but instead evolve over years. Legacy systems must frequently be considered, and conversions to new job ticket formats only take place little by little. That is why you also find workflow engines in the latest JDF/JMF-based workflow management systems which have communications channels that rely on proprietary (vendor-specific) or older job ticket formats.

Finally, here's one last more or less trivial observation: standards evolve! There are already four versions of the JDF specification, and, therefore, the danger naturally arises that workflow engines will "speak" and "understand" differing versions.

Extensibility

Despite a standard model, the WMS vendor must allow room for the possibility of appending a special extension. This demand for "extensibility" especially goes for the job ticket data format. For all that, what does such a feature for data formats look like?

Job ticket information is arranged according to the rules and structure (the "syntax") of other languages with respect to document structure. In practice these are:

- PostScript (PS)
- Portable Document Format (PDF)
- Extensible Markup Language (XML)

In Chapter 4 the job ticket formats **PPF** and **Portable Job Ticket Format** (PJTF) will be analyzed in more detail. It should be noted at this point that PPF is encoded in PostScript and PJTF is encoded in PDF. Job Definition Format has an XML document structure. The extensibility of XML is explained in Section 5.2

PS and PDF are well known in the graphic arts industry ([4] and [5]) as **page description languages,** or rather as a document structures for page description. The scope of their language may be expanded in both; in other words, private keywords may be freely chosen and optionally registered with Adobe Systems Incorporated (in order to avoid name conflicts). At this point we want to put forward the expansion possibilities of PS with the help of an

example. The process within the object-oriented PDF is exactly the same.

In PS, variable names can be defined and initialized with values. The variable name is a string and is indicated by a leading slash (/). The keyword *def* assigns a value to the variable. For example, the PS code

/hev /Helvetica-Bold def

defines a new variable with the name *hev* which holds the value Helvetica-Bold. In this case the new variable simply serves as shorthand for the longer original name. In exactly the same way, one writes the line

/CIP3PreviewImageWidth 1425 def

The variable name *CIP3PreviewImageWidth* is defined in the PPF specification and, according to the agreement, gives the width of the preview image in pixels. Likewise, you can define your own invented variable with your own values, for example

/MyExtension /TotallyAwesome def

This is valid PS code which, naturally, you can't do anything with. But companies can populate PPF with their own variables and other PS structures so that they may transmit their own supplementary workflow information, which the CIP3 organization didn't allow for in PPF.

These proprietary extensions do not describe "theoretical possibilities"; instead they are employed often in practice in PPF, PJTF, and also JDF! For this reason, in certain circumstances, compatibility between systems is restricted. Therefore, the proprietary entries only supplement and should not replace the standardized information. If workflow engines encounter proprietary extensions and they do not understand them, they are encouraged to ignore them.

In summary, you could say that the three terms *adaptability, standardization,* and *extensibility* contradict each other and that a kind of compromise must be found between these three properties in the specification of job ticket formats, and especially in the implementation of workflow management systems.

3 Print Workflow Models

In this chapter the following four workflow models are described within the context of print workflows, with the help of several examples:

- Activity lists
- State transition diagrams and activity diagrams
- Flowcharts
- Producer-consumer model or process-resource model

Other modeling approaches for common business processes can be found in [18].

Activity lists are also often referred as "to-do" lists and illustrate a collection of subtasks, such as a common shopping list. An expanded activity list can also, perhaps, list the responsibility and the beginning and the end dates of the subtasks. So it is just a question of who does what, when, and with what resources. Not shown in the activity lists are the dependencies between the subtasks or explicit listings of criteria for deciding between possible branches in work processes.

The latter is an important feature of **state transition diagrams.** In a state transition diagram, states are drawn as rectangles or circles and state transitions with arrows between the relevant states. A classic illustration is the states of a machine, such as *stationary, in use, on hold,* and *fault* (Figure 3.1). In the event of a fault, for example, there is a state transition from *stationary* or *in use* to *fault.* In the workflow description one can have the same kind of dia-

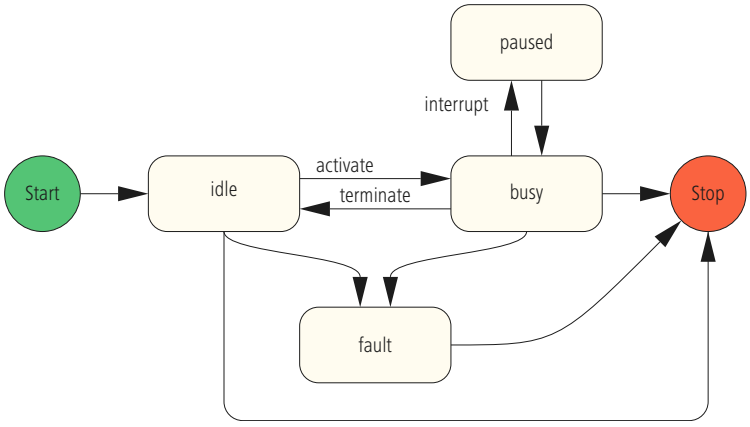


Figure 3.1
Possible states for a machine in production

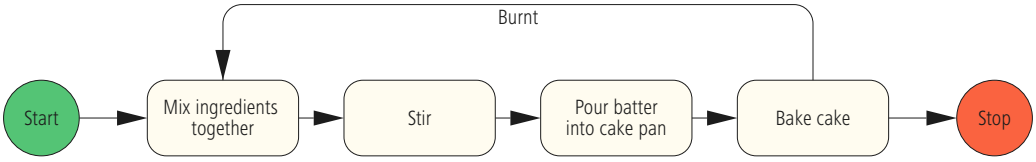
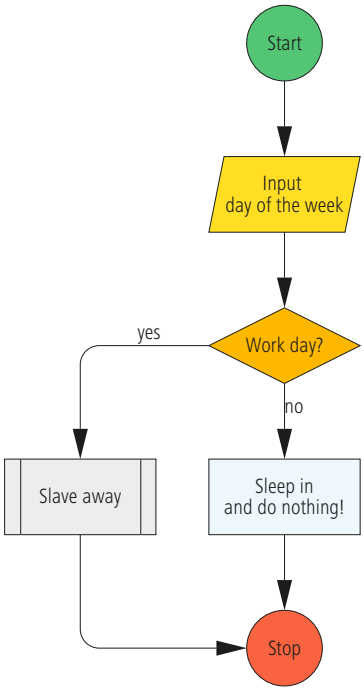


Figure 3.2
Activity chart for baking a cake

grams, but, instead of states, activities are represented here. The arrows are consequently the transitions between the activities, and in this book we will call such diagrams **activity diagrams.** In this way even baking recipes can be represented very clearly (Figure 3.2).

First and foremost, **flowcharts** serve as a visualization of program sequences in software development. One may also find them in general models of business and production process applications, or they may even be very helpful in planning for everyday life as we see in Figure 3.3. The green and red circles mark, respectively, the beginning and the end of the diagram, the yellow parallelogram shows an input or output, and the orange diamond indicates a choice that allows only a *yes* or *no* answer. The blue rectangle represents common operations; the gray rectangle with the double vertical line is a subroutine which is generally more precisely specified in a new flowchart. All of the geometric elements are standardized in their meaning, but not in the colors that we have given them.

Figure 3.3
Flowchart of various operations



The **producer-consumer model,** which we may also call the **process-resource model,** describes exactly what one would expect: a producer processes a product and places it in a "data buffer," such as on a shelf in a supermarket. The consumer takes a piece from the buffer and uses it. Both processes are uncoupled from each other temporally; of course, the producer must first produce something before the consumer may consume it.

In the print workflow, this model is very common, except that one simply speaks of processes and products (i.e., "resources") instead of producers and consumers. The plate-burning process, as an example, produces the burned printing plates, which are consumed by the printing process. In turn, the printing process generates a resource, namely a stack of printed sheets which are used by a finishing process, and so on (Figure 3.4).

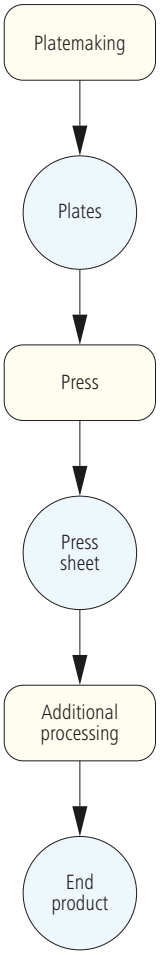


Figure 3.4
Producer-consumer model in the print workflow

Further, cake baking, which we modeled with the state transition diagram in Figure 3.2, can also be represented very nicely in a producer-consumer model. The mixing of ingredients, the stirring, etc., are the processes; the ingredients themselves, such as the flour, eggs, butter, and milk, as well as the results at each stage, such as the batter or the filled baking pan, are the resources. You can follow this rule of thumb: the processes are the verbs and the resources are the nouns.

The JDF model is based on the process-resource principle and therefore will be referenced throughout the entire book.

We will now apply these three models to the areas of order management, prepress, press, and postpress.

3.1 Order Management

The activities of order management, up to awarding the order, can simply be reduced to the following list:

- Request for quotation (customer to the printer)
- Quotation submission (printer to customer)
- Awarding the order (customer to printer)
- Order confirmation (printer to customer)

Clearly there is a temporal relationship between the subtasks and describing these activities with a diagram such as in Figure 3.5 helps to clarify the process.

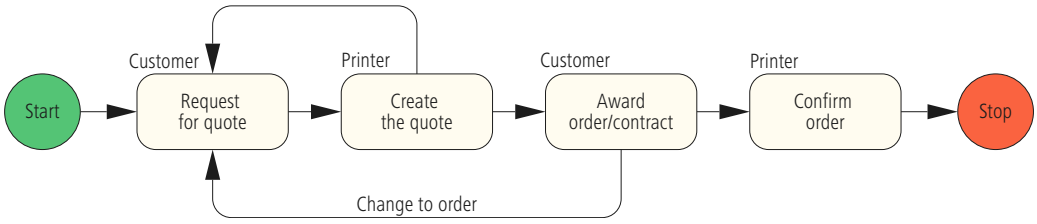


Figure 3.5
Order management activities

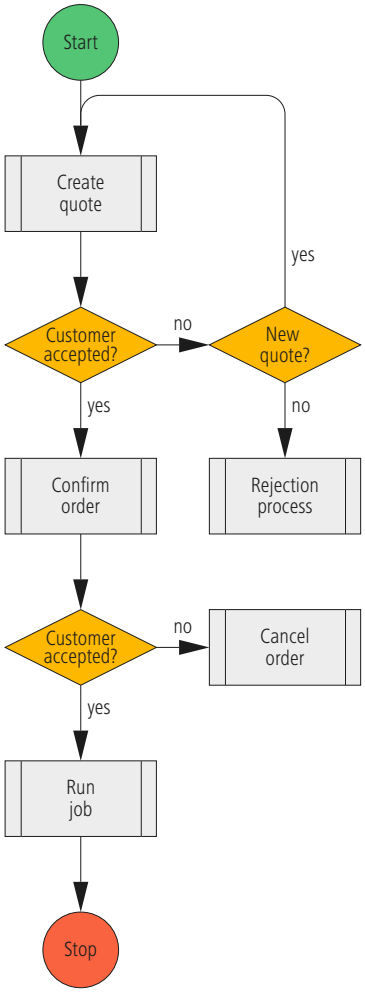


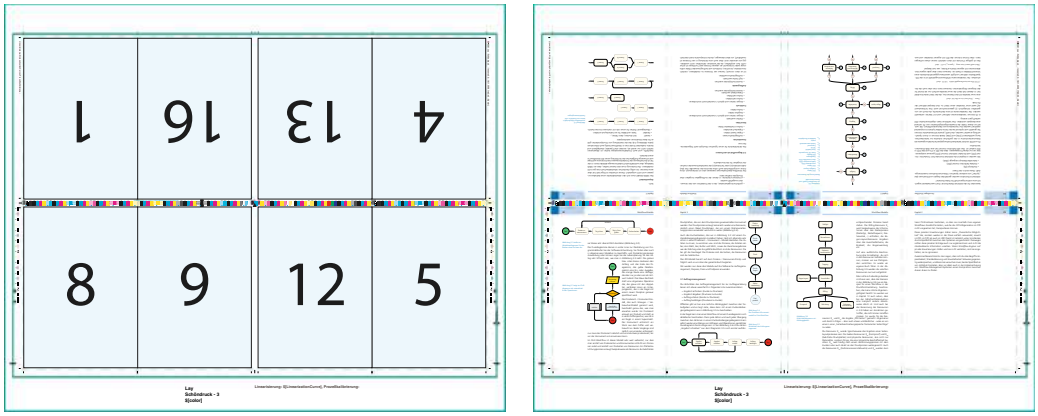
Figure 3.6
Flowchart showing actions of a print shop

As a general rule, a workflow can be illustrated in greater detail with a flowchart because each activity—and also each transition between the activities in an activity diagram—can themselves generate many questions and operations with respect to the flowchart terminology. In Figure 3.6 the print shop's actions in Figure 3.5 are represented once again in detail as a flowchart (see also Figure 3 In [40]).

3.2 Output Workflow in Prepress

The platemaking department of a print shop, on receiving the finished PDF pages, typically has the following activities (for clarification of terms, see the glossary at the end of the book):

- Conduct preflight and, if necessary, data correction
- Normalize the PDF data
- Perform color space transformations
- Trapping
- Create or, as applicable, select one or more imposition layouts (press sheet layout)
- Imposition
- RIP the data to the output of a plotter for the form proof(s)
- Print approval
- RIP the data to produce preview images



Sheet layout

Imposed sheet with content placed

Figure 3.7
Sheetfed layout creation and imposition

- RIP the data to imaging, platesetting, and developing of printing plates
- Calculate of the color zone preset values

Let's take a quick look at the difference between "stripping" and "imposition," because in practice these two concepts are not normally divided accurately. With the creation of a press sheet layout, the first thing that is determined is the size of the press sheet and the starting point of the paper (the position of the press sheet on the press plate). In commercial web printing, the pages or positioning are defined by their sizes and positioned on a virtual press sheet, often with the help of an imposition schema. This must include the type of binding and the type of perfecting with respect to the gripper margin and other margins, such as low folio and high folio, or take into account grind-off for adhesive binding. The page order is defined as a placeholder (template) with no content. The placeholder pages are referred to as "sample pages." Finally marks such as the color bar, cut and fold marks, register marks, and the like are placed on the sheet. The result is an imposition sheet, or a press sheet layout.

At the imposition, the pages or positioning which typically exist in PDF format are assigned manually or automatically and are added to the imposition sheet together with the markings. The sample pages are filled with content, and they contain data which the sheets depict as an example in PDF format (Figure 3.7).

In the past, both processes would be carried out within one program, the imposition or assembly software. Today, this is commonly separated: the generation of the sheet layout is done un-

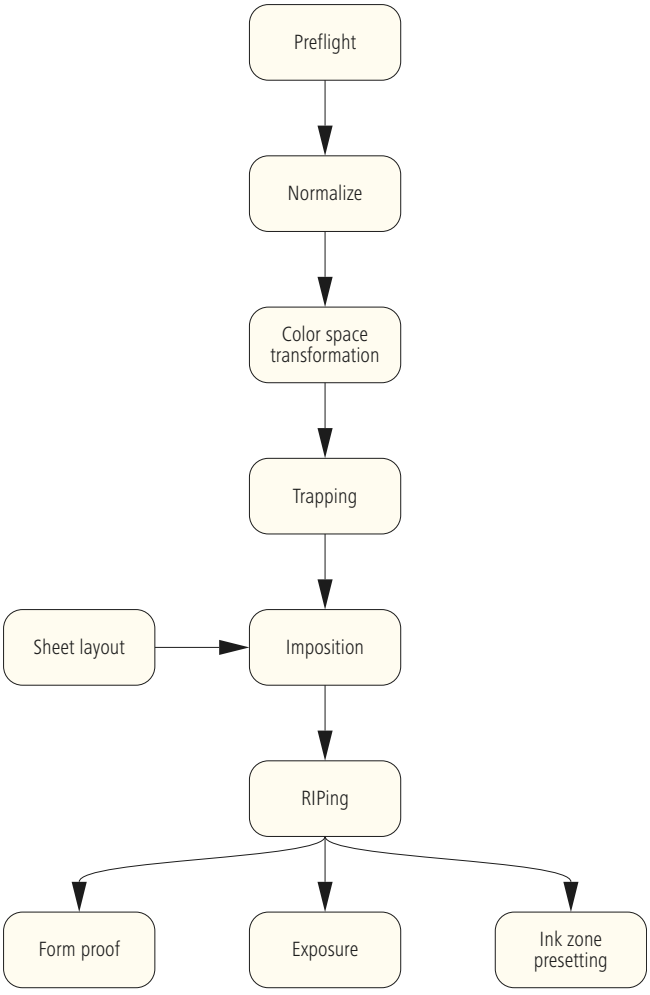


Figure 3.8
Activity diagram with dependencies

der the control of an operator, usually with standalone software on the desktop computer, while the automated process of imposition is performed on a server in the background.

The activity diagram, which also indicates the dependencies, is seen in the example in Figure 3.8.

The corresponding process-resource model is not only much more accurate, but also much more complicated (Figure 3.9). The processes are represented by rounded rectangles; the resources are represented by circles and distinguished as blue (R_1 to R_{10}) and red (R_{11} to R_{24}). The blue resources are pure input resources for the process, while the red resources appear as output resources. The blue resources are basically parameter resources, the detailed information for the preparation of the corresponding processes. The RIPing resource, R_7, by way of example, contains information about the raster process (raster type, line screen, screen angle, etc.); the sheet layout resource R_5 contains details about the imposition schema the binding type, sheet perfecting, etc.

To save space we must sacrifice an extensive description of all of the details which can be found in the resources. The descriptions would fill an entire book. The red resources are briefly listed in Figure 3.9.

It should be clear that the scenario described in Figure 3.9 is only an example of a platemaking workflow, which is not, strictly speaking, universally valid. We will also see in Chapter 12 that a completely different way of working is typical in packaging production. And in the naming of resources in Figure 3.9, we have made assumptions that might not precisely apply. Thus, the description "PDF pages"

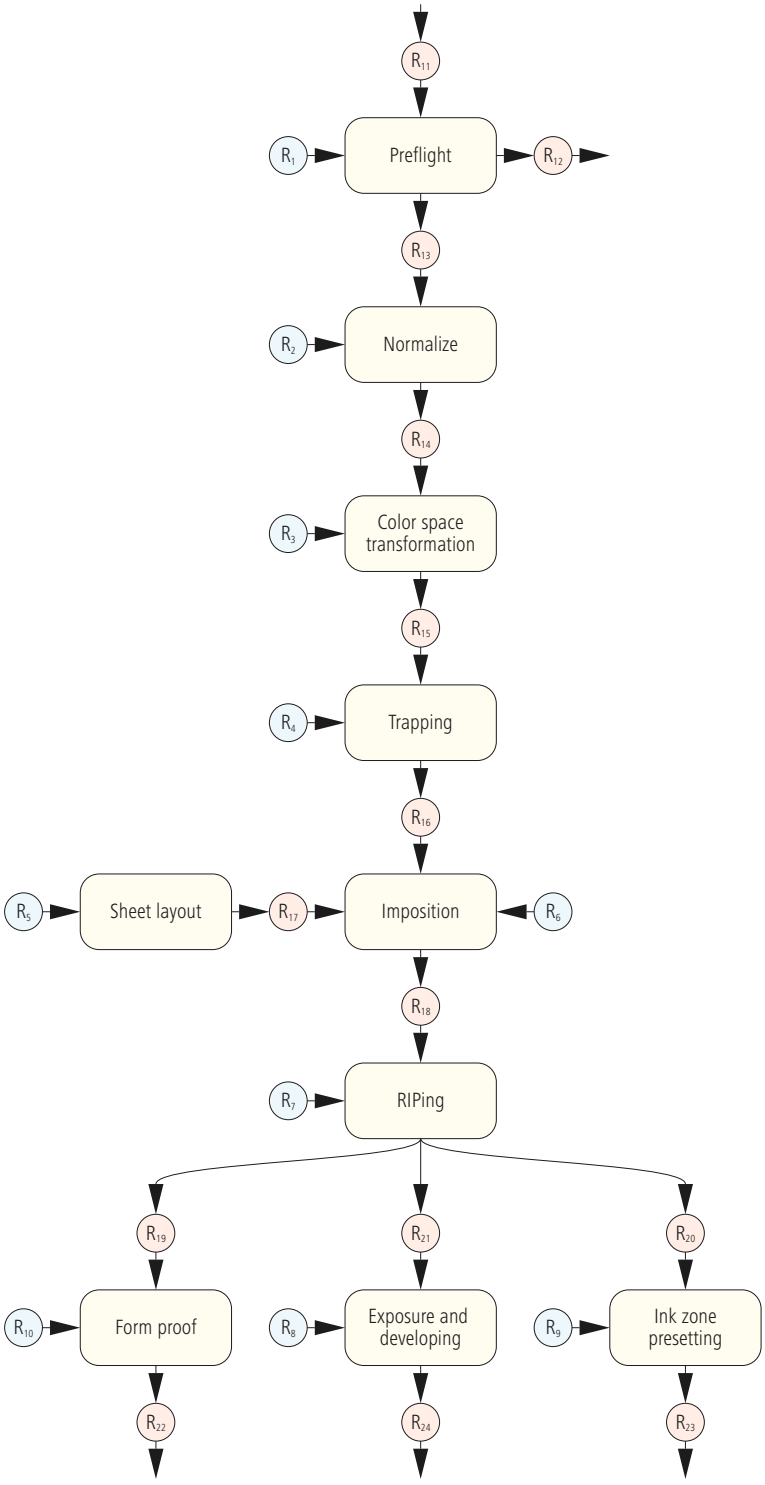


Figure 3.9
A complex process resource model

Description of resources:

R_1 bis R_{10}	Input parameters of the respective processes
R_{11}	PDF pages
R_{12}	Preflight report
R_{13}	PDF pages
R_{14}	Normalized PDFs
R_{15}	Color-converted PDFs
R_{16}	Over-/under-filled PDFs
R_{17}	Register sheet
R_{18}	PDF sheet
R_{19}	File from form proofer
R_{20}	Preview image
R_{21}	TIFF-B
R_{22}	Form proof
R_{23}	Color zone preset values
R_{24}	Imaged printing plate

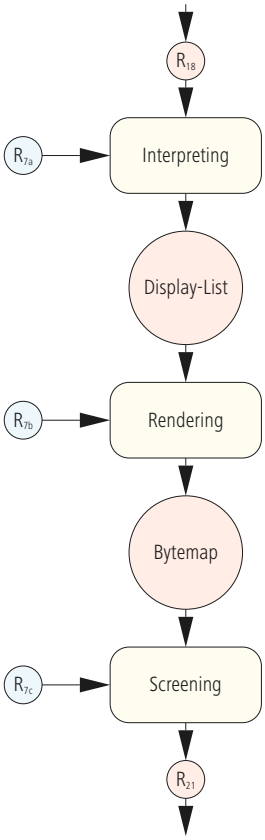


Figure 3.10
RIPing process

was made for resources R_{11} and R_{13}. It would be more generic, and therefore proper, but somewhat cumbersome, to speak of a "page description language formulated page sequence."

The resource R_{11} would typically be the output of a page layout process. Both resources R_{22} (form proof) and R_{24} (imaged printing plates) are physical resources, not just data records but things that have a physical quality. R_{22} often happens after a consultation with the customer or is passed on directly to the printing process. Also the resources R_{23} (color zone presets) and R_{24} will serve the printing process as input resources.

In the model of Figure 3.9, you can also certainly see dependencies in the process-resource model, but it doesn't describe a strict sequence. It does not specify whether to first provide a form proof and then produce a press plate or vice versa. The latter would, of course, be absurd. In Chapter 9 methods are presented which guarantee that proofs are duly made within the production chain (approval process).

Now how finely graded should the process be? You certainly can also define larger units, as shown in Figure 3.4. Naturally, you can also deconstruct the workflow model in Figure 3.9 more finely still. You could split up the process "Exposure and developing" (with a corresponding plate and a corresponding exposure) into: expose plates, punching, preheating, chemical development, rinsing, gumming, and burning. Yes, it would even be conceivable for one to split the exposure process itself yet again into: pulling the plate out of the stack, inserting the imaging drum, fastening, exposing, and removing the exposed plate. And so you could always go into more and more detail; a process could be illustrated up to the end of each impulse on a step motor.

That would certainly be unreasonable. It is true, however, each process in production, possibly triggered by themselves individually or executed by a device or software, is a process to define. No one positions even one plate on an imaging drum if it is not also fixed and exposed; therefore, it would be meaningless to split the exposure process. An exposure process and a developing process can be defined and would thus be useful again. Conversely, the process definition from Figure 3.4 would likewise not be appropriate since more independent equipment and applications are involved in plate production and in finishing.

The JDF model in fact looks very similar to Figure 3.9. However, the RIPing process is, by way of example, even more finely broken down and indeed is outlined like in Figure 3.10. It is easy to rec-

ognize that the process of RIPing was split into three processes: Interpreting, Rendering, and Screening. Accordingly R_7 was divided into three resources: in the R_{7a} *InterpretingParams* resource, in the R_{7b} *RenderingParams* resource, and in the R_{7c} *ScreeningParams* resource.

We will explain the three terms only briefly (see Figure 3.11). The interpretation process analyzes the page description language and the data structures. In the *InterpretingParams,* there could, for example, be an entry about whether the page should be adjusted or scaled with respect to the size of the output medium. The result of the interpretations process is a (nonstandard) data structure, called the *DisplayList.* The rendering process's next main task is to fill in the closed contours which are mathematically defined in the page description language. The result is a pixel structure, called a *Bytemap.* The resolution of this pixel structure is an entry in the *RenderingParams.* The resulting pixel structures have color depth, which means the pixels may be comprised of different color shades. The pixels are converted into a raster (bitmap) graphic within the screening process either in a periodic raster (AM screen) or a non-periodic screen (FM screen). The information important for the screen process (type, frequency, etc.) are entries in the *ScreeningParams.*

More detailed explanations about this process can be found as examples in [22] or in [29]. But why is RIPing divided up into three processes? The reason is simply that the screening process doesn't always need to be carried to completion; perhaps it only needs to be RIPed for the plate artwork or for a halftone proof but not usually for a digital print.

We want to point out again that this concerns the workflow *model,* not the workflow itself. An output resource contains only the description of the exposed plates, not the plates themselves. Nevertheless, this is referred to as a **physical resource.** However, in order to keep it simple, this book does not often differentiate between model and reality. For example, we will continue to simply speak of a "plate resource."

Even the parameter resources must not necessarily be included directly in the workflow model. For example, a PDF file, which specifies the printing elements of pages or sheets, is not embedded in a JDF file. Instead it only states where in the data system the PDF file can be found. Therefore, there is only a reference to the PDF file in the JDF file.

Perhaps you're asking yourself who actually composes the workflow model. Is it true that for each print order an operator must

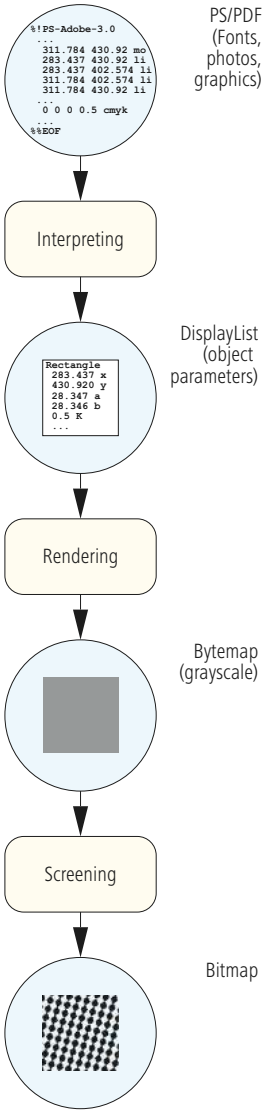


Figure 3.11
Rendering and screening process

prepare a workflow so production can proceed more or less automatically? If so, must he or she actually set all of the resources exactly? And if that is actually the case, is that not a terrible expense?

Well, in actuality, this does not work. But then which sources supply the workflow descriptions for a print job? The answer is that information is automatically extracted in part from the following areas:

- Estimating/order management system
- Print data from the client
- Preset production parameters (base or default values)
- Graphical input forms which are filled out by the users in production
- E-commerce systems, in which the client can provide information about print jobs

The workflow descriptions occur then in the background without help and mostly without the knowledge of the users. The manufacturer's or user's administrator or technician generates the requirements for the base values.

3.3 The Sheetfed Offset Process

The activities list for a typical print job in sheetfed offset looks like this:

Basic Setup

- Read the job jacket
- Prepare the paper
- Set up the paper path
- Fill the inkwells with ink

Setup

- Change plates
- Printing, sheet pulling, and visual/measurement device control
- Adjust registration
- Adjust color

Production Run

- Printing, sheet pulling, and visual/measurement device control
- Color readjustments
- Refill ink
- Changing the paper pile
- Washing the blanket cylinder

Order Completion

- Washing the blanket cylinder
- Color change (if necessary)
- Filling out the job jacket

Does it then make sense to model four processes here, namely *Basic Setup, Setup, Production Run,* and *Order Completion*? Or instead should each point underneath be its own process? The latter is certainly not recommended, since each individual activity is not independent of each other. But a division into four processes is problematic mostly because the points underneath are often identical or at least similar. The activities by *Setup* or by *Production Run* are actually the

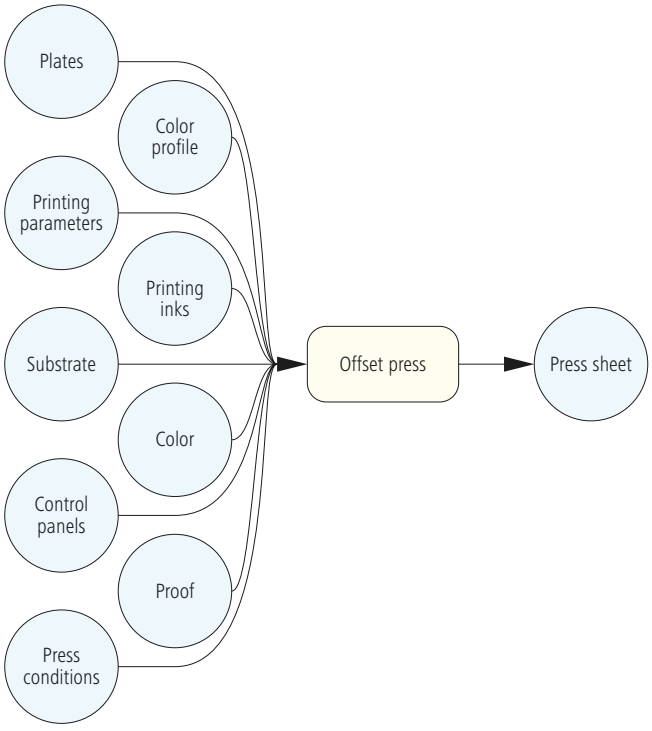
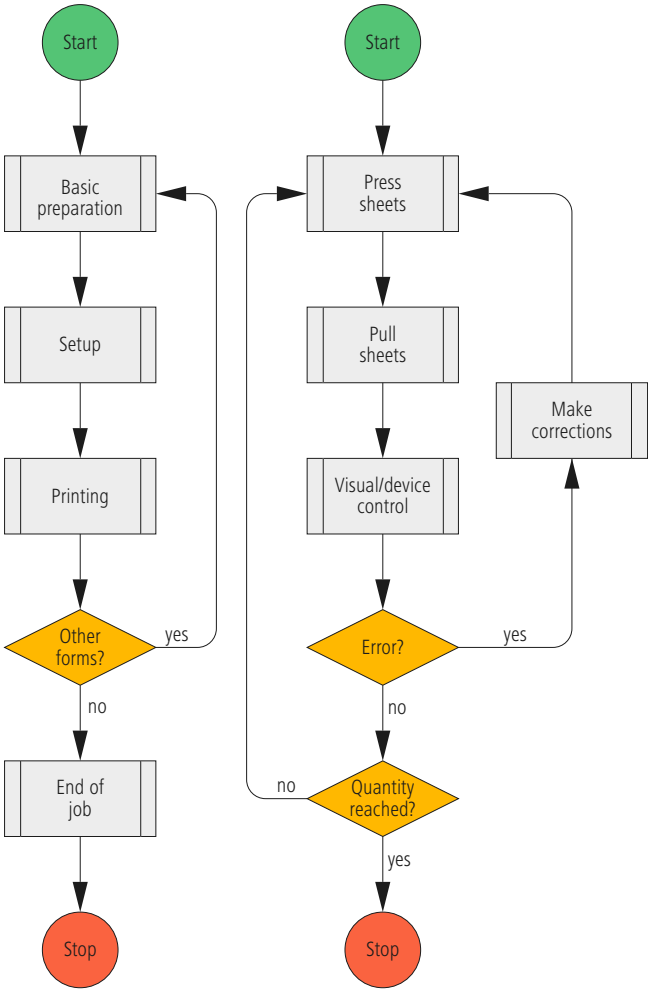


Figure 3.12
JDF model for offset printing

same; only the outcomes are different: one is spoilage, the other good sheets. Note that the schema of the job list is quite rigid. So in practice the work is "dovetailed"; that is, during a production job the printer is already readying the job ticket for the next job. Also, all of the activities are not absolutely mandatory. For example, if the paper is known in a print shop, the point "Set up the paper path," as a general rule, is not necessary.

Figure 3.13
Flow chart describing sheetfed offset



In this respect the CIP4 Organization has chosen a very simple process-resource model for offset printing, as shown in Figure 3.12. In order to distinguish between the individual activities in offset printing, we speak of "states" which the printing process are in.

With digital offset (the printing plates are imaged within the press), and with inline finishing, especially within roll-fed offset, more processes are found to take place within the press—not just printing but also plate burning, cutting, folding, etc.

It can be seen in the diagram in Figure 3.12 that only some input resources must be present; others, such as a proof, are only optional. You will also see that the model essentially says very little about the actual printing process. If you want to illustrate this more precisely, create a flowchart for yourself.

Figure 3.13 (left side) shows the main levels of the list of activities as a flowchart, each of which is shown as a subroutine. The subroutine "Production Run" is shown again on the right side of the diagram. But here we find the subroutines and one could again create new flowcharts for "Sheet pulling" and "Visual and/or mea-

surement device control." So one can go into deeper and deeper levels of detailed description.

3.4 Example Model of Postpress

Print finishing is too complex to develop an all-encompassing model. A universal list of activities would be long and always incomplete, so it does not make sense for us to keep putting one together. Instead we will only look at the process-resource model as an example, namely the saddle-stitched brochure, in Figure 3-14. Resource R_7 constitutes the input to further processing, such as the press sheets as they come out of the press room. Resources R_8 through R_{13} are descriptions of intermediate products which are formed during postpress:

- R_7 Press sheets
- R_8 Signatures
- R_9 Folded sheets
- R_{10} Blocks
- R_{11} Bound blocks
- R_{12} Trimmed blocks
- R_{13} Saddle-stitched products

Resources R_1 to R_6 are parameter resources, which keep the information for the respective processes. We want to name a couple of typical entries for each of these resources:

- R_1 Position of the cutting blocks/cut marks, system
- R_2 Folding sequence, fold position, system
- R_3 Sequence of gathered components
- R_4 Number, position, width, angle, shape, and profile of the wire
- R_5 Width and height of the final product
- R_6 Maximum height and weight, number of layers per stack, number of print products per layer

The size of the press sheet is not usually entered within JDF in R_1, but instead in R_7. Also the paper properties, which are not partic-

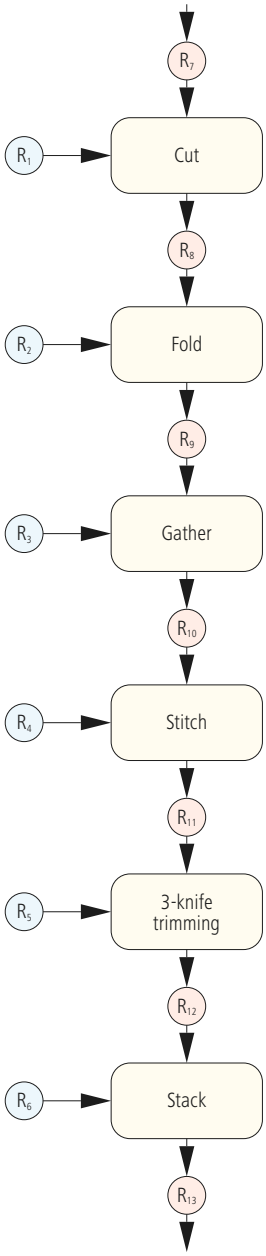


Figure 3.14
Process-resource model for the production of a stitched brochure

ularly important for the folding process, are not entered In R_2, instead in resource R_8.

We have now seen examples of workflow descriptions in the form of activities lists, activities diagrams, flowcharts, and process-resource models in the areas of prepress, press, and postpress. We believe that the process-resource model is particularly suited to clearly and simply visualize complex process flows. In contrast, a flowchart can be very helpful to illustrate the "plot lines" within a process. Activities lists are perhaps the easiest to create, but allow the least information to be revealed. They also lead to wild entries rolled together so that in the end an unstructured "brainstorming" result remains left over.

Exercise:

Plan your next weekend with the help of a flowchart. Define alternatives to activities if conditions are not able to be met.

Describe in detail the preparation of your favorite dish as a process-resource model—and do not forget an appetizer and dessert!

4 History of Metadata and Its Application

The difference between *content* data (print data) and metadata was discussed and defined previously in Section 2.4. Here, we will present some examples of the corresponding applications.

We will begin in Section 4.1 with the EXIF format, which allows technical values and calibration to be captured while shooting with digital cameras. Also, we will present the IPTC (International Press Telecommunications Council) standard for photos.

XMP stands for **Extensible Metadata Platform** and is specified by Adobe for different image and document formats. The idea behind the concept is the dissemination of information about the application's boundary. By way of example, metadata written in a TIFF image file is not only in the InDesign document in which this image is placed, but also in the PDF file that is exported out of InDesign. So you may find copyright information about photos in a PDF output file when reprinting even years later although the image file has not been available for a long time.

In Section 4.2 **Print Production Format** shall be discussed in more detail, having already been introduced briefly in Chapter 1 as an important predecessor format to Job Definition Format.

The last section of this chapter is about another precursor to JDF, the **Portable Job Ticket Format** (PJTF) from Adobe Systems Incorporated. This format is closely tied with the obsolete, interim **Extreme RIP architecture** from Adobe. This RIP architecture was licensed for many years to many manufacturers of CTP workflow systems and is often still found in the market.

The contents of this chapter are not directly required for the understanding of the other JDF-based chapters and may be skipped. Because many workflow installations currently (still) use metadata in Babylonian language diversity, especially JDF/JMF, PPF, and PJTF, we have included the basics of classical metadata in this book.

Outside of this, it is entirely conceivable that XMP and JDF will together augment a workflow implementation.

4.1 Metadata for Photos and Documents

There are three main reasons why one provides photos with metadata:

- Photos may be more easily identified and found in large archives.

- Copyright infringement out of ignorance may be hindered by copyright messages in the photos.
- A description of the technical details of the photos makes the analysis of the camera settings possible afterwards and also gives a quick overview of the suitability of the photos for different applications.

The most important issues in the design of photo metadata is (1) the definition of which information about the photos should be recorded and (2) how this data is maintained when the format is converted.

EXIF and IPTC

EXIF stands for **Exchangeable Image File** format and was developed by the **Japan Electronics and Information Technology Industries Association** (JEITA) [27] especially for digital cameras. In addition, scanning software makes some of this metadata available. For each photo that is taken with a digital camera, details about the camera, the time the photo was shot, and the settings at the time the photo was shot (exposure time, resolution, aperture, etc.) are stored in the image file. With various applications, this data may then be displayed and also modified and analyzed. In Figure 4.1 you can see the EXIF data which was provided automatically by the digital camera.

The EXIF specification Version 2.2 [28] defines many more fields than the figure shows, such as copyright, comments, or GPS information. It also describes the EXIF data structure for both JPEG as well as TIFF image files. Indeed, other formats, including PDF, are not supported. In order to carry over the EXIF entries in this format, the information is built into other structures as shown in the following section about XMP.

EXIF information may be incorporated into TIFF and JPEG because both data formats are extensible. The "T" in TIFF stands for *"Tag"* (or *"Tagged"*). In fact, it says that each unit of information about the recorded image (height, width, etc.) is recorded with a defined value in the form of a positive integer. In total there are 215 (=65536) different tags available, and because only a few are documented now, many more additional tags may be defined [3]. For JPEG images, the extensibility is realized through a similar technology. The images compressed by the JPEG standard are normally saved according to the **JPEG File Interchange Format** (JFIF) [20]. As with TIFF, the individual units of information are saved with iden-

tifiers which are called *"segments"* here and not *"tags."* The number of EXIF data is, for example, 255.

Some digital cameras save their photos in proprietary RAW format. In this case, EXIF data is typically saved in a separate text file.

The **International Press Telecommunication Council** [26] is a worldwide association, representing news agencies and national newspaper associations. In the 1990s they defined a set of metadata properties to be applied to images, which in many aspects is similar to EXIF and is generally known under the term **Information Interchange Model** (IIM). Here, also, ownership rights may be noted, but above all the images are described by keywords (Figure 4.2).

XMP

In 2004, with the **Extensible Metadata Platform** (XMP) specification, Adobe brought out a broad sweeping specification [2] which not only defined metadata for images, but also for documents of differing types. The idea for the specification is that meta-

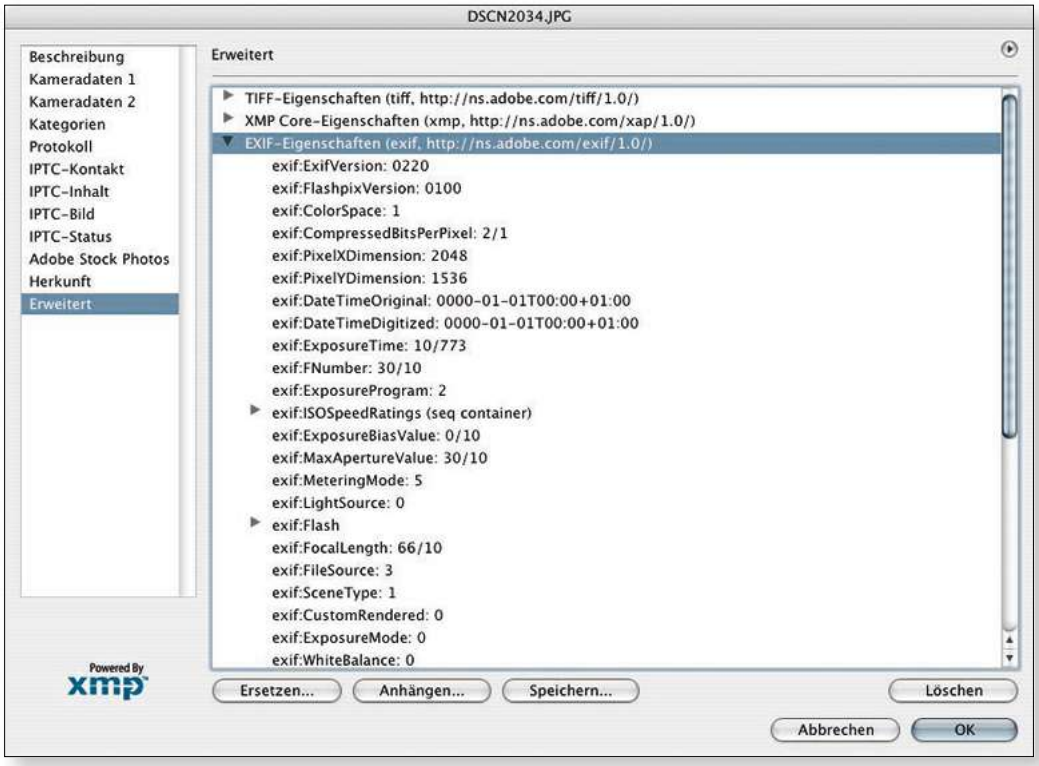


Figure 4.1
EXIF data of a JPEG image

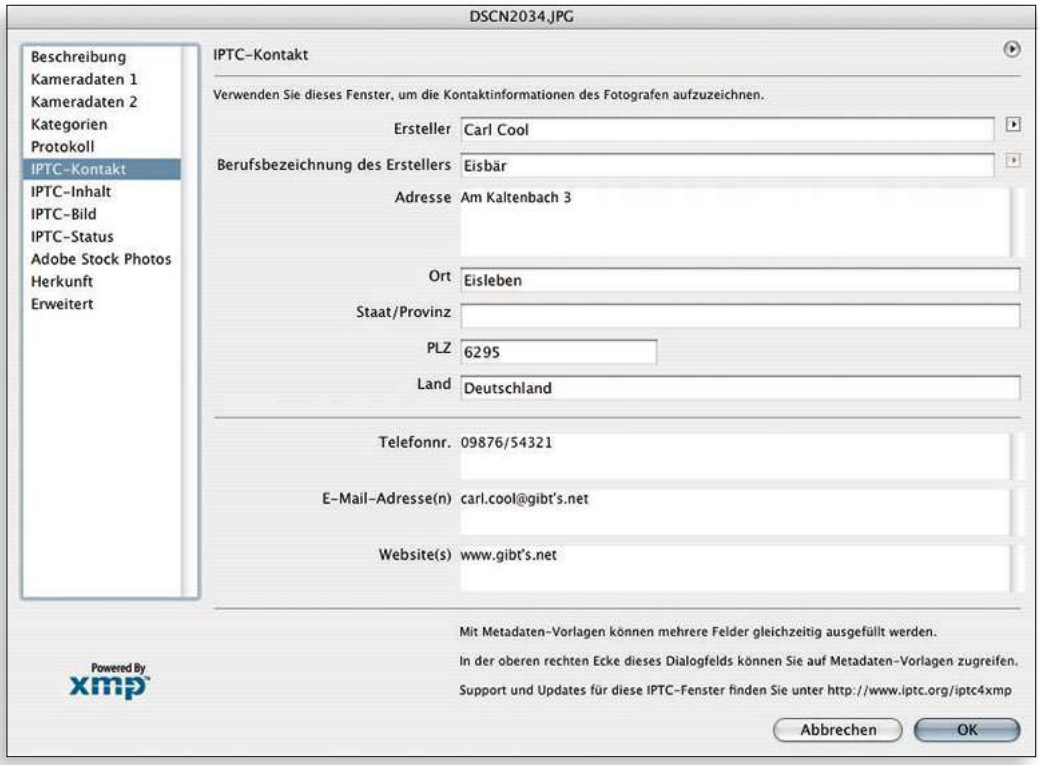


Figure 4.2
IPCT data within a JPEG file

data, once entered, will remain despite format changes. This naturally assumes not only that the embedding of metadata in a special data format appropriate to the XMP specification is at all possible (like TIFF, JPEG, JPEG 2000, GIF, PNG, HTML, PDF, AI, SVG/XML, PSD, PostScript, and EPS) but also that the applications which carry out the format transformations alter XMP data.

In fact, XMP information is stored separately and outside of the application files. Yet that is rather atypical and will at the utmost be made in database entries.

The XMP data may belong not only to a complete document but also to individual components of a document. One such situation allows, for example, the graphics and images which are embedded in a PDF document to be associated to their own XMP metadata. Figure 4.3 illustrates this principle whereas Figure 4.4 represents a page of this book in which the XMP information is visible. The word "resource" is employed in the XMP language which may be may be explained through XMP. A resource is either a document or a meaningful component of a document. The XMP data can be stored in XML syntax. Since we handle XML in the next chapter, it

suffices to say here that XML is easily extensible in that one can define new schemas, and therefore several schema can be used simultaneously. Adobe has several predefined schemas, such as:

- Dublin Core Schema
- XMP Basic Schema
- XMP Rights Management Schema
- XMP Paged-Text Schema
- IXIF Schema for IXIF-specific Properties

The complete list can be found in [2]. Just a few of the examples of the many possible entries which can be realized through

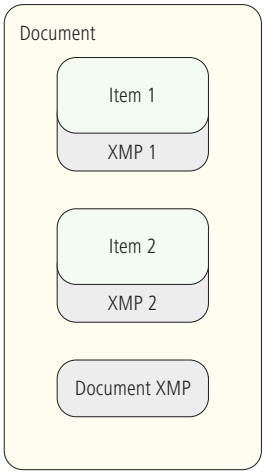
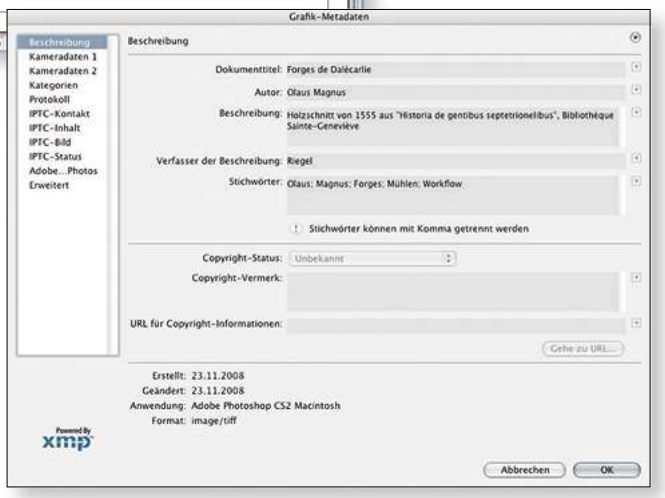


Figure 4.3
XMP principle components of a document

Figure 4.4
XMP information of an image within a PDF file



this schema are listed here. The *Title* and *Language* of a resource are properties which are defined in the **Dublin Core Schema.** The **XMP Basic Schema** makes it possible to capture information about the creation date (*CreateDate*) or Application Name (*CreatorTool*) with which the resource was produced. The **XMP Rights Management Schema** shows the ownership rights, and, for example, with the **XMP Page-Text Schema,** the page numbers (*Npages*) of a document can be recorded. In the EXIF schema most of the EXIF entries are defined yet again under XMP. This allows the metadata of the digital photos to be integrated into the XMP structures as well.

Of course not only Adobe but other firms may also define such schemas and then a few build vendor-specific information in the data. Thus a workflow management system (WMS) vendor may use the XMP options in order to convey metadata and control their workflow. A further example of the options XMP offers is shown in Figure 4.5. How a software vendor can wrangle its application into an existing WMS is outlined here. In Step 1, for example, the application outputs PDF data out of the job order of the WMS and saves the memory location of each file in an XMP entry. In Step 2 the work is carried out and thereby modifies the PDF files. Lastly, the data is then copied back again in the original order of the WMS according to the XMP entries.

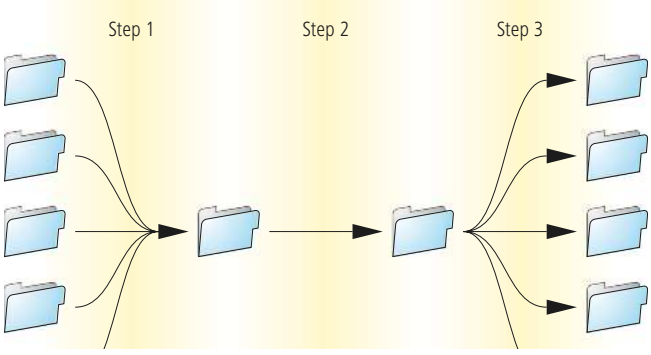


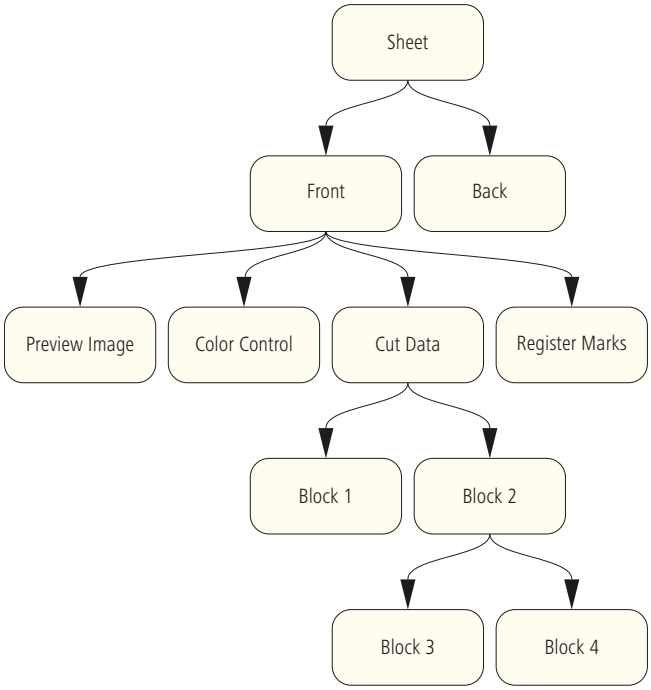
Figure 4.5
Third-party software within a WMS

In summary, the metadata that we have covered—EXIF, IPCT, and XMP—are descriptive, i.e., they describe properties of digital resources. Its core properties are quite limited, but expansion options, at least by XMP, are available. This type of metadata cannot describe processes which are required to modify the data. Production software may, of course, use the metadata in order to decide on predetermined design alternatives. Since the metadata is typically integrated together with the actual resource data in a file, they are naturally limited to the production areas which handle digital documents, and therefore only prepress. Both of the metadata formats which are handled in the next two sections are, in principle, completely different.

4.2 Print Production Format (PPF)

Back in the Introduction we presented the two most important applications of PPF: transfer of data from prepress to the calculation of color zone settings for an offset printing press, as well as for the calculation of cutting and folding programs for postpress (finishing). In the "Extensibility" Section 2.6, we mentioned that PPF is encoded in PostScript and presented a very brief example.

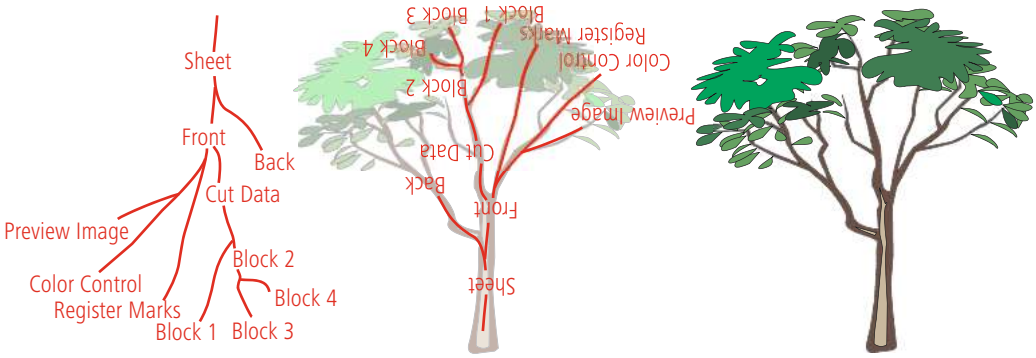
Figure 4.6
Structure of a PPF file



With the help of the description of a press sheet in Figure 4.6, Figure 4.7 shows that PPF information, in its logical structure, is built like a tree. The sheet forms the **root element,** all forks are tree branches, and all of the data points that have no branching are leaves. Preferably, instead of leaves and branches one would speak of **nodes.** Each node, except for the root element, is a child of the parent node; all nodes except the leaves are also parent nodes. Since in a tree structure each node only has one (a single) parent, one usually uses the technical term "parent" instead of "parents."

An important feature in the PPF tree structure (and also with many other tree structures) is that of

Figure 4.7
Tree structure



inheritance. Children inherit all properties from their parent, but they may also bear new values. Furthermore, they can also hold new properties. As an example, the nodes "Front" and "Back" are children of the "sheet." The properties of the sheet, such as the size (*CIP3AdmPaperExtent*) and the gram weight (*CIP3AdmPaperGrammage*), are inherited from it. Perhaps they also inherit the property "screen type" (*CIP3ADMTypeOfScreen*), but either "Front" or "Back" may override the screen type. Finally, it may be that the screen type is not defined at all in the sheet, but instead only in the two children.

Naturally more than one sheet within a PPF may be described; they are children of another root element called the **PPF Directory.**

Figure 4.8
PPF example

The tree structure in PostScript (PS) code incidentally is realized through a clinch. Figure 4.8 shows the tree structure of Figure 4.6 in this way.

```
CIP3BeginSheet
    CIP3BeginFront
        CIP3BeginPreviewImage ...
        CIP3EndPreviewImage
        CIP3BeginColorControl ...
        CIP3EndColorControl
        CIP3BeginCutData ...
            CIP3BeginCutBlock ...
            CIP3EndCutBlock
            CIP3BeginCutBlock ...
                CIP3BeginCutBlock ...
                CIP3EndCutBlock
                CIP3BeginCutBlock ...
                CIP3EndCutBlock
            CIP3EndCutBlock
        CIP3EndCutData
        CIP3BeginRegisterMarks ...
        CIP3EndRegisterMarks
    CIP3EndFront
    CIP3BeginBack
    CIP3EndBack
CIP3EndSheet
```

In Figure 4.9 we again see a somewhat more detailed code snippet from a PPF file. The first and last lines each begin with a percent sign, which in PostScript is a comment character. This means the lines are only PS comments and are not interpreted by the PS interpreter. The following lines are more or less self-explanatory. The paper format is given in DTP points; it therefore has a width of $1984{,}251968 \times 2.54/72 = 70$ cm and the height of $1417{,}32283 \times 2.54/72 = 50$ cm. The gram weight is defined as usual in g/m², the paper color in CIE L*a*b.

The best-known application for PPF is the color zone presetting for offset presses. The PPF therein actually takes a relatively modest role because it will save only the preview images (*Preview*) of the separated assembled sheets and some additional information in PPF which allows special interface software to calculate the zone values (see Figure 1.3). The zone preset values are not part of the PPF standards. Figure 4.10 shows a part of such preview images as PPF structure. The images themselves

```
%!PS-Adobe-3.0
...
CIP3BeginSheet
    /CIP3AdmArtist (Carl Cool) def
    /CIP3AdmJobCode (8) def
    /CIP3AdmJobName (8 Zustaende) def
    /CIP3AdmPaperExtent [ 1984.251968 1417.32283 ] def
    /CIP3AdmSheetName (FB 002) def
    /CIP3AdmTypeOfScreen (amplitude modulated) def
    /CIP3AdmPaperGrammage 100.0 def
    /CIP3AdmPaperThickness 0.035 mm def
    /CIP3AdmPaperColor [ 93.0 0.0 -3.0 ] def
    /CIP3AdmCreationTime (Mon Dec 11 18:29:57 2006) def
     ...
CIP3EndSheet
%%CIP3EndOfFile
```

Figure 4.9
Details of a PPF file

are compressed by length and are coded in ASC1185. The width (*CIP3PreviewImageWidth*) and the height (*CIP3PreviewImageHeight*) is in pixels, the resolution (*CIP3PreviewImagesResolution* is defined in pixels per inch (ppi). The latter is down to 50.8 ppi. The *CIP3PreviewImageMatrix* specifies the pixel direction; here it is defined from left to right and from top to bottom. One can find the definition of matrix algebra (calculation) in the PS and in the PDF specifications (see also the exercises at the end of Chapter 9).

In fact, it is not enough to give only the preview data to the calculation software for the color zone preset values. Because normally a change of tone values still occurs after the calculation of the preview image for the assembled sheet. And after all, it is indeed the

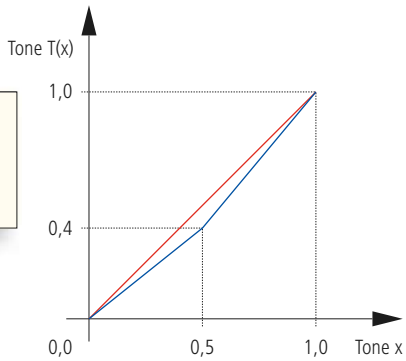
Figure 4.10
Definition of a preview image in PPF

```
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
        CIP3PreviewImage...image data
        CIP3EndPreviewImage
    CIP3EndSeparation
    same for separations of Magenta, Yellow, and Black
CIP3EndPreviewImage
```

tonal values that are important for calculating the ink zone preset values. But who or what changes the tone values? For the production of offset plates, the image, which indeed still has a color depth, must be screened. However, usually only one tone modification is made to the screens. By way of example, the tonal value modification should match the tonal value of a standardized dot gain in print. In the parlance of PS and PDF we call these curves **transfer curves,** or, depending on the purpose, **linearization curves, process calibration curves, tonal value compensation curves,** or in flexo printing **bump-up curves.** In Figure 4.11 two transfer curves and their descriptions in PPF are shown. It is always listed as a set of coordinate pairs (namely x and $T_{(x)}$), between which all of the definitions of the remaining functional values are interpolated. Curve A is a transfer curve that creates no tonal value changes, while curve B reduces the tonal values in the middle tones.

Figure 4.11
Transfer curves in PPF

```
A  /CIP3TransferPlateCurveData
     [0.0 0.0 1.0 1.0] def
B  /CIP3TransferPlateCurveData
     [0.0 0.0 0.5 0.4 1.0 1.0] def
```



A preview image and the transfer curves may be furnished by a RIP, but not cutting, register, color, and fold marks, as they arrive at the RIP only as meaningless common graphics. Only the assembly software knows the meaning of these marks and can assemble the appropriate PPF Information. The positions for register, color, and cut marks on the sheet will be specified in the PPF file. However, instead of cutting marks, cutting blocks can also be entered into a PPF file. In this case, no cutting sequences are defined, but only certain panel sizes which must be cut as shown in Figure 4.12. With the folding data, in contrast, folding sequence can also be established.

So far, all of the PPF properties have been related to the print sheet. After that, the scope of the format was also limited to version 2.1. With version 3.0, which was released in 1998 and the last version of this job ticket format released, broader product definitions have been possible. It allowed even more subproducts to be de-

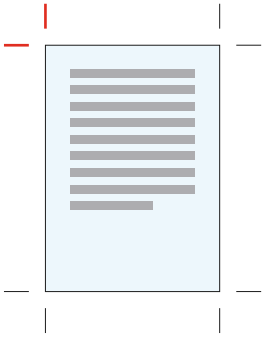
scribed within an overall product in a PPF file. Altogether, it was followed with the introduction of "Product Operations" (*/CIP3ProductOperation*) and "Product Parameter" (*/CIP3ProductParams*) of the process-resource model, introduced in Chapter 3. A range of product operations such as collating, stapling, smyth sewing, or the execution of three-knife trimming are defined in 3.0.

In the examples presented, we were introduced to two PPF producers, namely the RIP and the assembly software, as well as three PPF consumers, namely the respective PPF interface module for the printing press, the folding machine, and the guillotine cutter. Therefore this unfolds into the theory of the structural image which is shown in Figure 4.13. Unfortunately, in practice it is not so easy for different PPF producers to generate a PPF file together. The PPF specification also provides no guidelines on how it should be done and no vendor-specific solutions. Consequently, one likely has more to do with multiple PPF data per print order with a multi-vendor workflow, assuming there are multiple PPF producers in the mix.

But because in some circumstances a PPF consumer needs information from different PPF producers, the situation is somewhat

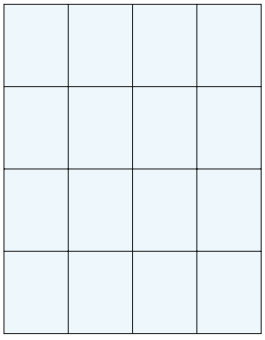
Figure 4.12
Alternative options for providing cutter information in PPF

Cut lines for cut mark positions



```
29.4 cm 5.0 cm /TopVerticalCutMark CIP3PlaceCutMark
29.4 cm 5.0 cm /LeftHorizontalCutMark CIP3PlaceCutMark
```

Alternative cutting block



```
/CIP3BeginCutBlock
    /CIP3BlockTrf [1 0 0 1 1 4 cm 4 cm] def
    /CIP3BlockSize [29.9 cm 68.7 cm] def
    /CIP3BlockElementSize[10.1 cm 6.02 cm] def
    /CIP3BlockSubdivision [4 4] def
    /CIP3BlockType /CutBlock def
    /CIP3BlockElementType /Unknown def
    /CIP3BlockName (Block1) def
/CIP3EndCutBlock
```

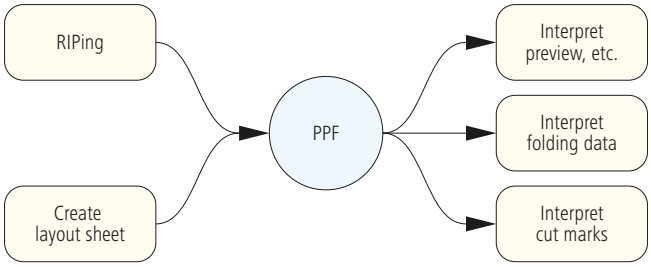


Figure 4.13
PPF producer and PPF consumer

messy. A possible example is the PPF printing press interface that receives a preview image from the RIP and position information about the register marks from the assembly software. Thus the PPF workflow is difficult to maintain with the complexity of increasing functionality: It must be set up with a variety of hot folders and controlled in case of error. Archiving the scattered PPF files is cumbersome.

Overall it is therefore perhaps not surprising that the PPF workflow is often limited only to the passing of the RIP data for the color zone information because the benefits (reduction of waste and setup time) are very high in relation to the cost.

4.3 Portable Job Ticket Format (PJTF)

The **Portable Job Ticket Format** [6] from Adobe Systems, Incorporated is the precursor format of JDF. This format is closely linked to the Extreme RIP architecture that, like PDF, was developed by Adobe.

The idea of the Extreme RIP architecture is to define those functions resident in the RIP as independent modules which are supplied with the necessary metadata by PJTF. The modules convert job tickets and are therefore called **job ticket processors** (JTP). JTP examples are not only the classic tasks of a RIP—that is *interpreting, rendering, and screening*—but also additional functions such as normalization, color space transformation, trapping, imposition, calculation of the proof data, or creating a PPF file for color zone presetting. The JTPs can run on one server, but also on different servers in a distributed system. In many cases, the trapping JTP is outsourced to its own server because it is a fairly time-consuming process which requires much CPU and working memory capacity (RAM). Multiple instances of JTP are also often set up on one or more servers in order to improve the overall throughput. Extreme RIP architecture was for many years the basis for most workflow management systems in the arena of CTP and only gradually replaced by the new Adobe **Print Engine** technology beginning in 2006.

The JTPs are controlled from a coordinator module, which receives the job tickets from the outside and distributes the information to

the JTPs. In general, the layout sheet definition as well as the specific settings for the different JTPs are translated from client software by the coordinator. A supplier of an Extreme RIP system can integrate their own JTPs into the system or draw on the JTPs from Adobe. In any case the coordinator must be licensed by Adobe. The user cannot integrate foreign JTPs (for example, from other manufacturers) into an existing system since the interfaces are not exposed: only the contents of the PJTF file are specified, but not how they flow into the JTPs (Figure 4.14)

The following items may be specified in PJTF:

- Data for order management (delivery addresses, press date, etc.)
- Specifications of the printed sheet
- Calibration values for processes in the area of output workflow (preflight, trapping, screening, etc.)
- Calibration values for the processes of print finishing

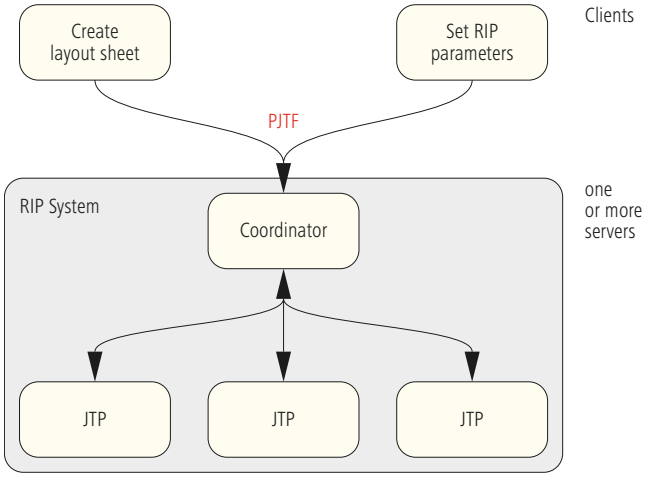


Figure 4.14
Architecture of the Extreme RIP

The focus lies firmly on the imposition layout specifications and the CTP output workflow; all of the other areas are in the specification have been somewhat neglected.

In the previous sections it was stated that the XMP format is coded in XML and PPF is coded in PostScript. PJTF is now based on PDF, which means that a PJTF file is structurally a PDF file only with PJTF-specific keywords substituted. However, these keywords are unknown to Acrobat, so their values cannot be displayed. A PDF file generally consists of a collection of numbered objects and a cross-reference table in which they are registered, where the exact location of each object is to be found. Figure 4.15 shows a PJTF object with the number 13. The object contains, similar to PostScript-coded PPF, a type of table in which each line is always provided in the same structure: on the left is a string, and on the right is the value of this string. This structure may also be interpreted as a keyword (or variable name) and value or also as an entry in a (rather peculiar) *dictionary.* In

fact, Adobe also named this structure "dictionary." Each dictionary is framed by double angle brackets (<<and>>). The same is true for PostScript and PDF.

Object 13 in Figure 4.15 defines details about trapping. The keywords are defined in the PJTF specification. It is */BCL* for *Black Color Limit* and the indicated value of 0.95 for the limit, that each tonal value of the color black over 95% should be flooded like the solid tone black. And the key */ITO* and the indicated value should be images to other objects overfilled or under filled (*ImageToObjectTrapping*). The trap width is marked with */TW* and is given in 0.2 inches.

Figure4.15
A PJTF object

```
13 0 obj
<<
    /BCL 0.95
    /ITO true
    /TW 0.2
    ...
>>
endobj
```

Like XMP and PPF, PJTF is also built in a tree structure. The reference between the nodes is realized through a reference from one object to another. The reference is marked by an "R," whereby the object number is placed before the object with is being referenced.

We will now explore a specific branch, which defines a layout sheet. Each rectangle in Figure 4.16 represents an object in the PJTF example; each arrow symbolizes the reference to another object. The IDs of the objects are marked in red above the rectangle.

The root element in PJTF is the object with the name *JobTicket.* It refers to two objects, the *JobTicketContent* and the *Audit* object. All of the changes to the PJTF file that are made during the course of production are logged into the *Audit* object. The *JobTicketContent* object refers to, among other things, the *Signature* object, in which sheets with the same imposition layout can be combined. Not visible in the figure are further references to the *JobTicketContent* object, for example objects which specify the memory location in the file system where the PDF files may be found and where on the layout sheet marks are placed. The *Signature* object 79 consists of two *Sheet* objects 60 and 78. This means, in the example, that the signature contains two print sheets. The object *Sheet* with the number 60 in turn references to two objects of type *Surface* (surfaces), namely one for the *Front* (straight) and one for the *Back* (wider), which are represented by the objects 57 and 46. Moreover, there is a reference to the *MediaSource* object with the marking 59 in which data about sheet size, sheet color, etc., are found. Here also the tree is not completely shown. Object 36 shows the *PlacedObject*, which includes the characteristics of the side placeholders such as position, size, and trim.

The actual code of Objects 79, 60, 46, and 38 is shown in somewhat reduced detail in Figure 4.17, and arrows more graphically indicate where these references are. In the first line of each object,

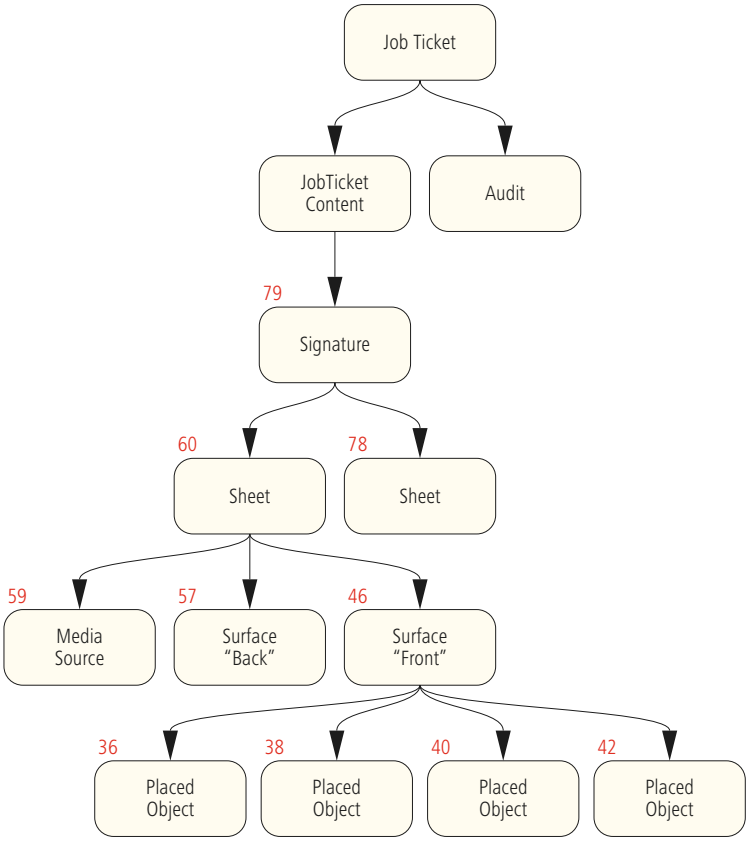


Figure 4.16
Structure of a PJTF document

its ID is defined and the dictionary is opened with <<. The second line indicates what type of Object it handles: *Signature, Sheet, Surface,* or *PlacedObject.* All other entries in the respective dictionaries are object-specific and are not further explained here. As an example, only the entry */CTM{0.0…0.0}* in Object 38 should serve to define, in the **current transformation matrix,** the size and position of the side placeholders on the sheet. Each dictionary and any object with the keyword *"endobj"* are closed with >>.

In summary, it can be said that PJTF will certainly be completely replaced by JDF, but currently some workflow management systems still import PJTF layout sheets, for example.

Exercise:

Open a digital photo in Photoshop and fill out the file information.

Place the photo into InDesign.

Export the InDesign document to PDF.

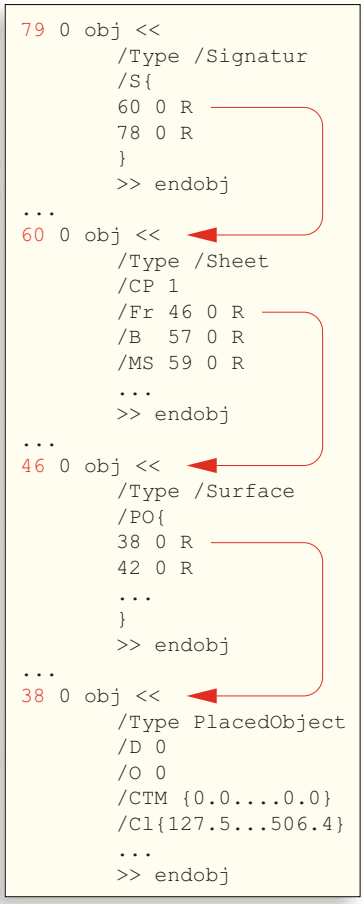


Figure 4.17
References between objects

Examine the PDF file in Acrobat with respect to the XMP data. Select the image and view the images properties in the context menu.

Examine the PDF file using a suitable text editor such as WordPad or MFC applications.

Search for "XMP" and examine the entries there.

5 A Brief Introduction to XML

Extensible Markup Language is, in fact, not a language; instead it is more like a system to define a language, that is, a metalanguage (a language about a language). XML is also a standard for defining the document types programs can exchange between themselves. An example for such an XML document type is Job Definition Format. That is the only reason we're dealing with XML here, so there is no need to treat the XML topic exhaustively in this chapter. Instead we will only present the terminology needed to understand JDF more easily. We will also not address the pros and cons of XML in relation to other document structures (such as PDF or PostScript) or compare to **Standard Generalized Markup Language** (SGML) or **Hypertext Markup Language** (HTML). A somewhat more exact, and much more complete, introduction can be found as an example in [42].

In Section 5.1 we will illustrate the practical construction of an XML document and explain terms such as *elements, attributes,* and *values.* Section 5.2 goes into definition of the options, the "vocabulary," of XML files which are combined into name spaces. Finally, in Section 5.3, **Commerce Extensible Markup Language** (cXML) is introduced, an XML-based standard for the electronic exchange of business data. cXML, in turn, forms the basis for parts of the JDF specification.

5.1 Construction of an XML Document

Generally, an XML file begins with the so-called "declaration," a prolog for the actual XML elements. This declaration can be recognized because it is within angle brackets (< and >) or even question marks (?). Additionally, in Figure 5.1, the specification of the version of XML, namely 1.0, appears in the declaration. Although version 1.1 was released in February 2004, version 1.0 is still often used. The version number is documented via an **attribute** that consists of an attribute name (here, *version*) and an attribute value bracketed in quotation marks (here, *"1.0"*). The attribute value is assigned to an attribute name through the equals sign (=). This is also called **value assignment.** Often one speaks of an attribute, referring only to the attribute name, saying, for example, "version." The various attributes are separated from each other by a space.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Figure 5.1
Declaration of an XML document

The *version* attribute is obligatory and must always be in an XML declaration—all other attributes are optional. This example also shows the optional *encoding* attribute, which furnishes information about the coding type of the present XML document. UTF-8 is an international coding based on the ISO/IEC-10646-norm with at least 8-bit character width. UTF stands for **UCS Transformation Format** and UCS for **Universal Multiple Octet Coded Character Set.** The UTF character set is based on the better-known Unicode character set.

In most cases, the root element of the XML data structure comes after the declaration. Within JDF, the *root element* is always an element with the name of JDF, which we will see in the next chapter. Let us not worry about JDF for now and illustrate instead another root element with the name of *Contact.* **Elements** are in a sense the building blocks of the XML structure. Each element has a name, for example *Contact, Person,* or *ComChannel.* Elements are delimited by start tags and end tags, and the tags are identified with angle brackets. The start tag of the element *Contact* is therefore <*Contact*>, the end tag is <*/Contact*>. The end tag is signified by the forward slash (/). Elements may contain subelements, so the element *Person* is a subelement, also known as a **child element,** of *Contact.* The three *ComChannel* child elements of *Person* are referred to as **siblings,** and *Person, Company,* and *Address* are sibling elements under *Contact.* The tree structure of the example coding in Figure 5.2 is more clearly depicted in Figure 5.3.

Not all elements have explicit start and end tags. The *Company* element contains no further subelements and may therefore be written without an end tag. However, a forward slash (/) must come before the final angle bracket. Elements that have no further subelements are usually identified as "empty." As such, only empty

Figure 5.2
XML elements

```
<Contact>
    <Person FirstName="Carl" FamilyName="Cool" NamePrefix="Herr">
        <ComChannel Locator="03475/101010" ChannelType="Phone"
         ChannelUsage="Private" ChannelTypeDetails="Landline" />
        <ComChannel Locator="03475/101011" ChannelType="Phone"
         ChannelUsage="Business" ChannelTypeDetails="Landline" />
        <ComChannel Locator="carl.cool@frisch.de
         ChannelType="E-Mail" ChannelUsage="BusinessPrivate" />
    </Person>
    <Company OrganizationName="Frisch GmbH"/>
    <Address City="Eisleben" Street="Am Kaltenbach 3" Country="Deutschland"
     PostalCode="06295" />
    <!--This is a simple XML structure, which also happens to be JDF code-->
</Contact>
```

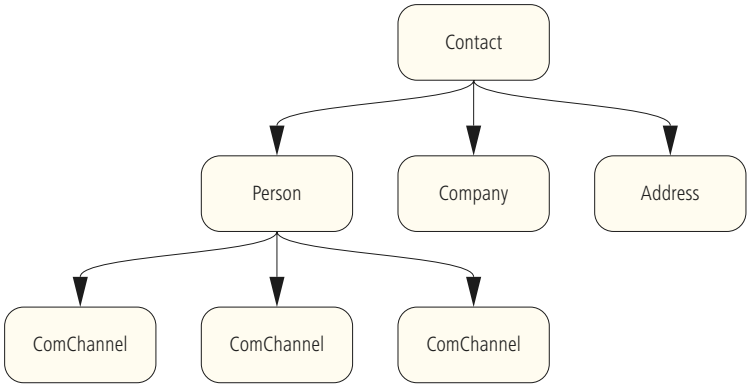


Figure 5.3
Tree structure of XML elements

elements may be written in the form <Elementname…/>. As you can see in the example, empty elements may certainly contain their own attributes.

Many, but not all, elements have attributes, as we see in the example of *Contact.* Which property of an element must be written as an attribute and which is defined as a subelement must be defined by specifying a document type. In our example, the attribute of the *Person* element could naturally also be defined simply as an attribute of the *Contact* element. Then the *Person* element would be superfluous and could be removed. Attributes and elements are case-sensitive. Also, not all characters are allowed in the element or attribute names, such as spaces or XML-specific characters such as angle brackets or the apostrophe.

Comments may be written in an XML document, which are defined only for people to read and not to be processed by the XML software. Comments begin with "<!--" and end with "-->" and may go over multiple lines.

5.2 XML Name Space

Data exchange with XML documents by massive applications is found in many fields. They go from descriptions of chemical molecules, the page description language XPS from Microsoft (**XML Paper Specification**), to business-to-business (B-to-B) interfaces for the exchange of business data like purchase orders, invoices, and product catalogs. But how are the necessary structures, element types, and their attributes defined for those purposes? This **Markup Vocabulary** and the rules of how to use them are not maintained from a central point; instead each may define their own model. When XML documents are to be used for data exchanges between different software modules, agreement on the

markup vocabulary is necessary. This naming convention is made through the declaration of a name space in the XML document. A name space is not necessarily declared in each XML document, for example, when it is only for internal or even for private purposes and no naming conflicts from double elements or double attributes can occur. But within JDF documents, for the most part, more name spaces are declared: one for the description and structure of all overall JDF elements and their attributes with respect to the CIP4 specification, and also to designate private supplements which the workflow vendor puts out.

An XML name space is declared by the *xmlns* attribute. The value of the attribute is simply an identifier, specifically a **Universal Resource Identifier** (URI). This identifier, as shown in Figure 5.4, generally looks like an Internet address, but actually refers to the name of the namespace. A resource on the Internet can stand behind the URI, but it doesn't need to. In other words, the URI can potentially be a **Uniform Resource Locator** (URL), but in general it is not. The example shows how one can easily believe that the first namespace declaration is a URL, while the second is only a URI (the Internet address does not exist).

In the second declaration behind the attribute name *xmlns* a colon separates the prefix *HdM,* where *HdM* only stands for "Hochschul der Medien" (Stuttgart Media University). With the help of such a prefix, it is then possible to associate element names or attribute names to different name spaces. When a name space does not contain a prefix, it is the default name space. In the example, the *ResourcePool* element is a subelement of the JDF element. Both element names stem from the default name space, which, given without a prefix, is *http://www.CIP4.org/JDFSchema_1_1,* while the element *HdM:PrivateElement* is associated with the second name space with the prefix *HdM.*

The structure of an XML document type can be defined in a **Document Type Definition** (DTD) or in an **XML Schema;** the schema is more modern and offers wider-reaching possibilities. We do not worry about the details as to exactly how the definition of a doc-

Figure 5.4
Namespaces in XML documentation

```
<?xml version="1.0" encoding="UTF-8" ?>
    <JDF xmlns="http://www.CIP4.org/JDFSchema_1_1"
     xmlns:HdM="www.hdm-stuttgart.de.com/schema/HdM"...>
        <ResourcePool...>
        ...
        </ResourcePool >
        <HdM:PrivateElement... />
    </JDF>
```

ument type is to be carried out. It is not important for the understanding of the relationships in this book. XML schemas are also sometimes shortened with XSD (**XML Schema Definition**). The file that contains the JDF schema is also named accordingly, "JDF.xsd," and may be downloaded from the CIP4 website. A schema itself is again defined in XML and consequently coded as text; it can be read and also edited with a normal text editing program or with a browser.

The following are usually defined within a schema:

- Which elements are allowed in a document
- Which attributes are allowed in these elements
- Which attributes are obligatory and which are optional for an element
- The parent-child relationships between the elements
- The data types for elements and attributes
- Frequencies of elements
- References between elements

XML documents must be well formed and valid. We take "well formed" to mean compliance with the general XML syntax, while validity can only be given if the additional rules which are defined in the schema (or schemata) are met. The XML document must define under which schema or schemata it is structured.

A program which can read XML is called a **parser** or could be called **analyzer** here: A parser only has the task of analyzing the XML text in particular to examine if it is well formed. The parser also checks the validity of the XML document, and so it is also referred to as **validating parser.** The advantage of the validating parser is that XML documents, which are not constructed according to the schemata, are immediately known and can be separated out if need be. This reduces the risk that an XML document cannot be processed correctly. A validating parser is to XML as a preflight program is to PDF files.

5.3 Resource Description Framework (RDF) and XMP

The XMP format was covered in the previous chapter, which also noted that it can be written in an XML markup language. This language is called the **Resource Description Framework** (RDF) [44]

and is also a system for the description of resources, especially on the Internet. It's used mostly for the storage of metadata for Internet resources as well as for additional information which, in the simplest case, should allow for more effective Web searches. Additionally, it does more, namely around the concept of the semantic Web, in which the meta information could be read and evaluated not only by search engines but also other programs, from so-called Web agents.

Although RDF is not important for JDF, we still want to introduce it briefly in order to describe the data presentation of XMP. We want to examine Figure 5.5 in somewhat more detail, which shows an RDF/XMP object in PDF format: the element *<x:xmpmeta>* is the root element that has exactly one child element *<rdf:RDF>*. The latter surrounds itself with three child elements *<rdf:Description>*.

Figure 5.5
XMP coded in XML

```
20 0 obj
...
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP toolkit 2.9-9, framework
 1.6">
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:x="http://ns.adobe.com/iX/1.0/">
      <rdf:Description rdf:about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/"
       pdf:Trapped="False" pdf:Producer="Acrobat Distiller 7.0 (Windows)"
       pdf:GTS_PDFXConformance="PDF/X-1a:2001"
       pdf:GTS_PDFXVersion="PDF/X-1:2001">
      </rdf:Description>
      <rdf:Description rdf:about="" xmlns:xap="http://ns.adobe.com/xap/1.0/"
       xap:CreateDate="2007-08-24T12:00:39+02:00"
       xap:CreatorTool="PScript5.dll Version 5.2"
       xap:ModifyDate="2007-08-24T12:00:39+02:00">
      </rdf:Description>
      <rdf:Description rdf:about=""
       xmlns:dc="http://purl.org/dc/elements/1.1/">
         <dc:creator>
            <rdf:Seq>
               <rdf:li>Carl Cool</rdf:li>
            </rdf:Seq>
         </dc:creator>
         <dc:title>
            <rdf:Alt>
               <rdf:li xml:lang="x-default">
                test.indd
               </rdf:li>
            </rdf:Alt>
         </dc:title>
      </rdf:Description>
   </rdf:RDF>
</x:xmpmeta>
...
endobj
```

In each *rdf:Description* element a separate name space is defined: the first with the prefix *pdf* is saved to PDF files, the second with the prefix *xap* goes to the authoring with respect to modifications data of the file, the third with the prefix *dc* around the author and title of the file which stands for "Dublin Core" (see Chapter 4).

A few lines in the example may still be unclear: the element *<rdf:Seq>* describes an ordered list; each list entry is a *<rdf:li>* element. The list here consists of only one entry. The element *<rdf:Alt>* provides the possibility of specifying alternative values. Something notable appears at once: the attribute *rdf:about=""*. RDF was, as said, developed primarily in order to describe Internet resources. The value of the attribute *"about"* then provides the URI of the corresponding resource in the Internet. Yet here one has an empty string which, in the XMP example, doesn't go with the metadata of an Internet resource but with the metadata of a PDF file on the local file system in which the XMP information within the PDF file is saved.

5.4 Commerce Extensible Markup Language (cXML)

Product or service catalogs, bid requests, quotations, placed orders, order confirmations, or invoices are sent back and forth between business partners in business life. That occurs not only through letters or emails but also through Web portals or other e-commerce systems. In the graphic communications industry, the print buyer and the print shop are, to put it simply, the business partners. There are different approaches to describing these business transactions through formal specifications. The CIP4 Organization responsible for the Job Definition Format propagated an XML markup language for this: **PrintTalk.** The current version of PrintTalk is 1.3. While the details of a press product can be described with JDF, the commercial transactions are published with PrintTalk. More specifically, in a PrintTalk transaction the JDF description of a print product can then be integrated. Both XML markup languages are also closely entwined, and each has its specific tasks. This is what makes e-business models like Web-to-print possible. Therefore, customers can specify print products in certain variations, communicate business data, and place orders.

In Section 8.5 we will cover PrintTalk in more detail, but since PrintTalk is based on the general **Commerce Extensible Markup Language** (cXML), we want to briefly describe their basic functions in this section.

```
<cXML>
    <Header>
    Header Information
    </Header>

    <Request>
    Request Information
    </Request>
</cXML>
```

Figure 5.6
cXML document structure

cXML documents contain commercial transactions and are built, in principle, as shown in Figure 5.6. The root element is always a cXML element that typically contains a *Header* element and a *Request* element. The *Header* element contains information for addressing the sender and the receiver. In addition, data for authentication of the sender can be sent—in effect a password. The framework for a header is shown in Figure 5.7. The *From* and *To* elements identify the sender and the receiver of the cXML document. The *From* and the *Sender* can, but need not, be identical. Because the *From* element defines the logical sender of the document, while the *Sender* element the defines the instance which sets up the http-binding to the receiver, these can be different if the cXML document are channeled over large e-commerce networks. Where required, the password is then also entered in the *Sender* element. The *Request* element in the example is an order for an article. This also has a general administrative function: information like the delivery address, the invoice address, and information on taxation and payment, for example, can be entered in the *OrderRequestHeader.* The *ItemOut* element contains details about the items that are ordered.

Figure 5.7
cXML document

```
<cXML>
    <Header>
        <From>
         ...
        </From>

        <To>
         ...
        </To>
        <Sender>
         ...
        </Sender>
    </Header>

    <Request>
        <OrderRequest>
            <OrderRequestHeader ...>
             ...
                <ShipTo>
                 ...
                </ShipTo>
                <BillTo>
                 ...
                </BillTo>
                <Tax>
                 ...
                </Tax>
                <Payment>
                 ...
                </Payment>
            </OrderRequestHeader>
            <ItemOut ...>
                <ItemID>
                 ...
                </ItemID>
            </ItemOut>
        </OrderRequest>
    </Request>
</cXML>
```

The PrintTalk header is identical to cXML, and the *Request* element can also be found in PrintTalk. It does not have the child element *OrderRequest,* instead one *BusinessObject* element. Such an element describes the transaction, like a request for quotation, offer, etc. For some *BusinessObjects,* such as in a quote request, you can also find the child element which stores information about the desired print product.

6 Introduction to JDF

In the previous chapters you learned something about JDF. The most important things are summarized here:

- JDF is a job ticket format in which technical data can be stored, and which also supports the function of order management.
- JDF includes the functionality of PPF and PJTF.
- JDF is based on the process-resource model.
- JDF is based on XML.

In the first six sections of this chapter, JDF code is presented in greater detail. These sections represent the basis for all of the chapters thereafter. The last section will try to place more emphasis on workflows, in particular the treatment of **Interoperability Conformance Specifications.**

This introduction to JDF should provide an overview into the product descriptions and specifications of the production processes that are possible with JDF. At the end of the chapter you should be able to read and understand a typical JDF document. Of course, this chapter can not replace the JDF specification, which contains much more detail.

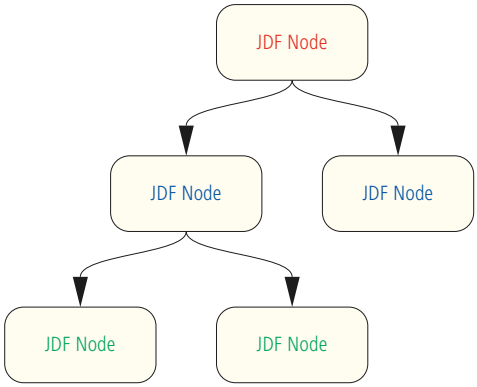
6.1 Construction of a JDF Document

For each print job in the JDF workflow, there is a minimum of one JDF document which contains the metadata for the print job. As expected for an XML document, this JDF document has a tree structure. The root element of the tree, but also each branch and every branch and every leaf, is referred to as a **JDF node.** Normally a print job consists of several JDF nodes, whereby each node describes a defined portion of the job. Each JDF node is an XML element that may itself have further subelements.

Naturally, it is not required to have exactly three hierarchical levels as in Figure 6.1. There may be more or fewer levels. Any given level can contain any required number of nodes.

Not all JDF nodes represent a process, just a few do. In general, these are the leaves of the tree. In this respect, the JDF data structure is based only partially on the process-resource model, namely, four different JDF nodes:

Figure 6.1
Nested JDF nodes



- Product node (or also product intent node)
- Process group node
- Combined processes
- Process node

Through the **product node,** a manufactured print product with its parts or subproducts is described. For example, a book is made from the cover and content, so as a result, there are three JDF product nodes; Book, Cover, and Content.

Process nodes define individual work steps for the production of a print product or one of its parts. We have already gotten to know the **combined process** of interpreting, rendering, screening, cutting, folding, gathering, stitching, trimming, and stacking. Of course there are still many other processes, and they will be presented further in the following chapters. Processes can also be grouped or combined for different reasons. This type of JDF node is then called **process group node,** as the case may be for combined processes.

As an example, RIPing in Figure 3.8 could either be a process group node or a combined process of the collected processes of interpreting, rendering, and screening (see Figure 3.10). For the exact differences between process group nodes and a combined process, we refer to section 6.4

Platemaking in Figure 3.4 is another example of a process group node, which contains the other processes of imposition, and imagesetting. It also contains yet another process group node, namely RIPing. One also sees that a process group node can contain processes as well as other process groups (which itself groups

further processes and process groups). This is similar to creating a new group of individual graphic objects with previously grouped graphic objects in drawing programs.

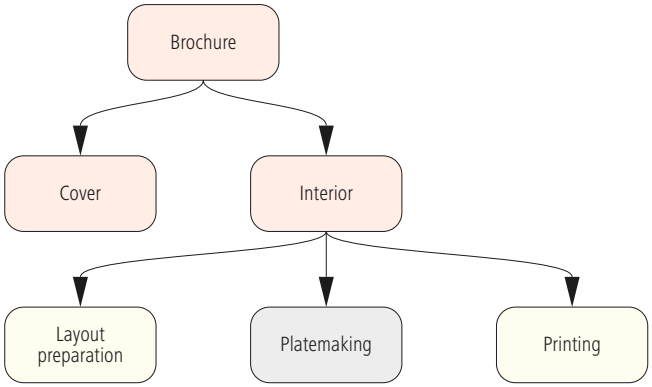


Figure 6.2
The relationship of the JDF node types Product node (red), Product Group node (gray), and Process node (yellow)

Figure 6.2 shows the relationship between the JDF node types, which is in no way complete but is only intended as an example. The red JDF nodes specify the product nodes, the gray nodes specify the process group nodes, and the yellow nodes specify the process nodes. A combined process is not available in the drawing.

It should be noted that these three node types do not necessarily appear in each JDF document. So it is quite possible that a JDF document may be made up of only one product node or out of a product node and process group node. Yet a JDF document of a complete print order cannot exclusively be made up of process nodes, since processes must always refer to products or product parts. In other words, the root element is, in this case, a JDF product node, which represents the entire job.

During the preparation of a print product, the JDF document grows as it goes through the JDF workflow. If the process nodes are missing at the beginning (because the processes are not yet clearly defined), they are gradually added. The entries are, as described in Section 3.2, generated in part through the production's preset default values or through user input during production.

The JDF nodes do not exist in isolation but require resources. A **resource,** as we have already seen in Section 3.2, is a physical artifact (paper, plates, etc.) or an electronic unit (set of parameters, file, etc.), which is input to or output from a node. Figure 6.3 shows the principle once more in a very simple manner. But, as we saw in Chapter 3, normally there is a much more complex network of resources and nodes for the description of a workflow (see Figure 3.8). Certainly, not only process nodes, but also product nodes have resources.

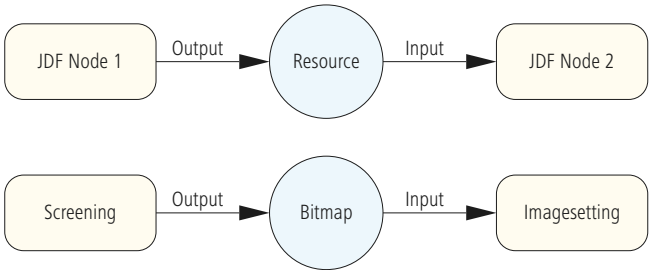
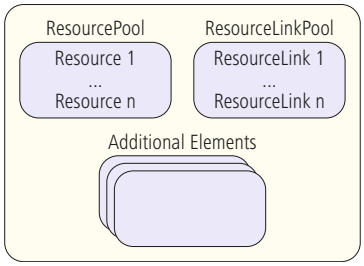


Figure 6.3
Relationships between JDF nodes and resources

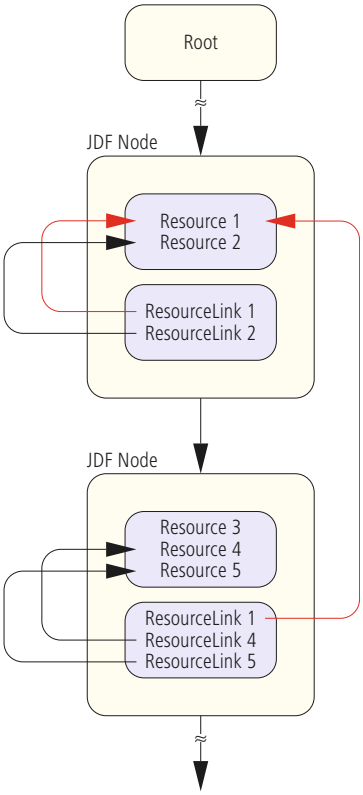
They describe, for example, details about the allocation of paper for a subproduct. One must therefore adopt the intuitive mindset of the producer-consumer model, which was quite evident in the processes. One can certainly still accept the idea that, for example, print process plates are "used up," but that a product node consumes the paper specification like a cover.

Figure 6.4
A JDF node and subelements



But how are the resources structurally interrelated with the JDF nodes? Each JDF node not only has attributes and their values, but also first and foremost subelements. Two subelements are particularly important here: the *ResourcePool* and the *ResourceLinkPool*. The *ResourcePool* contains resources for the JDF nodes; the *ResourceLinkPool* specifies which resources for the JDF nodes are used and whether, in each case, they constitute input or output resources for the nodes (Figure 6.4). Not every resource which is specified from a JDF node must exist in the *ResourcePool* of this node. In addition, resources which appear higher in the tree, that is closer to the root, and contained in a *ResourcePool* of another JDF node can also be used. Therefore, processes can share a resource without requiring that they be implemented twice (Figure 6.5).

Figure 6.5
Two JDF nodes access the same resources



It is not even mandatory that any JDF nodes have a *ResourcePool*. However, each JDF node must have a *ResourceLinkPool* associated with it because the *ResourceLinks* always refer to the JDF nodes in which the *ResourceLinkPool* is found. This means, in the event a JDF node

did not have a *ResourceLinkPool*, it would also have no relationship to any corresponding resource.

The structure of a JDF document with a root element and a JDF subnode typically looks like Figure 6.6.

Figure 6.6
Structure of a JDF with a root and subelement

6.2 Examples of JDF Nodes

Here, a JDF document will be more closely examined. However, so as not to overload this section, we have omitted many elements and attributes and present only some instructive excerpts.

Figure 6.7 shows the first line of the XML prolog, as introduced in the previous chapter. Subsequently follows the root element with the name JDF, which is marked in red. The attribute names in the code example are always colored in green.

Figure 6.7
Typical attribute for a JDF root element

```
<?xml version="1.0" encoding="UTF-8" ?>
<JDF ID="_4711" DescriptiveName="Frisch Advertising" JobID="_42"
 Status="InProgress" Type="Product" Version="1.3"
 xmlns="http://www.CIP4.org/JDFSchema_1_1">
<!-- Generated by the CIP4 C++ open source JDF Library version
 CIP4 JDF Writer Java 1.3 -->
…
</JDF>
```

The JDF nodes have multiple attributes which are briefly explained here:

- Each JDF node must have a unique ID within the document.
- The value for the optional attribute *DescriptiveName* is a human-readable identifier of the JDF node. In our example the node is called "Frisch Advertising" because the JDF document belongs to a print job that concerns an advertising brochure of the Frisch GmbH company.

- The value for the *JobID* is the identifier of the job, as the order management system has provided it.
- The attribute *Status* is the current state of the JDF node is reflected with the attribute *Status.* Besides *InProgress* there is also a range of other statuses like *Ready, Stopped, Completed, Aborted,* and others.
- The JDF node of the type *Product* behaves as a product node and thus not a process node, a process group node, or a combined process. A process group node is of the type *ProcessGroup,* and a combined process is of the type *Combined.* The type of a process node, however, identifies the process more specifically such as *Interpreting, Cutting, Folding,* or the like.
- The *Version* specifies the version of the JDF specification upon which the JDF node is constructed.
- The *xmlns* attribute defines the XML name space (see Section 5.2)

After the attributes and values, follows an XML comment that indicates how, with which computer language (*C++*), and in which program library (*CIP4 JDF Writer Java 1.3*) the document was composed.

We want to follow the Frisch Advertising brochure example further. Initially we have the *ResourcePool* and the *ResourceLinkPool* of the root element, of which Figure 6.8 only shows a part. The *CustomerInfo* is the first resource in the pool, in which the data

Figure 6.8
ResourcePool with a CustomerInfo resource

```
<ResourcePool>
    <CustomerInfo ID="_4712"= CustomerID="_0815" Class= "Parameter"
     CustomerJobName="Frisch Advertising" Status="Available" />
        <Contact ContactTypes="Customer">
            <Person FirstName="Carl" FamilyName="Cool" NamePrefix="Herr">
                <ComChannel Locator="03475/101010" ChannelType="Phone"
                 ChannelUsage="Private" ChannelTypeDetails="Landline" />
                 ...
            </Person>
            <Company OrganizationName="Frisch GmbH" />
            <Address City="Eisleben" Street="Am Kaltenbach. 3"
             Country="Deutschland" PostalCode="06295" />
        </Contact>
         ...
    </CustomerInfo>
     ...
</ResourcePool>
```

about the client is stored. Each resource requires, just like any JDF node, a unique ID. Moreover, we also see within the attribute *Class* that it handles a *Parameter* resource. Information about people is in the JDF jargon, namely from the *Class* parameters, and is not physical as one might expect. Finally the *Status* of the resource is set to *Available*, and is therefore usable. As a subelement, we see Carl Cool again, as used as the sample *Contact* in the last chapter (Figure 5.2). Of course other contact names could be noted, for example if the billing or shipping address is different.

The *ResourcePool* contains additional resources. Figure 6.9 shows the *Component* and *DeliveryIntent* resources. The *Component* resource is required for each JDF root element because it represents the end product to be produced. *Quantity* is specified as a *Class,* which is a possible subtype of physical resources. The final products are countable, in contrast to the example of the physical resource *Ink.*

Figure 6.9
Resource of the root element

```
<Component ID="_4713" Class="Quantity" ComponentType="FinalProduct"
 DescriptiveName="Frisch Advertising" Status="Unavailable" />
 ...
<DeliveryIntent Class="Intent" ID="_4714" Status="Available">
    <DropIntent>
        <DropItemIntent Amount="2000">
            <ComponentRef rRef="_4713" />
        </DropItemIntent>
    </DropIntent>
</DeliveryIntent>
```

The *DeliveryIntent* resource describes how and when the final product will be delivered to the customer and also, where appropriate, where and how. In our example not much is entered. It involves a delivery only because the *DropIntent* is specified and the *DropItemIntent* is recorded as 2000 copies.

The most significant line, however, is the reference to the *Component.* It means that the delivery or the pickup relates to the end product. The *rRef* attribute generally stands for a reference to a resource; in this case the resource is *4713,* which is the ID of the final product.

In Figure 6.10 you can see the *ResourceLinkPool,* or the information about which resources relate to the JDF nodes and whether they handle input or output resources for the node.

The relationship between the individual *ResourceLinks* and the resource itself is simple. Should a *ResourceLink* be defined to a re-

```
<ResourceLinkPool>
    <DeliveryIntentLink Usage="Input" rRef="_4714" />
    <ComponentLink Amount="2000.0" Usage="Output" rRef="_4713" />
    <ComponentLink Usage="Input" rRef="_4715" />
    <ComponentLink Usage="Input" rRef="_4716" />
</ResourceLinkPool>
```

Figure 6.10
ResourceLinkPool

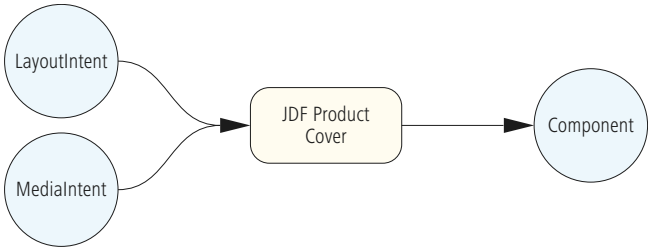
source, for which it has the name XYZ and the ID 333, then one chose XYZLink as a name for the *ResourceLink* and assigns the Reference with *rRef="333"*. The example is set up so that the *DeliveryIntent* input resource and the *Component* output resource are immediately evident.

We still have not explained two more input references (on resource 4715 and 4716) in the example. The JDF root element has two JDF subnodes in which the first is a cover and the other represents the content pages of the product. Both products are described further through *Component* resources because the cover and the content pages are required to produce the final product. The components are consequently input resources for the final product.

The JDF subnodes each have their own *ResourcePool* again and each has its own *ResourceLinkPool.* There will also be details about the subproducts: Which paper should be used? What about the color? How many pages will likely be included in the subproduct?

The *LayoutIntent* resource provides details such as the preferred size of the final format and the number of pages; the *MediaIntent Resource* describes the paper characteristics (Figure 6.11). If the project manager or CSR inputs this information into an MIS, often all of the details are not completely clarified. For example, it could be that the exact number of pages is not yet fixed or the pound weight of the paper sheet has been only approximated. Therefore these resources are allowed a certain degree of range. *Intent* refers to a "Project" and that is exactly what is described in these *Intent* resources. One finds *Intent* resources only ever in product nodes, which are also often called product intent nodes.

Figure 6.11
Separate JDF subnode "Cover" with its own ResourceLinks



It is clear that these two resources are not for the final product—instead they are, to stick with our example, defined separately for the cover and the contents. This must be so be-

cause naturally, as a general rule, both parts use different types of paper, different colors, and so on.

Figure 6.12 shows the *MediaIntent* Resource for the cover. The resource is from the *Intent* class. Please note that the two subelements are *Dimensions* and *Weight*.

```
<JDF DescriptiveName="Cover" ID="_4715" Type="Product" >
    <ResourcePool>
        <MediaIntent ID="_4717" Class="Intent" Status="Available"
        DescriptiveName="Paper for Cover">
            <Dimensions Actual="2437.7952755905512 1729.1338582677165"
             DataType="XYPairSpan"
             Preferred="2437.7952755905512 1729.1338582677165" />
            <Weight DataType="NumberSpan" Range="115 ~ 125" Preferred="120" />
        </MediaIntent>
    </ResourcePool>
<JDF>
```

Figure 6.12
MediaIntent resource with information about the intended substrate

The values of both the attributes *Actual* and *Preferred* in the *Dimensions* element are identical. This is the paper size in DTP points. Only when you divide the value by 72 and then multiply by 2.54 do you come to the more familiar measurements in centimeters (here 86 × 61 cm). *Preferred* shows the desired value of the client; *Actual* contains the intended value of the print shop. Both values here are obviously the same.

The customer's desired basis weight in Grams per Quadrameter is recorded in the *Weight* attribute. One can see that a possible range is named from 115 to 125 g/m², with the *Preferred* value at 120g/m².

An XML parser must naturally be told that it must handle a number range with the value "115 ~ 125." This happens because the data type is defined as *NumberSpan*, the element *Dimension* is the same data type, *XYPairSpan,* which means it can be defined as "a range of numbers": The paper size of Width1 × Height1 to Width2 x Height2. However, in Figure 6.12 this is not used; instead it is given only a fixed paper size.

In addition to the *ResourcePool* and the *ResourceLinkPool,* each JDF node can be comprised of another pool, which we have suppressed so far, namely the *AuditPool.* In this pool the log entries are written so that after production has ended there is a record of the creation of the print product. Not only are the modifications to the JDF document itself noted, but so is the information about the JDF process. So the various operations, for example, on the print-

ing press can be entered (plate changes, blanket washing, good print, etc.) so long as these values are reported accordingly by the press. Here is the general (but still incomplete) list of records (*Audit* elements) available in the *AuditPool:*

- Generation, changing or deletion of a node
- Process times (start, end…)
- End status (Completed, Aborted, Stopped…)
- Error
- Used or missing resources

The MIS could then make a final cost analysis of the print job with the contents of the *AuditPools*.

Figure 6.13 shows a very basic *AuditPool,* in which the entry only records where and by whom the node was created. Here …

- *AgentName* is the name of the software that created the JDF node.
- *AgentVersion* is the version of this software.
- *Author* is the person (the PC user) who created the nodes.
- *TimeStamp* is the point in time of the generation of the nodes.

The node was created on October 22, 2008, at 17:09. The *47* provides the number of seconds and the +01:00 is the time zone, which is the difference from Greenwich Mean Time.

Figure 6.13 AuditPool

```
<AuditPool>
    <Created AgentName="SuperJDF" AgentVersion="1.0" Author=" Administrator"
     ID="_4717" TimeStamp="2008-10-22T17:09:47+01:00" />
</AuditPool>
```

6.3 Partitioned Resources

For some resources, there is a problem. For example, are all of the printing plates for a print job one resource or is each plate its own resource? The situation with press sheets can be more complex. On one hand, it sometimes makes sense to see the entirety of all of the sheets from a subproduct (such as a cover or inside pages) when it comes to paper specifications. On the other hand, from

```
<Component ID="_4713" Class="Quantity" ComponentType="FinalProduct"
 DescriptiveName="Frisch Advertising" Status="Unavailable"
PartIDKeys="Condition">
    <Component Condition="Good" IsWaste="false" />
    <Component Condition="Waste" IsWaste="true" />
</Component>
```

Figure 6.14
Partitioned resource

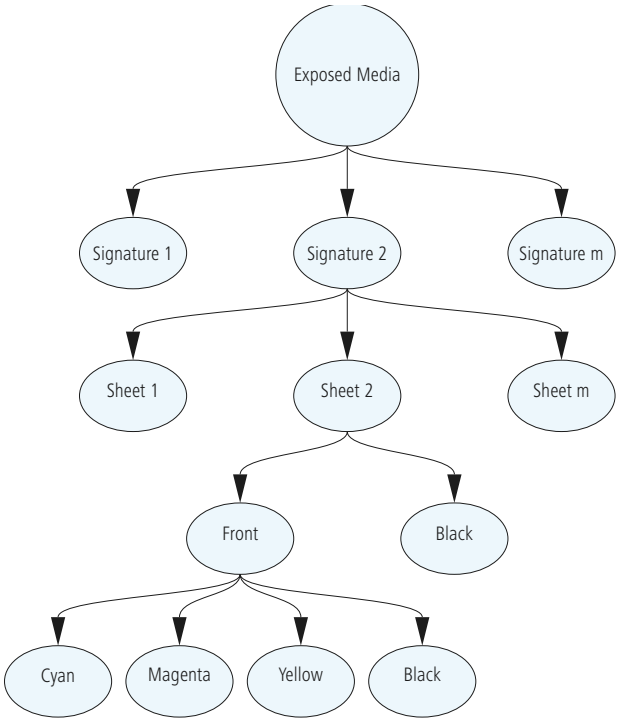
time to time single sheets must be counted, for example for determining the number of waste sheets.

Assigning each sheet as an individual JDF resource would create huge documents and clearly would be absurd. So it is better to have a resource for all of the press sheets, but a resource that when in doubt can be split into individual components (partitioning). One therefore requires portioned (more accurately, partitionable) resources.

Naturally, how these resources are structured must be specified. In Figure 6.9 the resource *Component* was presented. It should now be extended as a partitioned resource—even if the example (Figure 6.14) appears at first to seem absurd.

Figure 6.15
Component resources in the Plates example

The only new addition to the resource *Component* with *ID 4713* is the attribute *PartIDKeys.* The associated value effectively demonstrates the analysis segmentation rule—or to put it another way, a key to partitioning. In this case the press sheets are divided, like actors in Hollywood films, into two categories: good and bad. Both of the child components, and the subelements of the surrounding parent components, assume all parental attributes



In the last example the classification was only one level, but certainly in many situations one must divide it into multiple levels. Let us take plate burning as an example. The total of all of the plates for a print job, as Figure 6.15 shows, is divided into four

```
<ExposedMedia Class="Handling" ID="_4718" PartIDKeys="SignatureName SheetName
 Side Separation" Status="Unavailable">
   <ExposedMedia SignatureName="Signature1">
      <ExposedMedia SheetName=" Sheet1" Status="Unavailable">
         <ExposedMedia Side="Front">
            <ExposedMedia Separation="Cyan" />
            <ExposedMedia Separation="Magenta" />
            <ExposedMedia Separation="Yellow" />
            <ExposedMedia Separation="Black" />
         </ExposedMedia>
         <ExposedMedia Side="Back">
            <ExposedMedia Separation="Black" />
         </ExposedMedia>
      </ExposedMedia>
      ...
   </ExposedMedia>
   ...
</ExposedMedia>
```

Figure 6.16
Partitioned resource

levels. The printing plates (*ExposedMedia*) can be split into multiple signatures (*SignatureName*), each signature into more sheets (*SheetName*), each sheet into front (*Front*) and back (*Back*), and each sheet side (*Side*) into one or more separation(s).

Figure 6.16 shows just such a partitioned resource. The list *SignatureName SheetName Side Separation* now stands in the *PartIDKey,* exactly according to the layout in Figure 6.15. The sequence of the *PartIDKey* list may not be changed but must always go "from top to bottom." The rule can be more precisely stated as: The shorter the chain of predecessors to the root element, the further left the element in the list should stand. The same hierarchy must also be respected with the nesting of the subelements of the partitioned resources.

In the example in Figure 6.16 the front side from Sheet1 is CMYK and the back side is only black.

6.4 GrayBoxes and Combined Processes

There are a variety of reasons why it would be desirable to merge several processes to a new structure:

- Particularly at the beginning of production, individual processes are not yet firmly specified. It is then helpful to consider initially large production sections as a whole. Without that, the intermediate results among the processes within the sections must be stated. But in the course of the production run the intermediate results are defined.

- If a machine or software runs multiple processes in succession without anyone processing the intermediate results, it is unnecessary to specify them individually. This can, for example, be the case with an integrated digital press where the processes of interpreting, rendering, screening, digital printing, stitching, and trimming all follow each other. But the two processes of *PreviewGeneration* and *InkZoneCalculation* can also merge into a combination in prepress for offset printing (see Section 10.1).

The first case leads us to the *GrayBox* concept, the second to a combined process.

A *GrayBox* is one or more process group node(s) in which the details are not yet specified. During the course of production the details are added. When all of the processes and resources are available, the *GrayBox* is finally closed because they cannot themselves be listed like the actual processes. Typically, an MIS creates one or more *GrayBox*(es). Because, as a general rule, not all of the required production parameters are specified—for example, an MIS can determine how many plates in which format must be produced, but not the individual steps as to how they are manufactured—an MIS usually contains no exact trapping or raster information, since they are not required for estimating. The missing settings will be added either through default values in the workflow modules, through specifications in the print job, or by the data from the operator.

In the schematic in Figure 6.17 one sees that a GrayBox may certainly have input and output resources. The output resources must actually be specified. Figure 6.18 shows an example of the GrayBox which specifies offset plate production (platemaking). It comprises five (or more) input resources and one (or more) output resources. Since we also need the resources listed in the following chapters, they are briefly presented here:

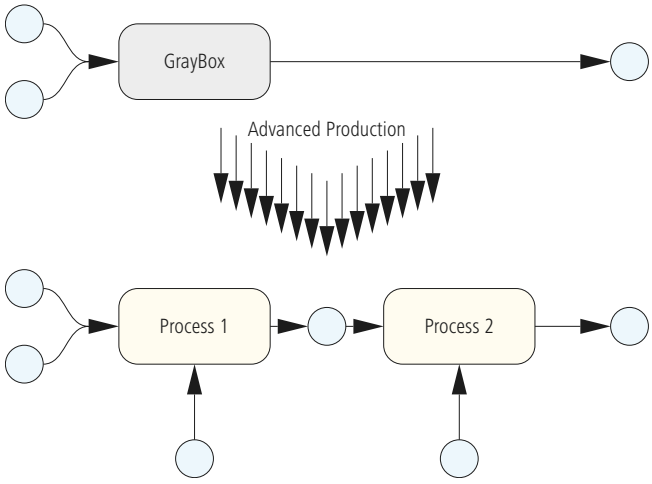


Figure 6.17
A GrayBox is resolved during the course of production

- A *RunList* is an ordered set of page descriptions, so that the content data is listed as a *RunList* (provided in Figure 6.18 with the addition of "Document"), but also the marking data that are

required for the imposition layout (in Figure 6.18 with the addition of "Marks").

- The *Layout* resource describes the layout or the layout sheet (imposition sheet).
- *ColorantControl* sets the color definitions (which color space, which colors, etc.).
- *Media* characterizes the printing plates (size, product name, etc.).
- *ExposedMedia* is, in this case, the exposed plates.

The code for platemaking is shown in Figure 6.19. Process group nodes of the type *GrayBox* are always known by the combination of the *Type* and *Types* attributes. The value of the attribute *Type* is *ProcessGroup,* and the value of *Types* is a list of processes which are included by the *GrayBox.* Note that RIPing is actually not depicted as a process, but is itself again a process group node.

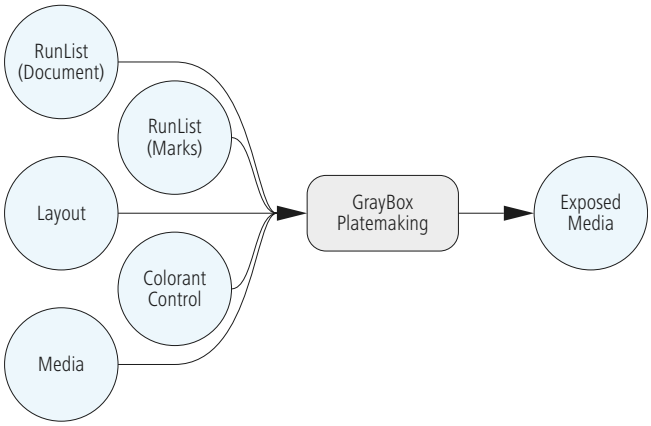


Figure 6.18 (above) and Figure 6.19 (below) Example of the GrayBox offset platemaking

There are incidentally also process group nodes which do not have the function of a *GrayBox.* The *Types* attribute is consequently missing and the *Type* attribute as the value *Process Group.* This type of process group nodes contains processes as JDF nodes.

Processes not only can be grouped but also be combined. The result of such a combined process is itself an additional process. It contains a list of process names

```
<JDF Type="ProcessGroup" Types="Imposition RIPing ImageSetting"
 DescriptiveName="GB PlateMaking" ...>
...
    <ResourceLinkPool>
        <RunListLink ProcessUsage="Document" Usage="Input" ... />
        <RunListLink ProcessUsage="Marks" Usage="Input" ... />
        <LayoutLink Usage="Input" ... />
        <ColorantControlLink Usage="Input" ... />
        <MediaLink Usage="Input" ... />
        <ExposedMediaLink Usage="Output" ... />
</JDF>
```

```
<JDF Type="Combined" Types="Imposition Interpreting Rendering"...>
    <ResourcePool>
        <RunList ID="_100"... />
        <Layout ID="_200""... />
        <InterpretingParams ID="_300" ... />
        <RenderingParams ID="_400" ... />
        ...
    </ResourcePool>
    <ResourceLinkPool>
        <RunListLink CombinedProcessIndex="0" Usage="Input" rRef="_100" />
        <LayoutLink CombinedProcessIndex="0" Usage="Input" rRef="_200" />
        <InterpretingParamsLink CombinedProcessIndex="1" Usage="Input"
         rRef="_300" />
        <RenderingParamsLink CombinedProcessIndex="2" Usage="Input"
         rRef="_400" />
        ...
    </ResourceLinkPool>
</JDF>
```

Figure 6.20
Example of the combined process offset platemaking

and the required resources. Figure 6.20 shows that in this case the value of the attribute *Type* is *Combined,* while the processes are listed out within the value of *Types*—similar to a *GrayBox*.

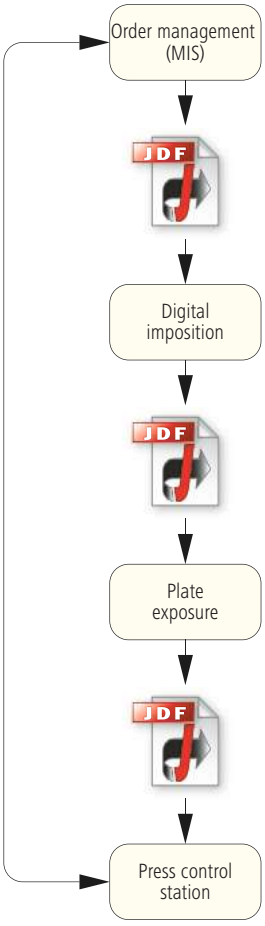
A combined process is similar in concept to a process group. The difference is that a combined process is from a single device, and a process group is generally executed from multiple devices. For this reason, resources, which only describe the intermediate results of the process, must not necessarily be listed in a combined process.

CombinedProcessIndex is actually the only new attribute in Figure 6.20. The values indicate which processes specified under *Type* stand in relation with the resources. It is counted from zero, which means the first process in the list, namely *Imposition,* has the index of 0, the second (*Interpreting*) has the Index 1, and the third (*Rendering*) has the Index of 2. Consequently the resources *RunList* and *Layout* are input resources for *Imposition, InterpretingParams* are input resources for *Interpreting,* and the *RenderingParams* are input resources for *Rendering.*

6.5 JDF Workflow Architecture

Now that we have seen in principle how a JDF document is built, we will examine how these documents get to the workflow modules.

The simplest possibility is the sequential relaying of information, which is sketched out in Figure 6.21. This very simple model is not realistic, or a best-case scenario only in very simple production environments, in which two or three neighboring process exchange



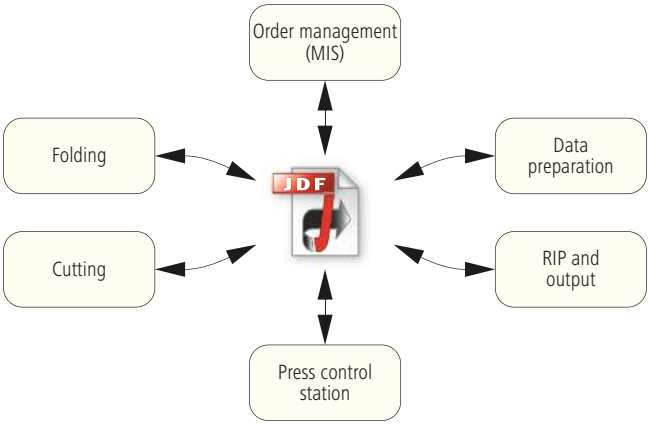
Figures 6.21 and 6.22
Sequential transmission (above) central administration (below)

JDF information. Example: assembly software (*Stripping*) creates a JDF document for imposition (*Imposition*).

A more realistic model is the central JDF data repository to which the different workflow modules have access, as shown in Figure 6.22. This data store can either be a database or a simple folder structure on a file server. Naturally the individual workflow engines are not completely free to access the JDF data, only under the administration of a special higher-level software which controls the consistency of the data. In contrast to the sequential situation, the data is stored on a central server which is advantageous in terms of data backup. Figure 6.22 suggests that JDF administration is a piece of the production software. But there is also the MIS-centric, central data handling that Figure 6.23 exhibits. Incidentally it is not necessarily the case that only one JDF document is generated per order. For, instead of always filling this document with current data, each of the workflow modules can also generate new versions of the JDF document. So one could, in the event of a failure, at least in theory, turn back to an earlier state.

However, to prevent a possible misunderstanding: Naturally, in a JDF environment not all of the data for production of print products is stored somewhere in JDF format. In fact, for example, an MIS will continue to operate with its own database tables and data sets, in which all of the data is also stored, which are not mapped in the JDF (such as cost of equipment and operations). JDF/JMF is also not the only external interface. The same sample also applies for prepress, press and postpress.

The CIP4 Organization suggests an additional hierarchical model, which is given in Figure 6.24 as an example. This model works; however, it is less concerned with the data store and more concerned about how the functions may cover the JDF software. The following terminology applies:



- An *Agent* can generate a JDF document, generate and modify a JDF node. Typically an order management system performs the *Agent* role.

- A *Controller* has the task of relaying JDF documents and also JDF messages to the devices. Conversely, it should also report messages back to the Agent from the devices.
- A *Device* interprets the JDF nodes and initiates the corresponding actions in the connected machine. The Device-Machine communication is not based on JDF/JMF, but is instead vendor-specific. In IT jargon, one would also call a *Device* a "JDF device driver."
- A *Machine* carries out one or more processes without themselves understanding or producing JDF or JMF.

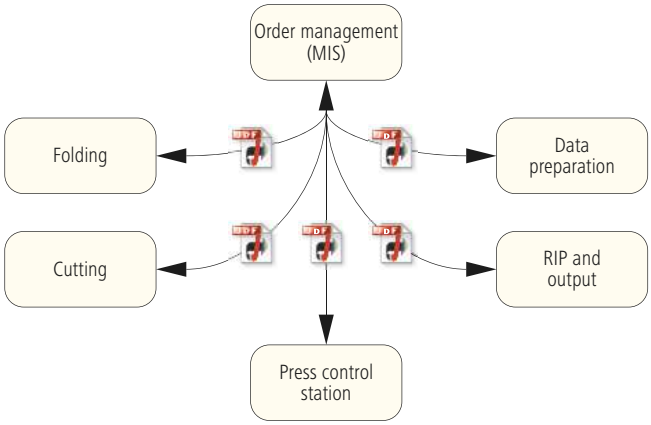


Figure 6.23
The MIS is responsible for JDF communication

The division is not always absolutely clear. Many controllers and devices may produce and modify JDF nodes; therefore, they have *Agent* capabilities. Also, a controller will often be an *Agent* and a *Controller* simultaneously, as is the case with some order management systems. This JDF architecture then has three levels, which means that the *Agent* (the MIS) is also the universal *Controller.* The division into three Controllers in Figure 6.24 is only an example; it could be more or less one *Controller.* Finally Controllers can work recursively; a *Controller* can handle one or more sub-controllers.

At the same time, one *Controller* can route data using the following methods:

- One *Controller* sends the complete JDF job to all devices in turn.
- The *Controller* determines by JMF which device can handle which processes and sends the respective part of the JDF job to the devices.
- As in the last point, only the properties of the device are "hardwired" in the *Controller;* i.e., they are strictly defined.

In the last two bullet points, parts of the JDF nodes must be separated out. We will discuss the technology for this purpose in the next section.

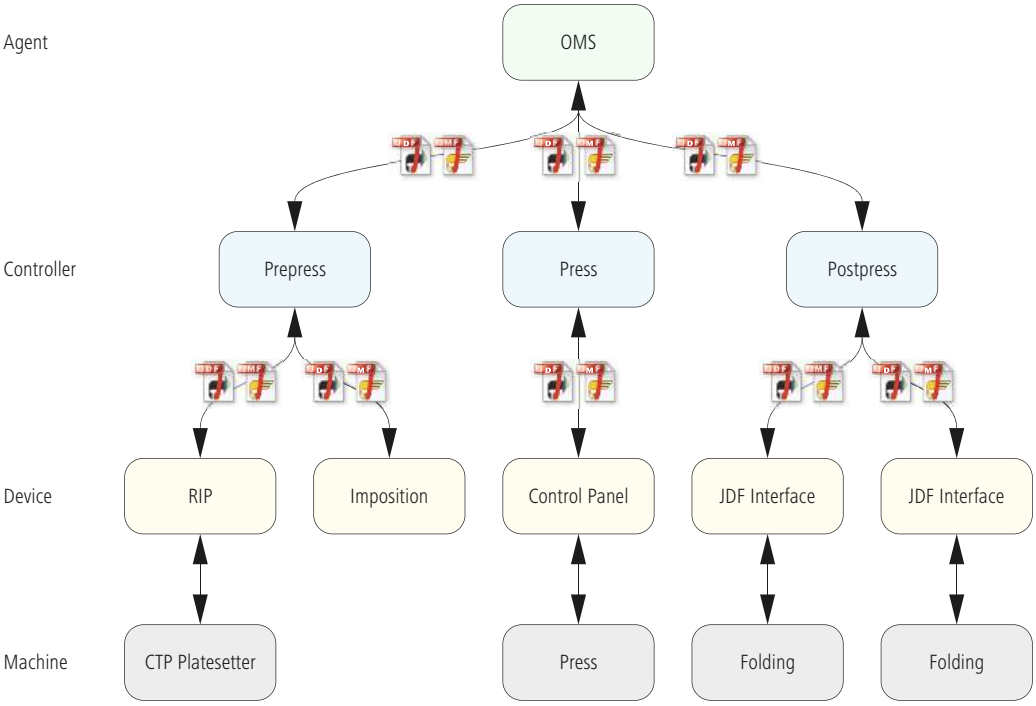


Figure 6.24
Hierarchically organized JDF/JMF communication

The combinations of devices and equipment were called "job ticket processors" or also "workflow engines" in the last chapter.

The models presented here may all, at first, seem quite complex and very static, and it brings up the question of which processes may be automated so that the user profits from such a JDF installation. In fact, JDF devices can automatically initiate processes on

Figure 6.25
NodeInfo resource

```
<ResourcePool>
    <NodeInfo FirstStart="2008-08-01T00:00:0+01:00"
     LastStart="2008-08-02T08:08:00+01:00"
     LastEnd="2008-08-02T15:00:00+01:00"... />
    ...
</ResourcePool>
```

a machine. This requires the following conditions:

- A process described in JDF is executable by the machine.
- The status of the process is *Ready* or *Waiting*.
- All input resources have the status *Available*.

- The current time is within the interval of defined production time.

The production time interval for a process node is recorded in its own resource with the name *NodeInfo.* Generally, information about execution, responsibility for the process, and consequences of a timeout are stored in this resource. In the example in Figure 6.25 the process should begin no earlier than on 1/8/2008 at midnight and at the latest on 2/8/2008 at 8:00 a.m. The process must end at 3:00 p.m. at the latest on the same day.

6.6 Separating and Merging

In workflow management systems, things can run in parallel. Accordingly, JDF information must be available on multiple JDF Devices. Moreover, it may be more practical on the grounds of data management to only provide the information to a JDF device that it needs and not the entire JDF document. In both cases one must also duplicate sections of the JDF code and send them to other devices. Since these devices can append JDF information to the copied files and, of course, the entire production log should be available again in a single JDF document, it must also be possible to merge the copied and modified JDF data back into the original document. In order for this whole process to work, some things need to be considered. Above all, multiple places in the JDF code may not be changed, and so there must be agreement on who has permission to read (*read only*) and who may read and write (*read-write*). In short, the consistency of the data must be ensured.

In JDF jargon the procedure is called "Spawn and Merge." Instead of "spawning" we talk about "separating," but it is actually more of a copy process.

In Figure 6.26 the situation is presented graphically. On top (1) is the document depicted originally, from which (2) a JDF node is separated and a new document is generated. One clearly sees in (3) how the separated node is changed independently from the original document, and (4) finally how it is reunited with the original JDF tree. The detached node can certainly generate even more child nodes.

Naturally, resources can also be separated as portioned resources. It must be noted in the original document which resources were separated. The *Spawn* and *Merge* operations will be logged in the *AuditPool.*

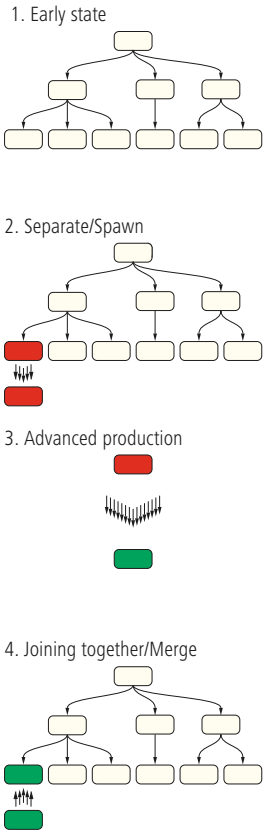


Figure 6.26
"Spawn" and "Merge" processes separate or merge, respectively

```
<JDF DescriptiveName="Total product" JobID="_001"...>
    <JDF ...DescriptiveName="Cover" ID="_002 " JobPartID="_003"
     Status="Waiting" Type="Product">
     ...
    </JDF>
</JDF>
```

Figure 6.27
Original file before separation

In the following, we want to show an example of a separation and merge.

In Figure 6.27 we see a JDF job before the separation process. In the next step the subproduct "Cover" is separated (in order to produce it, for example, at another site). In Figure 6.28 you can see how the master file was changed after the separation: in the *AuditPool* the *NewSpawnID* was registered which can also be found in the separated file. Moreover, the URL of the separated file is noted under the *jRef ID* of the node which was separated, and finally also in the *rRefsROCopied* attribute (R_1 to R_9) which were separated. The *RO* in the attribute name means that the resource is *Read-Only,* i.e., the process that handles the separated part may only be read and not changed. It is further noted that the separated pieces of the JDF tree—in this case the JDF node "Cover"—are still found in the original file. Figure 6.29 shows a section of the file with the separated JDF data. All of the IDs like *JobID* and *JobPartID* are not altered; only the *SpawnID* is additionally defined

Figure 6.28 (topmost)
Original file after separation
Figure 6.29 (top)
Separated JDF file

```
<JDF DescriptiveName="Total product" JobID="_001"...>
    <AuditPool>
        <Spawned NewSpawnID="_004" Status="Waiting"
         TimeStamp="2008-10-25T09:38:09+02:00"
         URL=file://file:/C:/GetrennteDatei.jdf
         jRef="002" rRefsROCopied="R1 R2 R3 R4 R5 R6 R7 R8 R9" />
    </AuditPool>
    <JDF DescriptiveName="Cover" ID="_002" JobPartID="_003"
     Status="Spawned" Type="Product">
    ...
    </JDF>
</JDF>
```

```
<JDF ... DescriptiveName="Total product" ID="_002" JobID="_001"
JobPartID="_003"
     SpawnID="_004" Status="Waiting" Type="Product">
    </JDF>
    ...
    <AncestorPool>
        <Ancestor FileName="file:/C:/Originaldatei.jdf" JobID="_001"... />
         ...
        </Ancestor>
    </AncestorPool>
</JDF>
```

```
<JDF DescriptiveName="Total product" JobID="_001"...>
    <AuditPool>
...
        <Spawned NewSpawnID="_004" Status="Waiting"
         TimeStamp="2008-10-25T09:38:09+02:00" URL="file://file:/C:/Test.jdf"
         jRef="002" rRefsROCopied="R1 R2 R3 R4 R5 R6 R7 R8 R9 />
        <Merged ... MergeID="_004" TimeStamp="2008-10-25T09:41:01+02:00"
         URL="file://file:/C:/GetrennteDatei.jdf" jRef="002" />
    </AuditPool>
    ...
    <JDF DescriptiveName="Cover" ID="_002" JobPartID="_003"
     Status="Waiting" Type="Product"...>
...
    </JDF>
</JDF>
```

Figure 6.30
JDF file after the merge

(the same number as the attribute *NewSpawnID* before it). Additional information about the original JDF job is stored under the *Ancestor* element, here the file name and the storage location as well as the JobID. This information is required in order to correctly merge the data back again later. In this state the main file and the separated file could be developed independently of each other, at least when the latter has write rights. Finally, at some point the two JDF trees will be merged. The result is seen in Figure 6.30, which is actually the same as before the separation process. Only the entries in the *AuditPool* describe that in the meantime a section was separated and was later remerged.

The entire process is recursive, which means that separated JDF elements can be separated again.

6.7 Interoperability Conformance Specifications (ICS)

Interfaces are always problematic. If, for example, an advertising agency gives an order to a print shop for a print product, they may not know exactly how the content data should be correctly prepared. Conversely, the printers do not know what the agency does not know, and so misunderstandings and frustrations arise. Even though they both speak the same language, the communication does not work satisfactorily. A solution to this dilemma in the output of content data is the agreement on PDF standards like PDF/X [23] or PDF/X-Plus [19], which limits the huge functionality of the Portable Document Format.

These interface problems also exist in a similar way with the JDF workflow: when an *Agent,* a *Controller,* or a *Device* to any degree generates JDF structure, then the expectation is that the recipient

can do something with it. Conversely, the recipient requires certain JDF data that they possibly did not receive. Here is where the **Interoperability Conformance Specifications** (ICS) provide a remedy. They limit the vast functionality that JDF/JMF offers for certain software components.

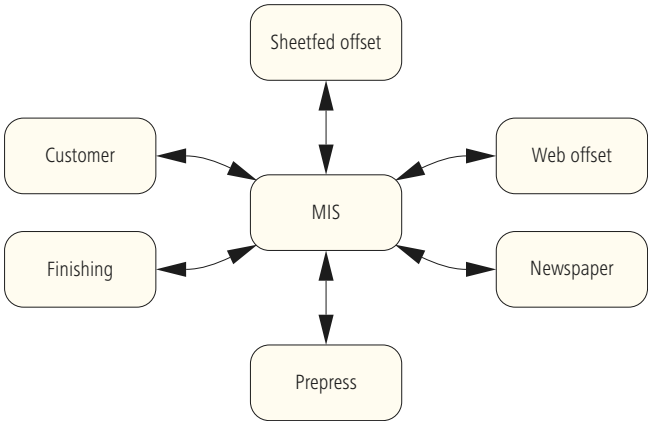
With this type of problem there are always two roles: one generates information, and the other consumes it, or at least it tries to. In this respect the concepts of *Agents, Controllers,* and *Devices* are not furthered and two new concepts are employed:

- The *Manager* sends predominantly JDF documents to a Worker (*Controller* or *Devices*). Moreover, it can optionally send JMF messages to them. Its second priority is to read JDF documents and JMF messages which are sent to it by a *Controller* or *Devices*.

- The *Worker* receives JDF documents from the Manager (*Agent* or *Controller*), perhaps also JMF messages. In addition, it can (again, as a secondary priority) send back JDF/JMF information.

One may argue about the word choice of *Manager* and *Worker,* but we will follow this language usage of the ICS papers. Note that both *Manager* as well as *Worker* can read and write JDF data and JMF messages. For example, a *Manager* sends a JDF process to a *Worker,* who performs it, and during and after the process sends back information about used, consumed, and generated resources. If one wants to emphasize the role of the writing of JDF/JMF (or also of modifications), one speaks of producers (*Producer*) and, with the role of the reader, of a consumer (*Consumer*). *Manager* and *Worker* are therefore sometimes producers and sometimes consumers.

Figure 6.31
MIS interfaces described in ICS papers



The ICS papers describe different interfaces, which are built in part on one another. The different MIS interfaces, which are defined through ICS papers and based on JDF version 1.3, are shown in Figure 6.31. The entire collection of ICS papers and their hierarchy are shown in Figure 6.32. *The MIS to Finishing ICS,* for example, is founded on the *Base ICS,* the *JMF ICS,* and the *MIS ICS,* while the *Of-*

fice Digital Printing ICS is based only on the *Base ICS*.

Moreover, the individual interfaces are classified yet again in different levels, which are also hierarchically structured. We will introduce some of the ICS papers in detail in the following chapters. Now we want to illustrate the *Base ICS* in more detail. There are three levels defined, namely Level 0, Level 1, and Level 2, whereby the functionality of Level 1 is comprised of Level 0 and correspondingly the Level 2 functionality is comprised of Level 1. In the remaining ICS interface papers the presupposed levels of the *Base ICS* can be specified. An interface satisfies only the *Base ICS* Level 0 if the *Manager* can transmit JDF documents via a hot folder and the *Worker* can read them. Level 1 is reached, conversely, when the *Worker* can also send JDF documents to the *Manager* and the *Manager* can read them. For Level 2 there are exactly two further requirements to fulfill:

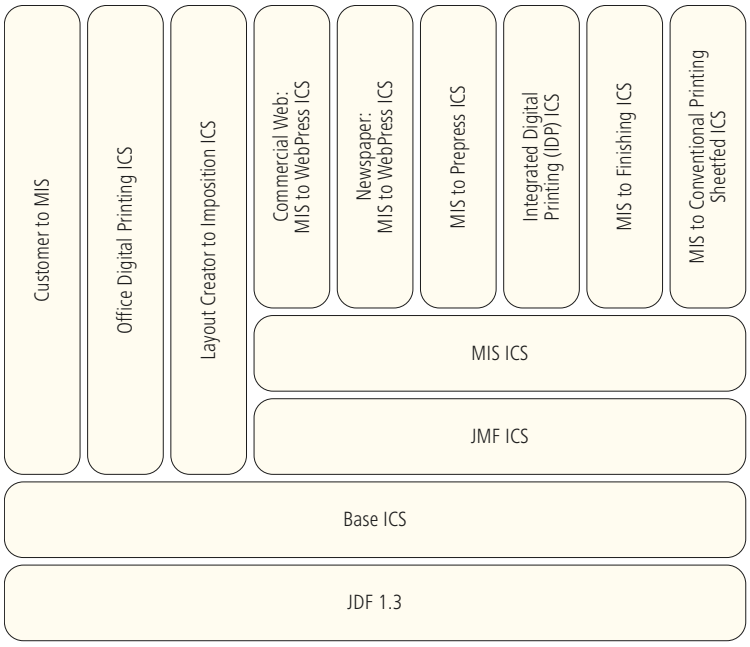


Figure 6.32
In part, ICS Papers build upon each other

- In general, it can be useful with a WMS implementation to transmit multiple data simultaneously such as JDF documents, JMF messages, PDF print data, preflight profile, ICC color profile, and preview images. These may come packaged together such as in a MIME package (MIME stands for **Multipurpose Mail Internet Extensions).** As with email attachments, multiple different data may likewise be sent together. At Level 2 the *Manager* must now collect the MIME packets and the *Workers* can interpret them (Figure 6.33).

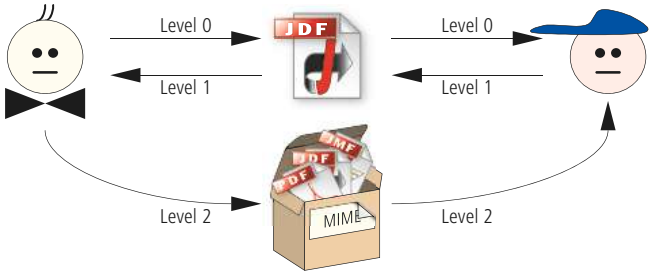


Figure 6.33
Different levels in Base ICS

- As a rule, files are referenced within a PDF document, for example a PDF. That means, for example, the server in the local network and the path in the file system where the files can be found are provided. In Figure 6.34 a *RunList* (page sequence) is defined which consists only of a PDF document (*LayoutElement*) whose file name and memory location is known over a path name (*FileSpec*). The value "0 ~ - 1" of the *Pages* attribute states that all of the pages of this file should be used. Level 2 additionally demands only that the *Manager* makes these files available over HTTP and the *Worker* can also pick them up over HTTP.

Figure 6.34
Reference to Content Data via the URL

```
<ResourcePool>
    <RunList ID="_4719„ Pages="0~-1" Class="Parameter" Status="Available">
        <LayoutElement>
            <FileSpec URL=file://Workflowserver/Pagefiles/Frisch-Advertising.pdf
/>
        </LayoutElement>
        ...
    </RunList>
    ...
</ResourcePool>
```

Besides these levels, several technical things are illustrated in the *Base ICS,* so for example, an *ID* may consist of any number of characters. Interestingly (but only slightly) are the presence of requirements which are generated from JDF nodes and their attributes and subelements. The configuration states, for example, that the *Manager* writes the *JobID* in the root element and the *Worker* must read it or who must enter which *Audit* elements are mandatory into the *AuditPool,* which are optional, and so on.

In summary, it can be said that the ICS papers are not just tools for developers of JDF software; instead they also form the most important basis for integration and tests between multiple JDF components. In the case of an error in JDF communication, one should first examine whether all of those involved in communications are adhering to the ICS rules.

Exercise:

Download the *JDF Editor* as well as some sample files from CIP4.org. Open the files and examine the structure of the JDF nodes and the resources.

Rename the *.jdf files to *.xml and open them with a browser. Analyze them directly as XML code.

Separate a JDF subnode with the help of the Editor and then subsequently proceed to put it back together again. Analyze the intermediate status with a browser.

7 Job Messaging Format (JMF)

In the last chapter we saw how data can be transmitted between different Agents, Controllers, and JDF devices by means of JDF data. In the simplest case, JDF data is written from a JDF producer and placed into the hot folder of a JDF consumer which scans for new data at certain time intervals. This unidirectional interface for file transfer is naturally quite static and allows very little dynamic interaction between the workflow stakeholders, which, for example, is important for data collection, job tracking, job changes, centralized error detection, analysis of material consumption, collection of the overall efficiency of machines, or for system management (especially at startup). **Job Messaging Format,** together with JDF was introduced for this purpose. JMF messaging is not implemented in each JDF workflow, yet we will see what additional value it can bring. However, for an MIS for example, Level 2 ICS is required for certain JMF messages.

A general understanding of the JMF technology is important in order to understand how a JDF/JMF workflow functions. Despite that, one needs to be clear about the fact that as a user one has less to do. Because while JDF data are mainly made available as files and thus are easily available for analysis and possible troubleshooting, JMF data are mostly sent over the HTTP protocol and therefore are not physically delivered as a file. Naturally, one can use an **HTTP sniffer,** programs that find HTTP packets and make them readable—although with the quantity of information that is usually sent over the network it tends not to be very fun. In this respect we will also place more emphasis on JDF than on JMF and leave out some JMF subtleties.

7.1 Communication Models

However, before we step into the structure of the JMF message protocols, we wish to discuss, somewhat more generally, different communication models that are common between software modules in the graphic arts industry:

- File transfer via hot folders (or manually)
- Data transfer over printing protocols
- Database interfaces
- Interprocess communication

- Web services
- Message protocols

File transfer over hot folders has a large advantage, namely the simple administration by users, especially system administrators. But they also pose some disadvantages:

- When a hot folder chain is broken (for example because a computer is not running) it does not give a fail message.
- The data producer receives no acknowledgment that its data was successfully read and could be interpreted.
- In the event of an error with the job execution, there are no system monitors which can detect the causes.
- The communication is slow and is therefore suitable for either shop floor data collection or for bi-directional handshaking methods with which two or more modules must coordinate themselves.

In many cases hot folders were and still are used by RIPs in which the print data is placed. Typically these folders are then configured in a way that certain RIP settings (resolution, line screen, etc.) are associated. With many different RIP settings, many hot folders must be set up, which can be confusing. In a JDF environment this is generally not a problem since the metadata is there and so the variable settings are entered for the RIP—the hot folder simply presents itself as the communication channel.

Instead of file transfer, data is also distributed to the RIP system by print protocol. The system may have made special settings for a job via the print menu, but for JDF this method is not suitable. Also the cross-vendor transmission of JDF via databases is impractical because no access methods exist. The same thing goes for the implementation of interprocess communication, which indeed is made by individual manufacturers but is not open source. These techniques are found, for example, with PJTF communication between job ticket processors.

Web services define the message exchange between applications in the Web on the basis of XML [45]. So by example, reservation systems at travel agencies communicate with hotels via Web services in order to make inquiries and bookings. Naturally, individual modules of a workflow management system can communicate in exactly the same way.

Web services require messaging protocols such as SOAP (originally **Simple Object Access Protocol**), for example, from W3C. This protocol is in fact employed in workflow management systems. SOAP services the exchange of XML messages and therefore can also transport JMF. HTTP is mainly used for sending.

```
<JMF TimeStamp= "2008-07-25T12:32:48+02:00"...>
 ...
</JMF>
```

Figure 7.1
JMF message

JMF messages are small XML documents which have a root element with the name of JMF, as you can see in Figure 7.1. Then, within the subelements of the JMF root element the messages are actually coded. The message itself is usually sent over HTTP or the HTTPS protocol, which normally transports HTML websites over the Internet. With this method a two-way communication can be built quickly. Alternatively some JMF messages are also placed into hot folders as files which the receiver must read. It is the same method that is also widespread with JDF (Figure 7.2).

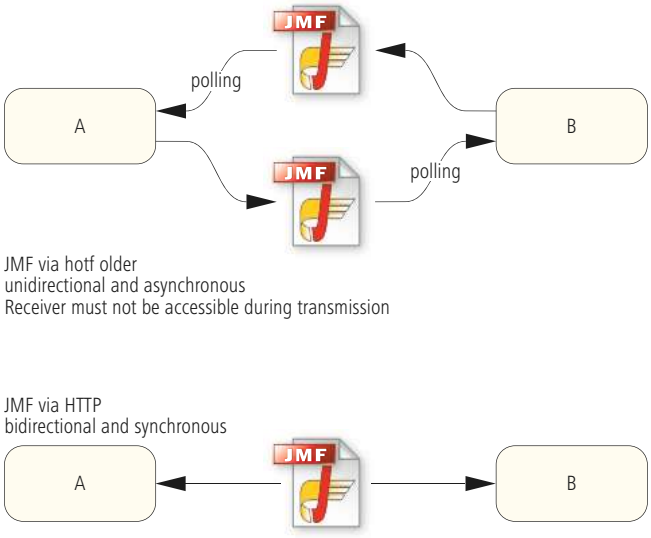


Figure 7.2
Different types of JMF communication

7.2 JMF Families

What is the most important content that can be transported with JDF now? Here are some examples:

- Initializing of devices
- Device and job status
- Control of queues and their entries
- Completion of milestones

Most of all, the dissemination of the status of devices and print orders is of central interest, because the user can draw a variety of benefits from this. Job tracking is even possible with JDF, as is final cost analysis. Also, central monitoring of the devices is easier with

the help of such communication. In practice it is often not particularly difficult to generate and to send messages regarding device condition and production progress. It is much more problematic to subsequently filter the mass of data for meaningful aspects. For this reason in part, final cost analysis does not commonly occur on the basis of JMF messages, but instead with the help of the *AuditPools,* which we presented in Section 6.2. These *Audit* entries from devices are typically entered in their own JDF, which, after the completion of all processes, are provided back to the MIS or to a central JDF workflow server. The *Manager* must take over filtering of the JMF messages and the writing of the *Audit* from the *Worker.*

JMF messages can be divided into six categories, the so-called JMF family:

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

A JMF Query is directed to a JMF controller or a JMF device in order to receive some specific information. It does not change, as opposed to a Command, the status of the addressee. The device or controller usually responds to the Query with a Response. In Figure 7.3 an example of a Query and a corresponding Response can be seen as a graphic. Each controller and each device has its own ID, and the question and the answer have their own identifiers. So the answer can unambiguously be associated to the earlier Query. In this case the controller with the *ID = 4711* wishes to find out which devices the addressed controller, with the *ID = 42,* monitors. Such a question could be important, for example in a plug-and-play method for JDF modules (although this is still far from reality). The device then answers that it administers a RIP and a CTP imager.

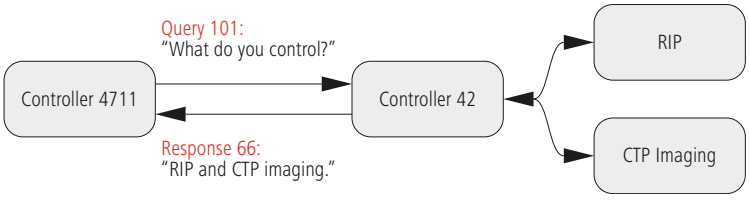


Figure 7.3
JMF query and response

```
<JMF TimeStamp="..." SenderID="_4711">
    <Query Type="KnownDevices" ID="_101"/>
</JMF>

<JMF TimeStamp="..." SenderID="_42">
    <Response Type="KnownJDFServices" ID="_66" refID="_101"/>
        <DeviceList>
            <DeviceInfo DeviceStatus="Idle">
                <Device DeviceID="Rip" />
            </DeviceInfo>
            <DeviceInfo DeviceStatus="Running">
                <Device DeviceID="CtP"/>
            </DeviceInfo>
        </DeviceList>
    <Response>
</JMF>
```

Figure 7.4
JMF Response to a Query

The corresponding JMF code is listed in Figure 7.4. It is perhaps obvious that the ID of the receiver is not included, neither with the Query nor with the answer in XML code. *ID = 42* is not placed in the Query and the *ID = 4711* is not in the Response. So how can the message reach the correct addressees at all? The solution to this puzzle is that the recipient is addressed over HTTP protocol (also via hot folders). Figuratively speaking, the JMF message is in an envelope with the corresponding address label stuck on it and then this is sent via HTTP. One can also see in the code that the Response is clearly set within the *refID* attribute directed to the previously sent request.

Commands can change things. This also goes for JMF commands. They allow entries in the queue to be erased through commands while a Query only can request the status of a queue. But before we go any further here, what is a queue? Of course it does not refer to the line at the supermarket; it is, instead, a data structure that acts as a buffer and as a rule works on the principle of *First In/First Out* (first come, first served). This could also be a data buffer from a RIP, for example, in which the RIPing data will be temporarily stored or as temporary memory for JDF jobs for a controller or a device.

With respect to queues, there are two types of JMF commands: those that either relate to individual entries within a queue or with the complete queue itself. This is how the command *HoldQueue* stops the entire queue and no jobs are processed further, while the command *HoldQueueEntry* puts only a single job in the queue on ice. Although if commands seem unimportant or even useless with respect to queues or their entries, one should realize that without them, workflows would be limited. For example, only when a de-

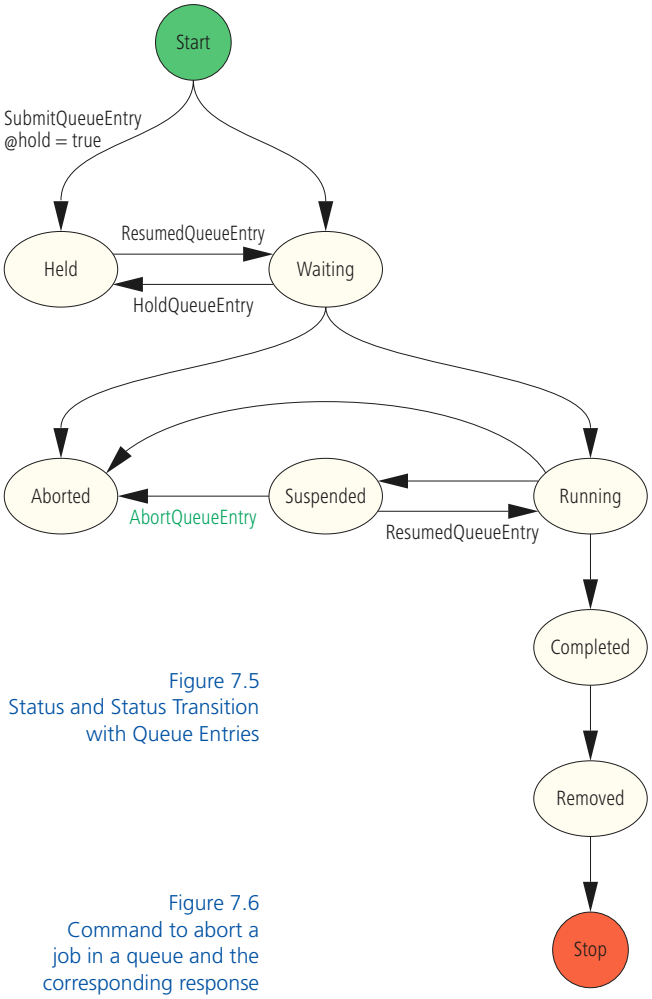


Figure 7.5
Status and Status Transition with Queue Entries

vice can communicate to a controller that the job no longer has to wait but is currently actively running can this information in turn be used to update the user with a snapshot of the production status.

Queue entries have different statuses like:

- *Waiting*
- *Held*
- *Running*
- *Suspended*
- *Aborted*
- *Removed*

These statuses can pertain not only to the queue but also to devices, as is seen in Figure 7.4 (*DeviceStatus*).

Figure 7.5 shows the status and some status transitions (in order not to overload the drawing, we have left some out). Moreover, in some cases it shows JMF

Figure 7.6
Command to abort a job in a queue and the corresponding response

```
<JMF SenderID="Controller-1" TimeStamp="2008-08-13T10:05:32+01:00"
Version="1.3"...>
<Command ID="C1" Type="AbortQueueEntry">
       <QueueEntryDef QueueEntryID="job-4711" />
    </Command>
</JMF>

<JMF SenderID="Device-1" TimeStamp="2008-08-13T10:05:33+01:00"
 Version="1.3"...>
    <Response ID="R1" Type="AbortQueueEntry" refID="C1">
        <Queue DeviceID="Device-1" Status="Running">
            <QueueEntry JobID="job-4711" QueueEntryID="job-4711"
             Status="Aborted" />
        </Queue>
    </Response>
</JMF>
```

commands which cause the status transitions. Notice the difference between *Held* and *Suspended.* In both cases jobs will be sent for further processing, only once from the *Waiting* status and the other time out of the *Running* status.

The Command to abort a job as well as the answer (*Response*) to this Command is seen in the code example in Figure 7.6. Controller 1 sends an *AbortQueueEntry* command (C1) in reference to the queue entry *job-4711*. The recipient of this *Command* is Device-1; its Response deals with Command *C1*. The aborted job is actually still an entry in the queue, only now with the *Status* set equal to *Aborted.*

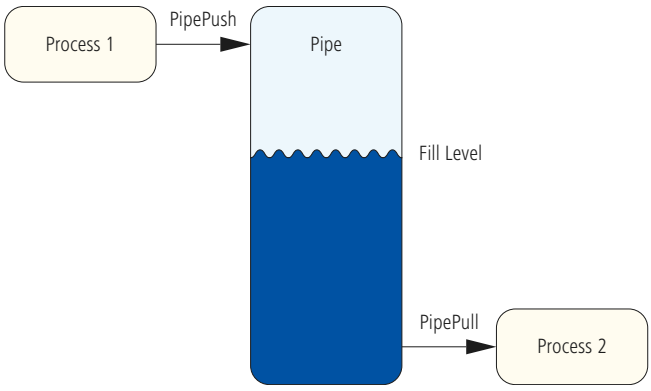
JDF jobs can, as we know, be transported as files via hot folders. Another possibility consists mostly of a controller or a device communicating how and where these JDF files can be picked up, at which point there are two possibilities given for "how": either via normal file transfer or over HTTP. The code example in Figure 7.7 shows the actual situation for file transfer.

Figure 7.7
Individual Queue Entry

```
<Command ID="C2" Type="SubmitQueueEntry">
    <QueueSubmissionParams URL="File://HdM/Server1/Jobs/job1.jdf" />
</Command>
```

Other examples of commands are found with the pipe data structure, which we briefly introduced in Section 2.6. With overlapping production, a process P1 is put into a pipe; after a time, a second process P2 brings them back out. What is special about this pipe structure is that a partitioned resource does not need to be completely brought into the pipe before it can be read from there. In the example mentioned in Chapter 2, P1 was the plate imaging process and P2 was the print process: while the job plates are being imaged, the first signatures can already be printed. The actions are likewise commands (Figure 7.8) to put a resource in a pipe (*PipePush*) and respectively to remove the resource (*PipePull*).

Figure 7.8
Storage with pipes and JMF commands for filling and emptying



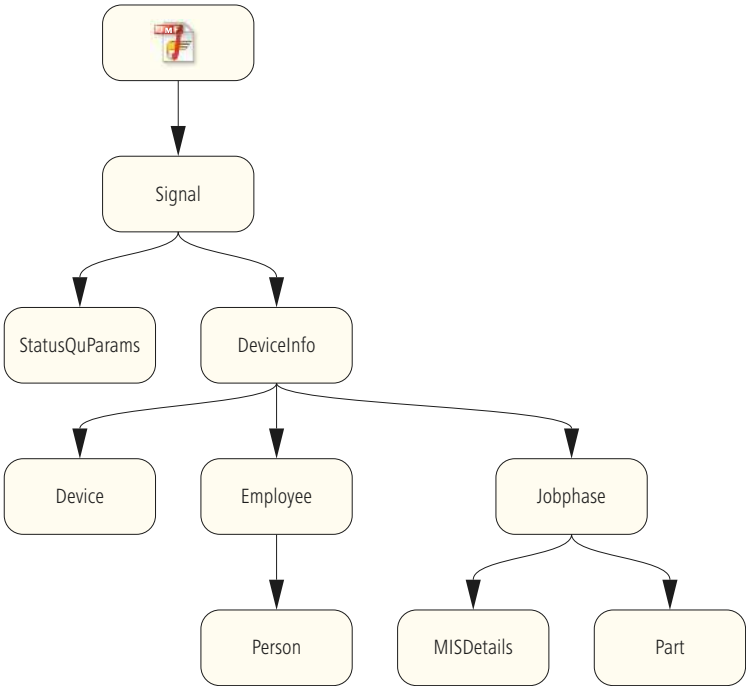
Finally let's discuss the **Resource Messages.** These are requests or commands concerned with JDF resources. For example material, device, or job resources are queried (*Re-*

sourceQuery) or are modified (*ResourceCommand*). With the help of commands, JDF resources can be sent to a device in either a completely new version or existing resources are updated. For example, if a plate is exposed then the corresponding controller sends a command so the status of the JDF resource or a section thereof is set to *Available.*

With an acknowledgment (*Acknowledge*) in JMF, one understands a temporally staggered (asynchronous) response to a Query or a comment. Because responses, especially commands, may require a while until they are executed, the receiver first sends a Response that the Query or the Command has been received and later a confirmation that the Command was carried out. In real life the terminology is reversed: an order will be carried out immediately and answered later.

Signals are unidirectional messages which are commonly concerned with status changes of machines or jobs. A controller can "subscribe" to those signals in different manners and ways, which means they give notice that they wish to receive a certain type of signal. In Figure 7.9 the structure of a signal providing information about the production status of a printing press is sketched out. The JMF messages consist of a signal, which in turn contains two subelements. The *StatusQuParams* element defines the job to which the signal refers. The *DeviceInfo* element gives information about the device such as instantaneous *Status* and also details about production counters and the like. Further information about the device, the operator of the device (as long as he or she has logged in), and the job is contained In the *Device, Employee,* and *JobPhase* subelements. In Figure 7.10 the shortened code for this signal is shown. In the *JobPhase* element one sees that the job was

Figure 7.9
JMF Signal



```
<JMF SenderID="_4004"...>
    <Signal ID="S1" Type="Status">
        <StatusQuParams JobID="_0001" JobPartID="_002"... />
        <DeviceInfo DeviceStatus="Idle" TotalProductionCounter="13862.0">
            <Device DeviceID="_4004" Status="Available"... />
            <Employee ID="_111" PersonalID="_013" Roles="Operator"
             Status="Available">
                <Person FamilyName=”Cool” FirstName=“Carl“... />
            </Employee>
            <JobPhase JobID="_001" JobPartID="_002"
             StartTime="2008-07-22T13:18:12+02:00" Status="Completed"
             TotalAmount="1002.0" Waste="71.0">
                <MISDetails CostType="Chargeable" WorkType="Original" />
                <Part SheetName="Blaetter" SignatureName="SIG1" />
            </JobPhase>
        </DeviceInfo>
    </Signal>
</JMF>
```

Figure 7.10
Code example for a JMF signal

completed (the *Status* is *Completed*) whereupon a total of 1,002 sheets were printed, of which 71 were spoilage sheets. Moreover, it shows that the customer must pay for the order (the *CostType* attribute is set to *Chargeable*) is noted in the *MISDetails* element and that the original order was carried out without any changes (the *WorkType* is *Original*). The *DeviceStatus* attribute in the *DeviceInfo* element stands at *Idle,* which is “inactive”; other statuses are: *Down, Setup, Running, Cleanup, Stopped,* or *Unknown*. One additional attribute in the same element (not included in Figure 7.10) is called *StatusDetails.* With this, more precise information about the status of machines, and particularly printing presses, can be made. The list in Figure 7.11 contains a selection of options. Exactly such a list details how exact and comprehensive JDF and JMF were specified.

Figure 7.11
Values for StatusDetails attribute

BlanketChange
BlanketWash
BreakDown
CleaningInkFountain
ControlDeferred
CoverOpen
CylinderWash
DampeningRollerWash
DoorOpen
Failure
FormChange
Good
InkRollerWash
Maintenance
OutputAreaFull
PlateWash
Repair
ShutDown
SizeChange
SleeveChange
WaitForApproval
WarmingUp
WashUp
Waste

The last member of the JMF family is the registry. At this point the idea is to prompt the sending of commands to third parties in certain circumstances. By way of example, an MIS can request a prepress WMS to send CIP3 data to a particular printing press. This command hierarchy requires fixed communication channels (*persistent channels*) which it is first required to set up. Since these are also important for signals, we want to illustrate them more accurately, but only in the next section; in that respect, we will not go further with registrations in particular.

In the end it should be mentioned that the JMF protocol can be extended with private content. Through the inclusion of name spaces, new elements and attributes can be created. Similar to JDF, it increases flexibility at the expense of compatibility.

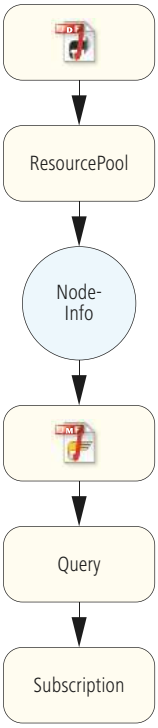


Figure 7.12
A subscription is requested with the help of NodeInfo

7.3 JMF ICS

The *Interoperability Conformance Specification* for JMF [12] specifies conditions for JDF/JMF modules in two stages (levels):

Level 1

a) JDF data is distributed via hot folders.

b) Managers can build fixed communication channels only such that the corresponding queries or registrations can be placed in the JDF document.

Level 2

a) Managers can build fixed communication channels through JMF messages directly via HTTP.

b) JDF data can be distributed via JMF.

c) There is a bidirectional JMF message exchange between Manager and Worker. Specifically, appointments can be realized with respect to the queue via JMF.

The element *NodeInfo* was already presented in Section 6.5, using the example of a schedule manager for the execution of a process. In fact, with the help of this resource, fixed communication channels (*persistent channels*) can be built as Level 1b calls for. These fixed communication channels are typically required for a controller to subscribe to signals from JMF-capable devices. So too can an MIS determine that it will be informed regularly about the status and about production details of a printing press. The term "subscribe" is thereby the magic word, because in fact it is a *Subscription* element created in the *NodeInfo* resource. The parent/child relationships are shown in Figure 7.12. A fixed communication can also be defined within a question (*Query*), which remains valid so long as it is either capped or the corresponding JDF nodes have been processed.

Of course, requests for these fixed communication channels are also packed in HTTP and sent directly as JMF nodes. This is then the requirement which must be fulfilled in Level 2a. The requirements 2b and 2c have already been presented. In particular we saw in Figure 7.7 how, using the command of type *SubmitQueueEntry,* JDF jobs can be presented to devices over the HTTP protocol.

Finally we want to summarize one more time, independent of the ICS Levels, the different possibilities for sending messages in a JDF/JMF environment whereby complexity of the implementation increases from top to bottom.

- No JMF messages
- JMF signals
- JMF queries/JMF responses
- JMF commands
- JDF transmissions via HTTP

Exercise:

Download a freeware HTTP sniffer from the Internet and install the software. Look for and read the HTTP packets in the sniffer which are generated through the use of a browser.

8 Order Management Systems

Some vendors of order management systems:

Alphagraf
Hiflex
Megalith
Optimus
Pagina
Printplus
Prosecco

Order management systems have several names in the graphic arts industry: they're also called management information systems (MIS), industry-specific software solutions, enterprise resource planning (ERP), production planning and control systems, or simple costing/spreadsheet programs. Naturally there are differences between the terms—but in practice they are often used interchangeably, and we will hereafter use both OMS and MIS. We also will not pursue definitions of terms; instead we will list the main points of what software can include:

- Estimating and final costing
- Commercial order processing (from quotation to invoicing)
- Customer data management
- Materials management
- Interface with production
- Handoff of production details
- Production planning/scheduling (deadline and resource planning)
- Activity recording/data collection
- Order tracking (job tracking)
- Interface with finance and payroll bookkeeping
- Interface with customers (business and Internet portals)
- Interface with suppliers (paper supplier)

Order management systems are typically installed in print shops, but not all of the functionalities listed above are essential and are mostly depicted as optional modules only, such as production planning. Not all of them are essential for the JDF workflow, such as interaction with finance and payroll accounting. However, estimating, commercial order processing, and customer and material management are the most important functions of an order management system. In the following sections we will highlight some of these concepts in more detail and also analyze the dependencies between them. For the JDF workflow, the interface from the order management system to production is of special importance,

so we will investigate it further in the following section. The interface can also be realized in the opposite direction, from production to the order management system, whereby the production floor data such as production status and dates can be returned as feedback. Communication between print product buyers and print product manufacturers using PrintTalk is the subject in Section 8.4.

8.1 Basic Functionality of an Order Management System

The subject of this book is not how a quotation is created and how it functions. This is only a question of which input data a quotation requires; the actual price determination of a print product remains a magic crystal ball. The input data for estimating are important insofar as these, at least in part, can be passed on to production via JDF.

When estimating a printed product, as a rule, information from the following areas is required:

- Business data
- Equipment components of the product
- Production machinery
- Material data
- Production processes

In many cases the required production processes are, indeed, concluded from product definition and production machinery depending not only on prepress, but also on the properties of the data, which a printer receives from the customer.

In the estimating program there is a very clear distinction between user and administration. While the CSR carries out the quotation, system administrators maintain the root data. In particular master data maintenance consists of:

- Setting up of cost centers
- Entry of costs and duration of individual work processes (place cost calculation)
- Definition of product types
- Entry of parameters for the print shop machinery
- Setup of customer and material databases

- User management
- Form processing

The user can, if the master data is set up correctly, quickly and easily assemble the required data for the estimate out of predefined data sets. Next, the customer is usually selected from the database, which then populates the contact person, their contact data, delivery and billing address, and even creditworthiness into the job. During the product definition, the product type (book, brochure, business card, etc.) are selected and the binding type, page size, color, final size, bleed, and run length are defined. With complex products, product components and their values must be provided separately. For instance, for a perfect-bound book the color of the cover can be different from the color of the contents. Under the "production machines" rubric the user can select a print shop's printing press stored as a set of parameters. With that, the plate size, the gripper edge if applicable, the paper edge, perfecting (duplexing) options, and the minimum and maximum sheet size with respect to the roller width and the cutoff length are configured. Subsequently, depending on the product, one or more imposition templates are configured, which can usually be selected from a template catalog. With the selection of the printing material, the exact price can be loaded out of the corresponding database but can still be overridden by the user. Of course the production machinery for a print job can be chosen by the operator or also automatically assigned by the system.

Finally the individual production steps must yet be defined, which are not set up as part of the production standards by the system administrator. These may be additional proofs requested by the customer, special expenses for production of the actual print data, or extra equipment generated in the processing of the additional steps.

From all of this information the order management system calculates the production costs and outputs a sale price. Thereafter, the offer letter (quotation) can be printed for postal mailing or faxed or sent via email. After order entry, an order confirmation is sent to the customer, and job jacket data is generated for production—the latter, at least, by a JDF workflow.

The final costing is used for monitoring how the job was completed and also to identify order-related discrepancies. Through final costing, not only can estimating be optimized, but ways to improve production efficiencies can be discovered as well. Usually the term

"final costing" is somewhat limited, namely to the determination of the actual working time of the utilized devices and the actual material usage for the order. The estimating basis, namely the hourly rates and the cost centers and therefore the costs of the individual operations, are seldom captured from the final costing. A final costing also assumes a production floor data collection system. These can happen via time sheets, which the company's employees fill out manually and must be subsequently transferred into bookkeeping software. Instead of filling out paper time sheets, the details are often entered by employees in shop floor data collection terminals. There have long been proprietary systems, most of them supplementary components of order management systems, for this purpose. Yet the actual goal of a JDF/JMF environment is to generate the shop floor production data almost automatically. With JMF there is a chance that various devices can send their data to a central evaluation point, for example the MIS, using a standardized protocol. With these dynamically generated data, the cost analysis is less expensive and at the same time more exact, but also device failure and the production status of jobs can be tracked.

But in a JDF world not all of the data are automatically generated. So commonly there are computers with appropriate software directly next to machines that are not connected or next to handwork areas into which time is recorded and consumed or generated resources can be entered. These data are then sent to the MIS or another instance for analysis using JDF or JMF.

8.2 JDF Interfaces to Production

The JDF interface between order management system and production is actually one of the most important JDF interfaces. By now the implementation of this interface is not only widespread but also open in the sense that order management systems and production software from different vendors can communicate via JDF and JMF. This is not the case for all JDF interfaces, which we will see in Chapter 9. For the print service providers Y and Z in sections 2.2 and 2.3, the JDF interfaces between MIS and prepress have already been sketched out.

There are essentially two models as to how JDF data can be passed to production. Either the order management system hands off the JDF data off to a central JDF production controller, which subsequently distributes the information to a controller and devices in the prepress, press, and postpress departments, or it takes control of the transmission to the different devices itself (see Figure 8.1).

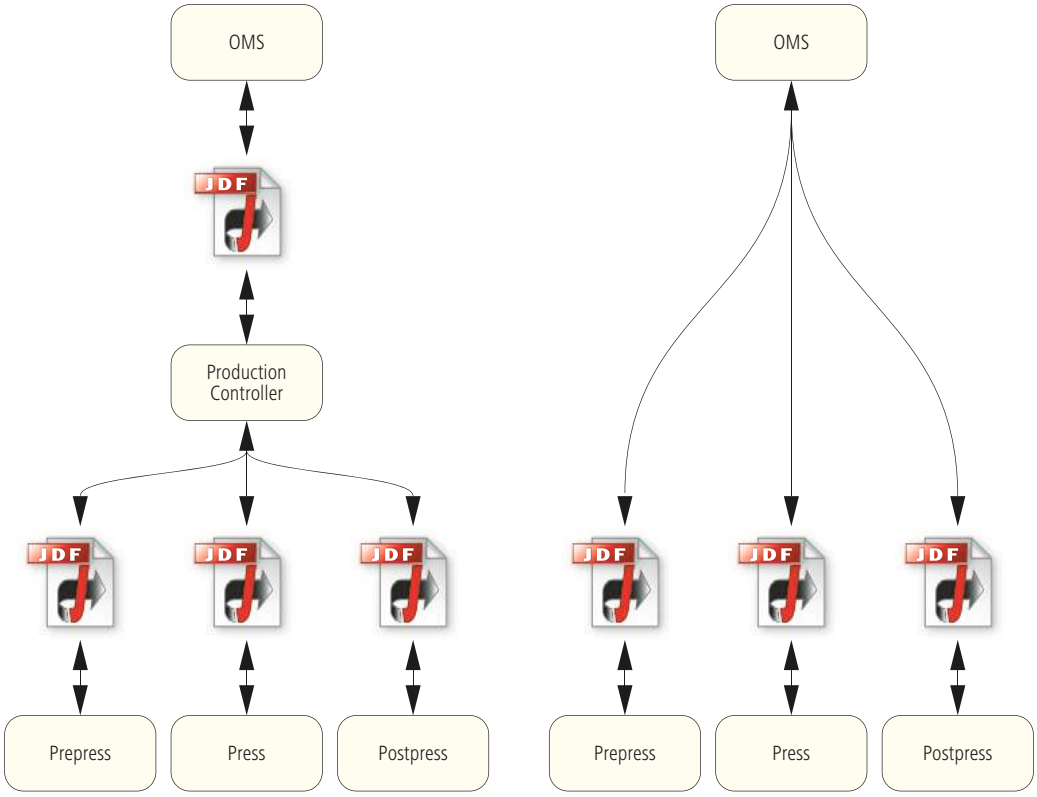


Figure 8.1
JDF between MIS and production

This seemingly small difference, however, has large implications. Because in the configuration on the left the workflow control lies in production while in the graphic on the right the MIS takes control of this assignment. As the MIS carries out the workflow control, it must be able to receive the technical details from production, such as color zone preset values, the cutting positions, or the fold sequence and forward them to the appropriate device/controller in the pressroom or in post-processing. Therefore, the request to the MIS goes far from the classic commercial challenge. Both of the configuration types shown in Figure 8.1 have been implemented by manufacturers.

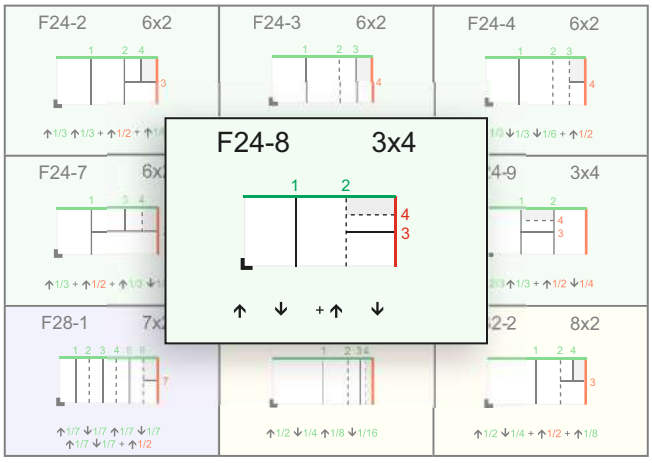


Figure 8.2
F24-8 of the JDF folding catalog

In this section we will presuppose the existence of a cen-

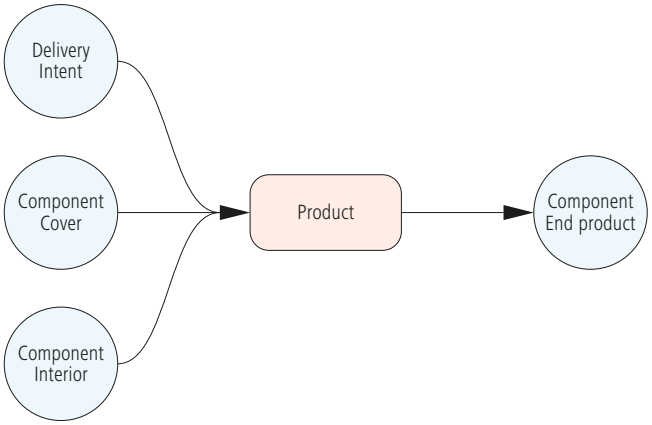


Figure 8.3
JDF root element of a brochure

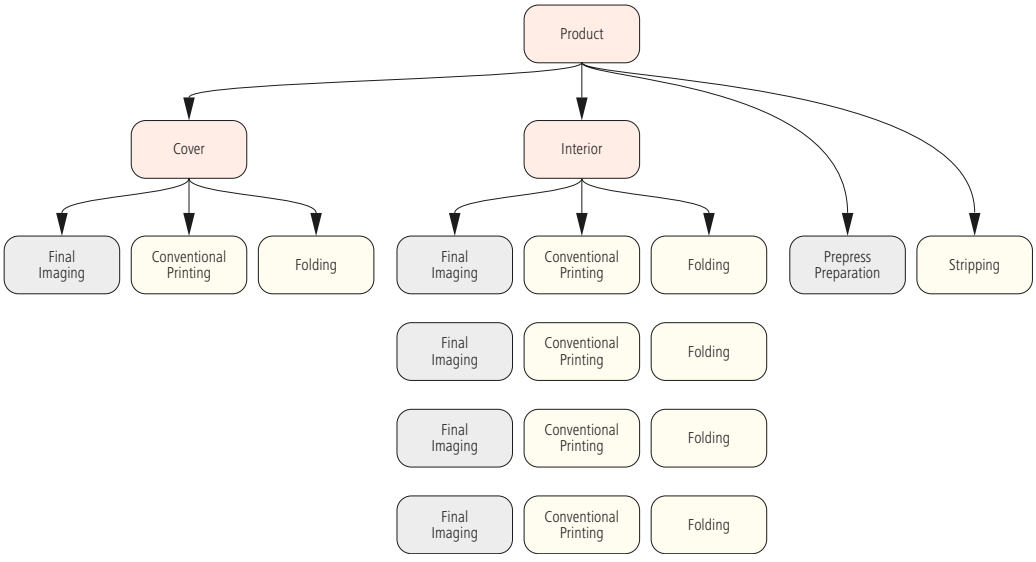
tral JDF production controller in our example. The transfer of JDF data for post-processing machines is so far rarely in use, and therefore the interface is treated separately in Chapter 11.

The interface is not abstract; instead it is best described by an example. The example product is a 96-page, 4-color, perfect-bound booklet with a final size of 12 × 12 cm. Both the covers as well as the interior pages are printed on 43 × 61 cm sheets. The signatures for the inside are built in accordance with F24-8 fold type catalog with a 3 × 4 imposition matrix so that there are a total of 4 signatures (4 × 24 = 96 pages). The cover is printed duplex, 6-up on the sheet (Figure 8.2).

The component view of the JDF file, which is typically generated from an order management system, is shown in Figure 8.3. The product node has two *Component* resources as input, namely the cover and the inside. The end product is the output resource from the type *Component*.

Figure 8.4
Tree structure of a JDF node

From the perspective of product nodes, *GrayBoxes,* and process nodes we get a tree structure which is reproduced in Figure 8.4.



The red rectangles are product nodes, the yellow are the process nodes, and the gray rectangles are the *GrayBoxes*—thus product group nodes. Since four different signatures must be produced for the inside, the *GrayBox* of the *FinalImaging* category and the processes *ConventionalPrinting* and *Folding* are accordingly also created four times.

Some of the processes and the process groups listed here were already mentioned in the introduction to JDF. We would like to characterize them at this point only briefly before analyzing them in detail.

- **PrepressPreparation:** *GrayBox* which includes all of the work steps from the processing of the content data through to imposition.
- **Stripping.** Page assembly.
- **FinalImaging:** This *GrayBox* includes *Imposition,* the *GrayBox RIPing,* the *PreviewGeneration,* and *ImageSetting*.
- **ConventionalPrinting:** Conventional print with physical press form. Here: offset print.
- **Folding.**

The process group node *PrepressPreparation* is structured very simply, as one can see in Figure 8.5. The input resource *RunList* defines the press pages which will be supplied by the customer. At this point, after receipt of the order, only the page count of the press product is known but not the name and save location of the content data (see Figure 8.6). In contrast, the output resource *RunList* represents the pages prepared for production which in particular are normalized, color space transformed, and trapped.

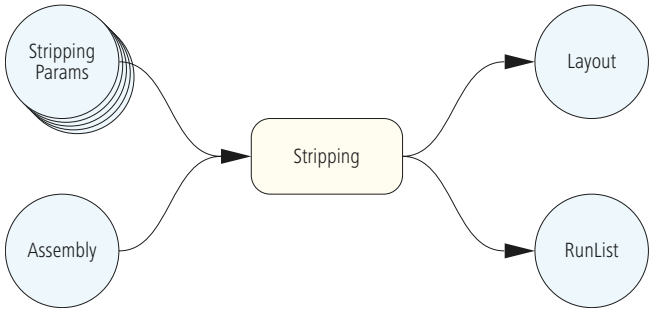
The *Stripping* process with both its input and output resources is shown in Figure 8.7. For each Signature there is a *Stripping-*

Figure 8.5 (top)
Resources of the GrayBox PrePressPreparation

Figure 8.6 (middle)
RunList Resources as Input for PrePressPreparation

Figure 8.7 (bottom)
JDF Process imposition sheet creation


```
<RunList Class="Parameter" ID="_001"
NPage="106" Status="Available" />
```



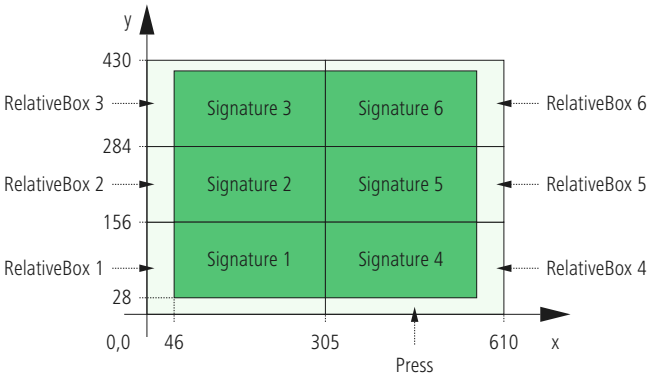


Figure 8.8 (above) StrippinParams contain the fold sheet position

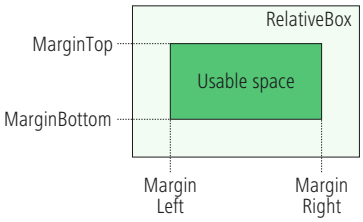
Figure 8.9 (below) StrippingParams contain information for stripping (imposition)

Params resource, and therefore there are a total of five in our example. These mainly contain the position of the folding sheets on the press sheet (see Figure 8.8) and a link to the imposition template of the folding sheet. In Figure 8.9 the positions of the six folded sheets are listed individually. However, the information is not very easy to understand. First, the press sheet is separated into six rectangles, whose

```
<StrippingParams Class="Parameter" ID="_200" PartIDKeys="SignatureName
 SheetName" PartUsage="Explicit" Status="Available">
   <StrippingParams SignatureName="SIG001">
      <StrippingParams SheetName="Umschlag" WorkStyle="WorkAndBack">
         <BinderySignatureRef rRef="_201" />
         <Position MarginBottom="79.370" MarginLeft="130.39"
          MarginRight="0.0" MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.0 0.0 0.5 0.362" />
         <Position MarginBottom="0.0" MarginLeft="130.39" MarginRight="0.0"
          MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.0 0.362 0.5 0.660" />
         <Position MarginBottom="0.0" MarginLeft="130.393" MarginRight="0.0"
          MarginTop="51.023" Orientation="Rotate0"
          RelativeBox="0.0 0.660 0.5 1.0" />
         <Position MarginBottom="79.37" MarginLeft="0.0"
          MarginRight="130.393" MarginTop="0.0" Orientation="Rotate0"
          RelativeBox="0.5 0.0 1.0 0.3627" />
         <Position MarginBottom="0.0" MarginLeft="0.0"
          MarginRight="130.393" MarginTop="0.0"
          Orientation="Rotate0" RelativeBox="0.5 0.362 1.0 0.660" />
         <Position MarginBottom="0.0" MarginLeft="0.0" MarginRight="130.393"
          MarginTop="51.023" Orientation="Rotate0"
          RelativeBox="0.5 0.660 1.0 1.0" />
         <StripCellParams BleedFace="7.086" BleedFoot="7.086"
          BleedHead="7.086" BleedSpine="0.0" Spine="0.0" TrimFace="11.338"
          TrimFoot="11.338" TrimHead="11.338" TrimSize="340.157 340.157" />
         <MediaRef rRef="_202">
            <Part SheetName="Cover" SignatureName="SIG001" />
         </MediaRef>
         <MediaRef rRef="_203">
            <Part SheetName="Cover" SignatureName="SIG001" />
         </MediaRef>
         <DeviceRef rRef="_204" />
      </StrippingParams>
   </StrippingParams>
...
</StrippingParams>
```

dimensions are given in the attribute *RelativeBox*. The four values there all have values between 0 and 1 because they describe the parts of the press sheet whereby the first two numbers define the (x, y) values of the left bottom corner of the rectangle and the next pair of (x, y) values define the top right corner. The *RelativeBox* with the values of 0.0 0.0 0.5 0.3627 defines the rectangle below left, and the press sheet below right also begins with a zero and begins in the X direction at the half sheet and in the Y direction and ends at 36.279% of the press sheet. Since the press sheet is 430mm high, the first *RelativeBox* ends accordingly in the Y direction at 156mm (0.36279×430mm). In this RelativeBox lies folding sheet 1 (see Figure 8.8). The other rectangles are calculated in the same manner. In our example, the respective values lay within each *RelativeBox*. The values in the *RelativeBox* are determined through the values *MarginBottom, MarginLeft, MarginRight,* and *MarginTop*. The chart in Figure 8.10 again shows this situation. The *StrippingParams* resource is actually a good example for a portioned resource (at least more valuable than Figure 6.14), which is split according to the keys *PartIDKeys, SignatureName,* and *SheetName*.

Figure 8.10
Position of the tiles in the RelativeBox



There are also a few other things that should be explained in the JDF code from Figure 8.9. In the element *StripCellParams* the values for the first cuts as well as the final size of the partial product are defined. Both references to the *Media* resources are respective of the substrate and the plate to be used. Finally the *BinderySignature* resources can be mentioned, which is referred to in the fourth row and can be seen explicitly in Figure 8.11. It obviously contains the imposition template. In this case it is simply an entry entered into the folding catalog. Alternatively, the imposition template can be directly entered into the subelement (*SignatureCell*) of *BinderySignature*. The *BinderySignature* resource therefore corresponds to a conventional folding pattern which can be applied to multiple folding sheets and are defined independently of the size of the pages with respect to the "up" and the print sheet.

Figure 8.11
The BinderySignature resource contains the imposition template

```
<BinderySignature Class="Parameter" DescriptiveName="F04-01_ui_2x1"
 FoldCatalog="F4-1" ID="_201" NumberUp="2 1" Status="Available" />
```

The *Assembly* resource, however, is mentioned only once in the JDF document. In it, with the resources introduced in JDF Version 1.2, all of the information generally exists whether the sheets are to be collated or gathered.

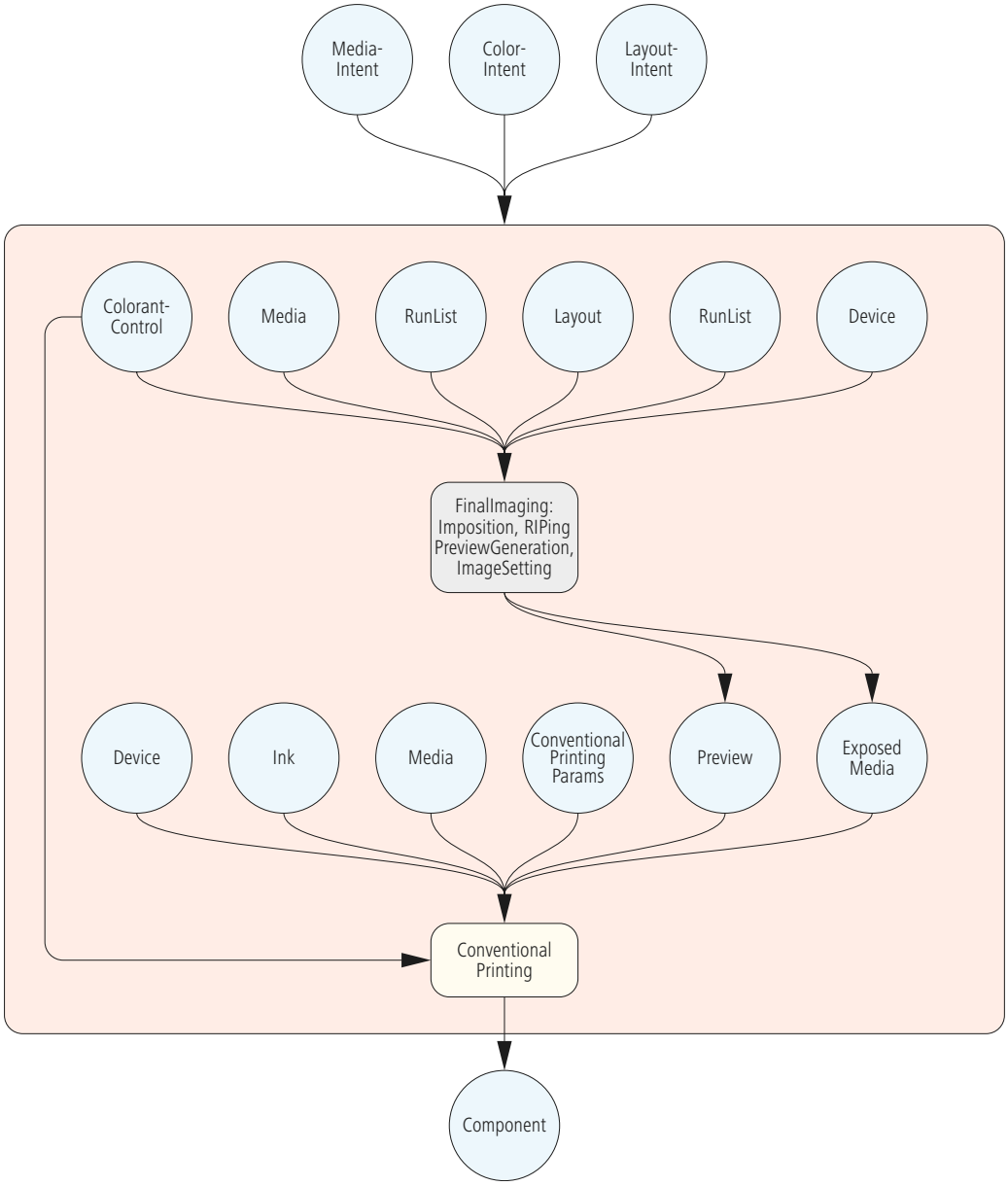


Figure 8.12
Product node "Cover" with its substructure and the resources

In Figure 8.12 the product intent node *GrayBox* "Cover" and a process and its links to the process are sketched out. The four resources outside of the red rectangle provide the input and the output resources of the product intent node. The resources above it serve as inputs, which describe the intended end product: the size of the final format is identified in *LayoutIntent,* the color in *ColorIntent,* and the substrate in *MediaIntent.* The output resource *Com-*

ponent below the red rectangle represents the subproduct "Cover." This resource is then, in turn, input from the higher-level product intent nodes which define the end product.

The *GrayBox* and the process are tied together with a row of resources. The resources do not necessarily lie within the current product intent nodes for the cover; instead they can similarly also be one level higher in the root element, therefore in the product intent node for the overall product. The arrows between resources and the *GrayBoxes* therefore actually only depict the *ResourceLinks.*

At first glance it may appear that the diagram in Figure 8.12 is erroneous, since the input resource *RunList* is listed twice for the *GrayBox.* This is in fact intentional since the contents of the resources are different than the resources themselves. Specifically, the *GrayBox* has the following input resources:

- **Device:** Definition of the platesetter
- **RunList:** File which contains the processed content data, therefore the output from PrePressPreparation
- **Layout:** Signatures of the final product
- **RunList:** Files that contain the marks for the sheet
- **Media:** Plate definition
- **ColorantControl:** Colors of the individual signatures

Some of the resources do not only describe things from the component part, "Cover," but from the total product. These resources (*Layout, RunList, ColorantControl*) must also be within the JDF element of the total product, so the subproduct "Interior" may refer to these resources. From the resources named above there are really only two which have a status equal to *Available: Device* and *ColorControl.* All of the others are actually already applied formally, but without important entries. So the status is set to *Unavailable.* Order management, for example, does not yet know the file name and the location of the content data and also the choice of the printing plate is not made until it is in production.

The *GrayBox "Imposition RIPing PreviewGeneration ImageSetting"* is not elaborated in detail. Only the output resources of this *GrayBox* are set, namely *ExposedMedia* and *Preview,* which in contrast are required input for the following process. The statuses of both resources must naturally also be *Unavailable,* since the *GrayBox* was not yet run—or more specifically: since the *GrayBox* was not

yet converted into processes and these in turn could not yet be executed. As we mentioned in Chapter 6, *GrayBoxes* are indeed fundamentally not executable.

ConventionalPrinting has the following process as input resources:

- **ExposedMedia:** Plates for the job
- **Preview:** Preview images of each separation (for the calculation of the color zone presets)
- **ConventionalPrintingParams:** Information about the printing press type (roll, sheet) and turn
- **Media:** Information about the substrate
- **Ink:** Information about the colors
- **Device:** Information about the printing press
- **ColorantControl:** Colors for individual signatures

Figure 8.13
Element structure of the partial product "cover"

```
<JDF DescriptiveName="Cover" Status="Waiting" Type="Product"...>
    <ResourceLinkPool>
     ...
    </ResourceLinkPool>
    <ResourcePool>
     ...
    </ResourcePool>

    <JDF Category="FinalImaging" DescriptiveName="Cover(CTP)"
     Status="Waiting" Type="ProcessGroup"
     Types="Imposition RIPing PreviewGeneration ImageSetting"...>
       <ResourceLinkPool>
        ...
       </ResourceLinkPool>
       <ResourcePool>
        ...
       </ResourcePool>
    </JDF>

    <JDF DescriptiveName="Cover" Status="Waiting"
     Type="ConventionalPrinting"...>
       <ResourceLinkPool>
        ...
       </ResourceLinkPool>
       <ResourcePool>
        ...
       </ResourcePool>
    </JDF>
</JDF
```

```
<ResourceLinkPool>
    <ComponentLink Usage="Output" rRef="_100" />
    <LayoutIntentLink Usage="Input" rRef="_101" />
    <ColorIntentLink Usage="Input" rRef="_102" />
    <MediaIntentLink Usage="Input" rRef="_103" />
</ResourceLinkPool>
```

Figure 8.14
ResourceLinkPool of the JDF node "cover"

Also, in this case all of the resources up to *Ink, Device,* and *ColorantControl* have a status equal to *Unavailable.*

The element structure of the JDF nodes from the subproduct "Cover" is provided in Figure 8.13. It only lacks the resources and the links to the resources. No matter how one sees the diagram in Figure 8.12, it is clear that the *ResourceLinkPool* of the production node "Cover" must be visible as in Figure 8.14. The other *ResourceLinkPools* are built in the same way. Also the resources themselves are only in part worth mentioning. The *Ink* resource in Figure 8.15 shows only how, for the front and the back of each signature, the required printing colors are to be defined. It defines itself here as a portioned resource of which the breakdown is defined through the values of the *PartIDKeys* attributes.

It is evident that not all of the resources of the "Cover" node are presented here, and the JDF node "Interior" will also not be discussed further, because its construction is similar to the "Cover." Additional resources of the root element were swept under the rug as well, such as the *CustomerInfo* resource with contains the customer data. These all look quite similar to what is already shown

Figure 8.15
Required separations for the partial product

```
<Ink Class="Consumable" ID="_104" PartIDKeys="SignatureName SheetName Side
 Separation" PartUsage="Implicit" Status="Available">
   <Ink SignatureName="SIG001">
      <Ink SheetName="Cover">
         <Ink Side="Front">
            <Ink Separation="Cyan" />
            <Ink Separation="Magenta" />
            <Ink Separation="Yellow" />
            <Ink Separation="Black" />
         </Ink>
         <Ink Side="Back">
            <Ink Separation="Black" />
         </Ink>
      </Ink>
   </Ink>

    And so on for the other four signatures of the interior

</Ink>
```

Name	Data Type	Description
BillingCode ?	string	A code to bill charges incurred while executing the Node.
CustomerID ?	string	Customer identification used by the application that created the Job. This is usually the internal customer number of the MIS system that created the Job.
CustomerJobName ?	string	The name that the customer uses to refer to the Job.
CustomerOrderID ?	string	The internal order number in the system of the customer. This number is usually provided when the order is placed and then referenced on the order confirmation or the bill.
CustomerProjectID ? New in JDF 1.2	string	The internal project id in the system of the customer. This number might be provided when the order is placed and then referenced on the order confirmation or the bill.
rRefs ? Deprecated in JDF 1.2	IDREFS	Array of *ID*s of any Elements that are specified as ResourceRef Elements. In version 1.1 it was the IDREF of a ContactRef. In JDF 1.2 and beyond, it is up to the implementation to maintain references.
Company ? Deprecated in JDF 1.1	refelement	Resource Element describing the business or organization of the contact. In JDF 1.1 and beyond, Company affiliation of Contacts is specified in Contact.
Contact * New in JDF 1.1	refelement	Resource Element describing contacts associated with the customer. There SHOULD be one Contact [contains (@*ContactTypes*, *"Customer"*)]. Such a **Contact** specifies the primary customer's name, address etc.
CustomerMessage * New in JDF 1.2		

Figure 8.16 Requirements of the CustomerInfo resource in the JDF specification

in Figures 5.2 and 5.3. Now we will stop showing the code and our examples of it.

Instead we want to follow a different path, namely to clarify the abstract definition of the *CustomerInfo* resource, as reflected in JDF specification 1.4. This should make it easier for the reader to grasp the specification. The tables are basically built like the tables in Figure 8.16: in the left column are the names of the structure elements (like an attribute, a subelement, or a referent to a resource), the middle column gives the data type, and in the right column the structure element is explained. Noticeable in the first column are the blue comments like "New in JDF 1.2" or "Deprecated in JDF 1.1." The last comment means that the corresponding structure element from Version 1.1 has been abandoned, so no JDF code with a version higher than 1.1 can contain it. For the sake of compatibility it must naturally remain in the specification table; however, we do not wish to proceed further with such "outdated" concepts. In addition, you will see special characters in the name like a question mark (?) or asterisk (*). Sometimes (elsewhere in the specification) you may also see a plus sign (+) or no special characters at all (see, for example, Figure 12.8). These refer to the number of possible instances:

? Optional

* Never or several times

+ Once or several times

If no special characters are in the name, then the structure element is required and must appear exactly one time. The attribute *CustomerID* may therefore appear only once, but must not appear again while the subelement *Contact* can appear never, once, or several times.

The datatype *String* is simply a character string, a *RefElement* is an Element or a reference to an element, whereas with an *Element* (which is not present in Figure 8.16) the element must be directly accessible.

Here is the short explanation of the structural elements, which are current in Version 1.4:

- **BillingCode:** An account number for the job
- **CustomerID:** Customer number from the MIS
- **CustomerJobName:** Customer name
- **CustomerOrderID:** Customer's order number
- **CustomerProjectID:** Customer's project number
- **Contact:** (Reference to an) Element which describes a contact
- **CustomerMessage:** Description of a message to the customer, for example an automatic emailing when the node is produced

The *Contact* and the *CustomerMessage* elements get their own table in the specification. Since both elements have subelements (*Address, ComChannel, Company,* and *Person*), one finds these specified in other tables. The hyperlinks were, however, not introduced so as not to confuse the reader; instead, because elements can be employed from different parent elements, with this technique they must stand only one time in the specification. For example, the *ComChannel* element is a subelement of both *Contact* as well as *CustomerMessage*. In reality the reader must not only follow the links at the bottom (i.e., to the possible child elements) but instead also to a certain extent above: there are so-called "abstract resources" which the attributes describe and which all of the resources may have. These general attributes (for instance, *ID,*

Class, or *Author*) are then listed no more in the table of concrete resources. The same also goes for JDF nodes.

Up until now we have covered nearly all of the flow of data from MIS to prepress and not in the reverse direction, which is crucial for final costing. For this path there are two possibilities that are often used in parallel:

- MIS receives JMF messages from the production system.
- The JDF, which was changed by the production system, is transferred back to the MIS.

Above all, the *AuditPools* are read at the return receipt of the JDF from the production system, as already stated in Chapter 6.

8.3 MIS ICS Papers

The importance of the interface between the order management system/MIS and production can be seen in the number of the corresponding ICS papers (Figure 6.7). To date, the following documents are current in Version 1.3; however, these have relatively short lives and the reader is cautioned to obtain the newest version on the Internet [12].

- *MIS ICS*
- *MIS to Prepress ICS*
- *MIS to Conventional Printing – Sheetfed ICS*
- *Newspaper: MIS to WebPress ICS*
- *MIS to Finishing ICS*

We want to go into the first of these ICS papers only briefly; the others are presented in part in the relevant Chapters 9–11. The *MIS ICS* document specifies general requirements for the MIS which are not specific for prepress, press, or post-processing.

Also in the *MIS ICS* are—similar to the *JMF ICS*—different levels of compliance are defined. There are three levels which concern JMF messages; we will only present the first two (Figure 8.17).

- **Level 1:** The MIS hands off only JDF via hot folders to the *Worker* (Figure 8.17, Level 1).
- **Level 2:** The MIS may additionally generate JMF queries with the help of the *NodeInfo* element, in order to create a

persistent channel from the *Worker* to MIS (Figure 8.17, Level 2a). With these the signal can be sent from *Worker* to MIS. This method corresponds to Level 1 in the *JMF ICS.* The Worker in this case must not be an HTTP server, which means it is not able to accept and process HTTP packets. Alternatively (Figure 8.17, Level 2b) in Level 2 the MIS can directly send JMF messages. With this option the *Worker* may very well be an HTTP server and one assumes Level 2 of *JMF ICS.*

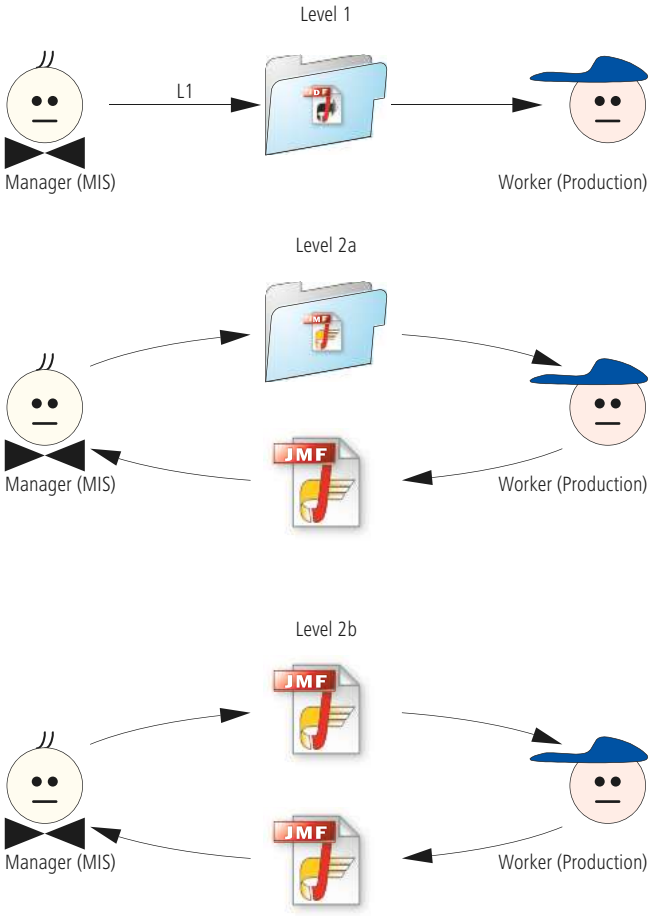


Figure 8.17
Level 1 and 2 of the MIS ICS

Other requirements regarding JMF are also raised. For example, when the MIS sends a resource query to a *Worker,* the information about resources which so far were consumed or produced is returned (Figure 8.18). MIS software which may fulfill Level 2 of the *MIS ICS* must also be able to send the corresponding queries (whether via *NodeInfo* or via a direct JMF message). An ICS Level 2 MIS *Worker* must be able to receive these queries, interpret, and respond. The same goes for requests concerning device status and job statuses. In other words, job tracking and device status monitoring require *MIS ICS* Level 2 both for the MIS software as well as for the Controller and devices in production. In practice, there are certainly implementations of JDF workflows that do not support these functions.

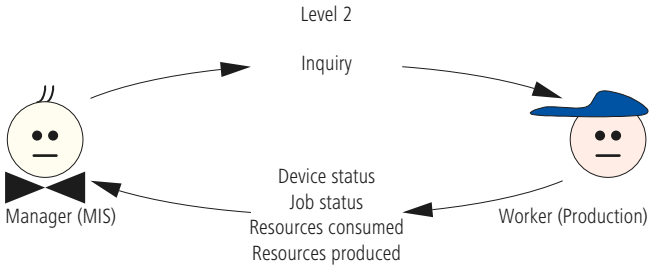


Figure 8.18
The Worker must inform the Manager about jobs and resources when asked

But requirements for JMF elements are not only defined in the *MIS ICS;* many also concern JDF. Many small details must be

established. For example, it is required that the ICS level is entered in each of the MIS-generated JDF root elements in a special attribute (ICS Versions) on which the JDF document was built. Further, each JDF root element must also have two special resources as input, namely the *CustomerInfo* resource in which customer data is placed and the *NodeInfo* resource which contains the schedule and optionally one or more JMF message(s). Each of the MIS-generated JDF root nodes is a product node, and the properties of the product are stored in *Intent* resources. Each JDF root element must also at minimum have one *Intent* resource as input. If there were none, the product would be without properties and therefore rather meaningless. We already mentioned earlier that each JDF root element must be related to an output resource *Component* which represents the end product.

After completion of the processes, each *Worker* must record information in the JDF process nodes about the start, end, duration, and the like to hold as an *Audit* element. In *MIS ICS* Level 2 it must also enter further information into the *ResourceAudit* element in the *AuditPool.* These contain information from the *Worker* about materials used during execution of the process (plates, paper, ink, etc.).

8.4 PrintTalk/JDF Interface to Customers

The PrintTalk specification was introduced to the public in 2000 by the organization of the same name under the umbrella of the NPES (Association for Suppliers of Printing, Publishing and Converting Technologies). In 2004 PrintTalk integrated itself with CIP4 and was then dissolved as an independent organization. The PrintTalk specification is based on cXML, as previously discussed in Section 5.4.

Some order management systems can be JDF documents which were sent from the customer and imported to generate a new quotation or a new order. So a customer can, by way of example, generate a JDF file in Acrobat which describes the product, fundamentally made from product nodes. The print shop would then import these into the order management system. For this reason the order management system would receive the necessary product description from the outside; however, the business operations would not continue to be covered. This mode of operation, however, has a catch: The *JobID* is created by the customer, and it is doubtful that it would be in the number range of the print shop. Conversely it makes more sense when a print shop formally enters a job into the MIS, the nearly empty JDF file is sent via email to the customer, who then can import the file into Acrobat and

complete it with order data. Finally, the customer can send the completed JDF file back to the printer, which can, in turn, automatically allocate the information to the existing order.

Figure 8.19
PrintTalk communication between customer and printer

End customers and delegated production agents/service providers can communicate order parameters with the print shop via JDF. Thereby the agency can automatically draw important parameters (for example, the delivery time and/or the number of copies) out of the customer's system, complete them accordingly, and forward them to the print shop of choice. With this scenario electronic feedback is very advantageous.

With JDF, communication between customer and print shop can be achieved. PrintTalk, however, goes one step further.

The principle is quickly explained in Figure 8.19. Business objects like requests or offers are exchanged over the Internet between customers and print providers. Orders, order confirmations, delivery documents, and invoices are also communicated in this way. Consequently, through the integration of a PrintTalk interface with an MIS system, many administrative steps can be simplified. In principle it is therefore a portal or Web-to-print solution.

There are two basic configurations: either a client communicates directly with a printer's Web server which offers such a service, or there is a broker between both parties. In the second case, the customer would log themselves into the Web server of the broker, who would send price requests to different print shops via PrintTalk. This will then generate offers (either fully automated in real time or time delayed under the control of an employee) and send them as PrintTalk objects back to the broker. The broker would then potentially filter the offers and re-edit them before sending them online to the customer. This procedure is widely used in other service sectors, for instance in the travel industry. The automated queries for prices naturally further generate price wars.

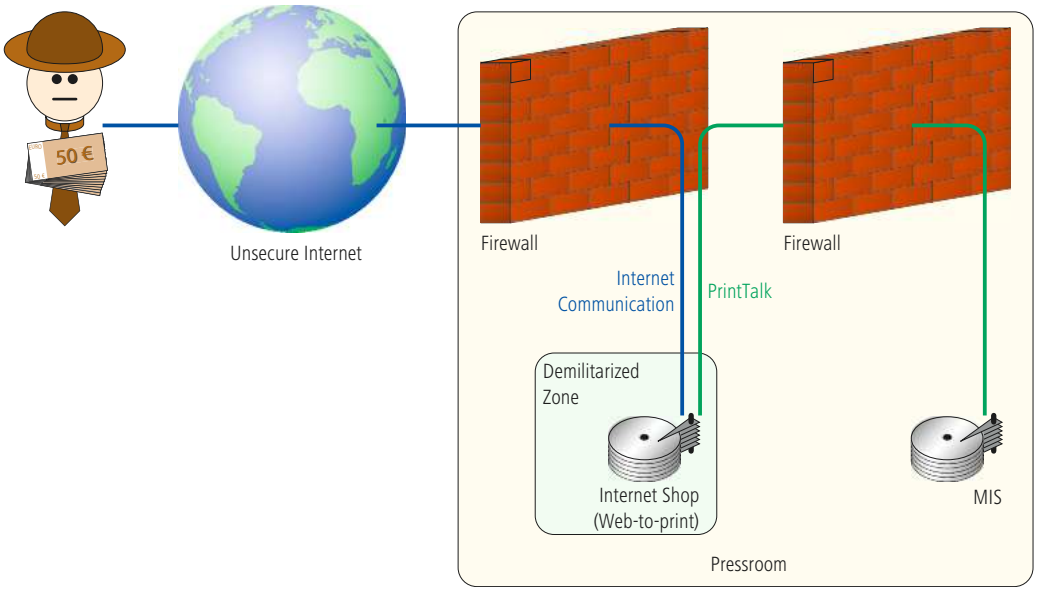
If the communication takes place directly between customer and printer, the technical implementation can once again turn out differently. Either the customer already generates PrintTalk documents with his or her browser and sends them to the printer's appropriate server, or there is a "standard" Internet communication between customer and printer and the Web-to-print server in the print shop first generates the PrintTalk communication. In the last

case PrintTalk would only form a printer's internal interface protocol between the Web-to-print software and the MIS. This situation is shown in Figure 8.20. Of course an MIS server must lie within a protected area in the local network of the printer, also behind a firewall. The server which has the e-commerce shop installed on it is usually in the "demilitarized zone." The (potential) customer of a printer also communicates over the Internet with an online ordering system, which in turn generates PrintTalk data and passes these to the MIS. For this reason predefined products from outside can be ordered with the help of HTML and the MIS picks up defined *BusinessObject* elements in PrintTalk as well as JDF product descriptions. This configuration allows an independent communications interface between Web portal and MIS as long as PrintTalk and JDF are both supported.

With PrintTalk a business-to-business (B-to-B) communication between broker and printer as well as a business-to-consumer (B-to-C) interface between the customer and the printer are built. In fact there are very few PrintTalk applications, and we will see in the future which configuration will be more frequently implemented. It's possible that the reason for a reluctance to implement it is that MIS vendors often offer their own Web portal solutions and are therefore less interested on an open interface to MIS.

Figure 8.20
A printer's Web server and MIS; only the PrintTalk protocol is used within the print shop

In Figure 8.21 the structure of a PrintTalk document is shown, which is very similar to the general cXML structure which we presented in Figure 5.7. The header in the PrintTalk document is built



exactly the same as in cXML. Also there is a request element in PrintTalk. Its subelements now look somewhat different; it contains exactly one so-called *BusinessObject*. In this example it is a purchase order, but in total there are twelve different *BusinessObjects*.

- **Quotation:** Estimate
- **PurchaseOrder:** Purchase order
- **Confirmation:** Order confirmation
- **Cancellation:** Cancellation of a business object
- **Refusal:** Rejection of a business object
- **OrderStatusRequest:** Query about order status
- **OrderStatusResponse:** Answer about order status
- **ProofApprovalRequest:** Request for proof approval
- **ProofApprovalResponse:** Proof approval or rejection of proof
- **Invoice:** Invoice
- **ReturnJob:** Print shop sends the job back to the customer

```
<PrintTalk...>
    <Header>
        <From>
         ...
        </From>

        <To>
         ...
        </To>

        <Sender>
         ...
        </Sender>
    </Header>

    <Request>
        <PurchaseOrder...>
            <jdf:JDF ...>
                <jdf...>
                 ...
                </jdf>
                <jdf...>
                 ...
                </jdf>
            </jdf:JDF>
        </PurchaseOrder>
    </Request>
</PrintTalk>
```

Figure 8.21
Structure of a PrintTalk document

Each *BusinessObject* naturally also has attributes which specifically define the business transaction. In order placement, for example, the currency is defined as are the options for subcategories like form of payment and price level. In the example in Figure 8.22 we even have two prices, one for the production of the print product

Figure 8.22
Ordering with PrintTalk

```
<PurchaseOrder AgentID="CC" AgentDisplayName="CarlCool"
 RequestDate="2008-11-13T11:00Z" BusinessID="A001" Currency="EUR"
 Expires="2008-12-13T11:00Z ">
    <Pricing>
        <Price LineID="_1" DescriptiveName="Hard Cover Books” Amount="6800"
         Price="15000.00" />
        <Price LineID="_2" DescriptiveName="Shipping” Price="980.00"/>
    </Pricing>
    <jdf:JDF...>
        <JDF...>
         ...
        </JDF>
    </jdf:JDF>
</PurchaseOrder>
```

and the other for the delivery. In earlier version of JDF, price information and payment modalities were allowed in the *DeliveryIntent* resources. Meanwhile all of this information was removed from JDF and put into corresponding PrintTalk elements.

Some of these *BusinessObjects* contain only one or more JDF nodes. The *PurchaseOrder* element from Figure 8.22 actually contains three JDF nodes, where only the top node is depicted. The description of the print product is then stored in the JDF element. Subproducts of the order will be specified in other JDF nodes. For example, the information about the customer (*CustomerInfo*), about the intended delivery of the end product (*DeliveryIntent*) would then be in the *ResourcePool* of the JDF root element like normal. At this point, however, we will not continue this example since it is built exactly according to the JDF rules which we have already described in the last sections and in Chapter 6.

Exercise:

Create a JDF file with Acrobat. Indicate the customer, material, and the product itself (for example, a brochure with 60 CMYK pages inside and cover in CMYK+spot color).

Download the *JDF Editor* from CIP4.org, open the previously produced JDF file, and analyze the structure.

Rename your *.jdf file to *.xml and open it with your browser. Examine the JDF data and identify there the values you entered in Acrobat.

9 Prepress

From the perspective of prepress, the JDF/JMF world looks like Figure 9.1, where each arrow illustrates a potential JDF/JMF connection. Typically a prepress workflow management system receives its JDF input data from an order management system (Arrow 1). Depending on the configuration, prepress can also return messages back to the order management system. We have already seen that the JDF data which the MIS provides is not really sufficient for production (keyword: *GrayBox*!) which means prepress will create new JDF structures and optional JMF messages in different modules (Arrow 2). A section of it is taken over from the order management system and the newly generated data will be passed from prepress to the printing presses (Arrow 3) and to postpress processing (Arrow 4) either directly or via the MIS. Interfaces 1 and 2 are only highlighted in this chapter, while interfaces 3 and 4, print and postpress processing respectively, are analyzed in Chapters 10 and 11. Interface 1 is characterized by the appropriate ICS paper, the *MIS to Prepress ICS,* which is discussed in Section 9.1. We will explain internal prepress interface 2 with the help of examples in Sections 9.2 to 9.5.

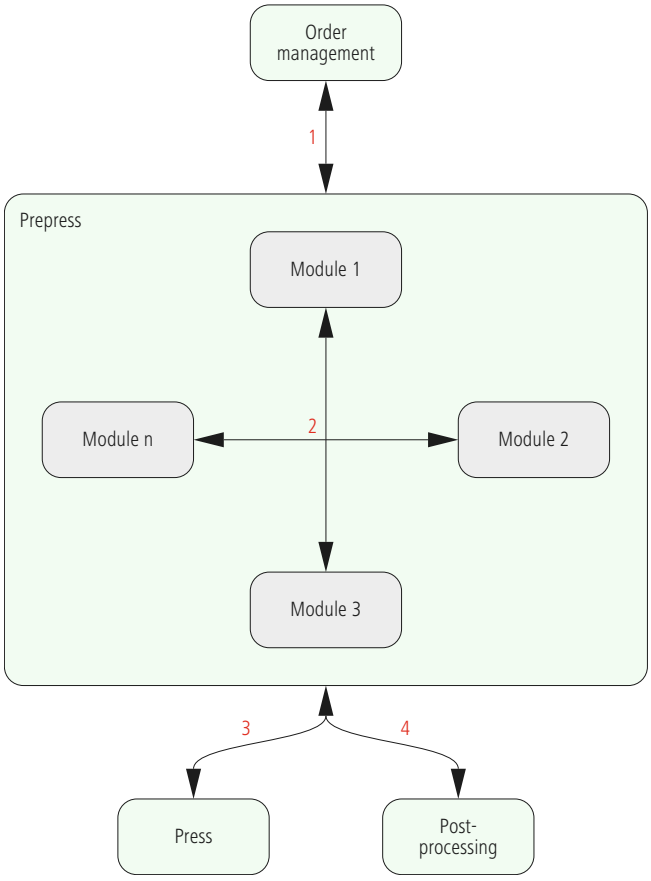


Figure 9.1
JDF/JMF linkages in prepress

But first remember the steps in prepress which we already listed in Section 3.2. We have assumed that prepress receives ready-made PDF data. The actual layout and the PDF creation are therefore not on the list. These tasks are indeed typical for job shop printing, but there are naturally also variations. So a form proof could be implemented as a soft proof, as a remote proof, or even completely eliminated; additionally color binding page proofs can be created and so on. Workflows for packaging print prepress often look quite different.

In order to cope with the myriad tasks in prepress, a prepress WMS exists in most cases as different modules. In Section 4.3 we have seen that the PJTF realizes the communication with the Extreme workflow architecture between the coordinator software and the job ticket processors (just the modules). This function takes over only the JDF/JMF (arrow 2 in Figure 9.1). As in the case of PJTF, which also goes for a JDF workflow, a standardized data exchange format is not enough in order to permit the user of a WMS the cooperation of individual modules from different vendors. Myriad specific agreements are required.

9.1 Interfaces between MIS and Prepress

Before we explore the *MIS to Prepress ICS* in this section, we want to engage in a couple of general considerations about this interface.

Ideally, besides the job definition and the definition of production already in the order management system, one would also determine the production path in prepress. In fact, some prepress work-

Examples of JDF-compatible WMS (2009)

Manufacturer:	***Main product:***
AC&C HSH Group	PuzzleFlow
Avato System GmbH	PMS Production Management System
Agfa	Apogee
DALiM	PRiNTEMPO
EFI	OneFlow
EskoArtwork	BackStage Server
Fujifilm	XMF
Krause-Biagosch GmbH	KIM
LithoTechnics Oty Limited	Metrix
Heidelberger Druckmaschinen AG	Prinect
manroland	Printnet
Kodak	Prinergy
OneVision Software AG	Asura Pro
RamPage	RamPage Workflow System
Screen	TrueFlow
Xerox	FreeFlow Print Manager

flow systems propose certain workflows on the basis of the JDF information they receive from the order management system. But as many details relevant to production are not known at the order management stage, the WMS itself must fill in the gaps with default values. The production paths are therefore not fully established but clearly can still be changed.

Overall, the sequence of work steps the prepress software performs for a print order can be determined in multiple stages:

1. A system administrator can define new workflows through scripting.
2. The WMS agent of a print shop can define defaults for different work paths and production parameters for classes of print products. These settings are tangential to the basic workflow that the manufacturer of the WMS software provides.
3. The user can make individual changes to the workflows suggested by the WMS for individual print products.

In the first case there are different scripting possibilities which are either furnished by the operating system or from the WMS manufacturer. **AppleScript** [8] or **Windows Script Host** (WSH) [34] with the **VBScript** or **JScript** engine belong to the first category; the **rules-based automation** (RBA) [30] in **Prinergy** (Kodak) belongs to the latter category. These technologies are all very powerful and it would take too long to deal with them here. If interested in RBA refer to the box on page 126.

In the second case, the WMS agent defines the default raster settings in RIP or the trapping values. He or she can also define hot folders for the input of content data, an automatic page order on the imposition layout via rules for file names, or integrate new machines into the workflow.

In the third case, a graphical representation of the available WMS workflows is presented on the user's screen. The user or users have the option to change single work steps for the print job or—for example, to turn off trapping or to modify the default raster settings in the RIP.

The interface between MIS and prepress is principally critical. In theory, after a job is placed into the MIS, the technical details are passed to the prepress WMS, which subsequently executes the order; in practice order changes occur while a job is already being worked on in prepress production. These can be values that are not problematic, such as the number of copies (prepress must gener-

ate more sets of plates due to the lack of durability of press plates with a high volume), as well as things that do affect them directly such as color or on which printing press the job will be printed. Indeed there is an *UpdateJDF* command, which could be sent from the MIS to the prepress system, yet only now are users beginning to implement these options. Naturally, in practice, it commonly goes the other way: A prepress operator changes things, and the operator should inform the MIS so the production path would not be in two separate versions in the JDF description. The problem with JDF updates is only at the beginning of being solved so far and will surely be an important issue in the coming years.

In the *MIS to Prepress ICS, GrayBoxes* are mainly defined as information containers, which the MIS builds and must be filled out with details from prepress. There are a total of 11 of these *GrayBoxes:*

Rules-Based Automation

With RBA, one can affect and further automate the workflow of Prinergy outside of the normal Prinergy user interface. This can be done with a graphical tool to create rules. With a rule, an event that can occur in Prinergy is associated to an action.

For example, one could specify that if a fatal error occurs while checking the data (in Prinergy jargon: when refining), a specified person in the business receives an email message about it. So you can organize the import of the content data in the Prinergy system via a hot folder, allow data proofing in the background, and only intervene manually in the case of a failure upon receipt of the email.

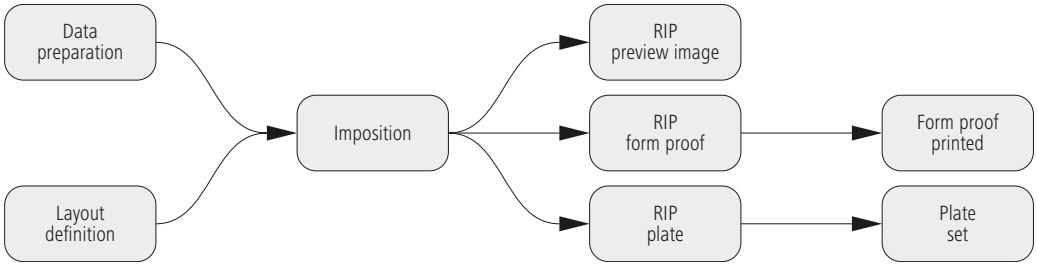
Another example: using a naming convention for the file names allows you to place the pages on the layout automitacally. An RBA rule captures the "Sheet Filled" event and the output is automatically fed to a form proofer.

Via RBA rules, third-party software can also be integrated into the Prinergy environment. In addition, you can also define workflow controls with the Visual Basic computer language using graphical RBA tools .

- *PrepressPreparation*
- *ImpositionPreparation*
- *ImpositionProofing*
- *ImpositionRIPing*
- *PlateSetting*
- *PlateMaking*
- *ImpositionSoftProofing*
- *PageProofing*
- *PageSoftProofing*
- *ContentCreation*
- *ProofAndPlateMaking*

In order to describe the prepress workflow, these *GrayBoxes* could also be assembled like building blocks, and, as is usual with building blocks, there are large and small ones. Based on the activity diagram in Figure 9.2, we clarify this somewhat nebulous analogy. A typical workflow is sketched out showing schematics for each task (preview image, form proof, plate) in each case, newly RIPed. The RIP processes, however, are very different. While the preview image has perhaps a resolution of 72 ppi and the form proof is generated at 600 ppi, for plates it must reach the addressing resolution of the imagesetter, which is usually 2400 or 2540 dpi. Furthermore, separated images are required for plate imaging; in contrast, the form proof requires composite images. A preview image is sometimes separated and sometimes a composite depending on the intended use.

Figure 9.2
Prepress activities flowchart (commercial printing)



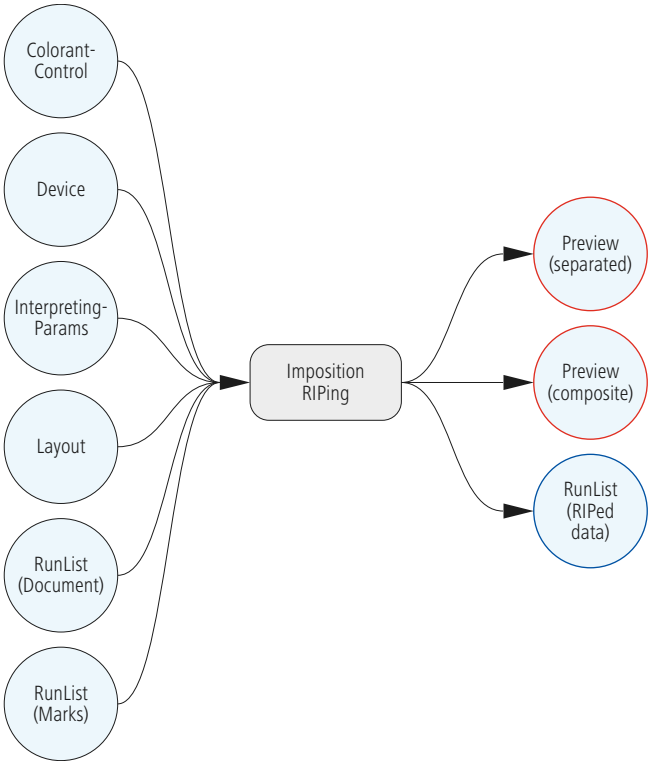
The activities in Figure 9.2 are also reflective of the *GrayBoxes:*

- **PrepressPreparation:** Data Preparation
- **ImpositionPreparation:** Definition of imposition layout
- **ImpositionProofing:** Imposition, RIPing the form proof, form proof output
- **ImpositionRIPing:** Imposition, plate RIPing; also optional: RIPing the preview image
- **PlateSetting:** Plate burning

The *GrayBox PlateMaking* is to some extent a combination of *ImpositionRIPing* and *PlateSetting* and therefore a large "building block" which makes both of the smaller ones redundant. An even larger one is the *GrayBox ProofAndPlateMaking,* which is comprised of *ImpositionProofing* and *PlateMaking.*

The *MIS to Prepress ICS* defines an attribute for each *GrayBox,* but above all, it defines the required input and output resources as well as their attributes. The three *GrayBoxes ImpositionRIPing, PlateSetting,* and *PlateMaking* are presented with their resources in Figures 9.3 to 9.6, whereby *PlateMaking* has already been illustrated in Figure 6.18. Normally *ImpositionRIPing* and *PlateSetting* are daisy chained and the blue-outlined *RunList* resource, which typically represents the screened separations in TIFF format, forms the handoff between the two. Figure 9.6 shows how the *GrayBoxes PlateMaking*, *ImpositionRIPing,* and *PlateSetting* are comprised. Of course, here the output from *ImpositionRIPing,* as far as it doesn't serve as input for *PlateSetting,* also occurs as output from *PlateMaking.* We have outlined these two *Preview* resources in red for clarification. In addition, *PlateMaking* must comprise all input resources

Figure 9.3
GrayBox ImpositionRIPing



from *ImpositionRIPing* and *PlateSetting,* but the blue-outlined *RunList* resource is only depicted as an intermediate result.

An MIS can also describe the workflow in prepress differently, as one can see in Figure 9.6, where the arrows simply symbolize the result of the *GrayBoxes.* Depending on the *Manager's,* which is to say the MIS, love of detail, different *GrayBox* drawings are built. The *Worker,* that is to say the workflow system in prepress, must be able to interpret all of the versions in order to convert them to JDF process nodes in the course of production.

Also several incremental levels are defined in the *MIS to Prepress ICS.* All of the introduced *GrayBoxes* are defined in Level 1 (and therefore automatically also for Level 2). Level 2 meets additional conventions for versioning, such as that which occurs in multiple languages of a product, and also for complex production in which different parts of the production are brought together on the same sheet (e.g., cover and contents).

Finally, it should be noted that the *MIS to Prepress ICS* paper contains the requirements of the other ICS documents upon which it is based. So it is assumed *Base ICS* and *JMF ICS* are in Level 2—and indeed for both stages of the *MIS to Prepress ICS.* The steps, therefore, can also be different. With *MIS*

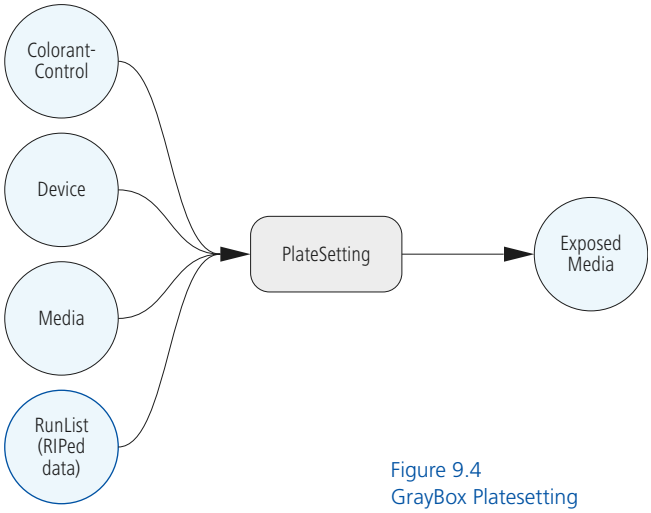


Figure 9.4
GrayBox Platesetting

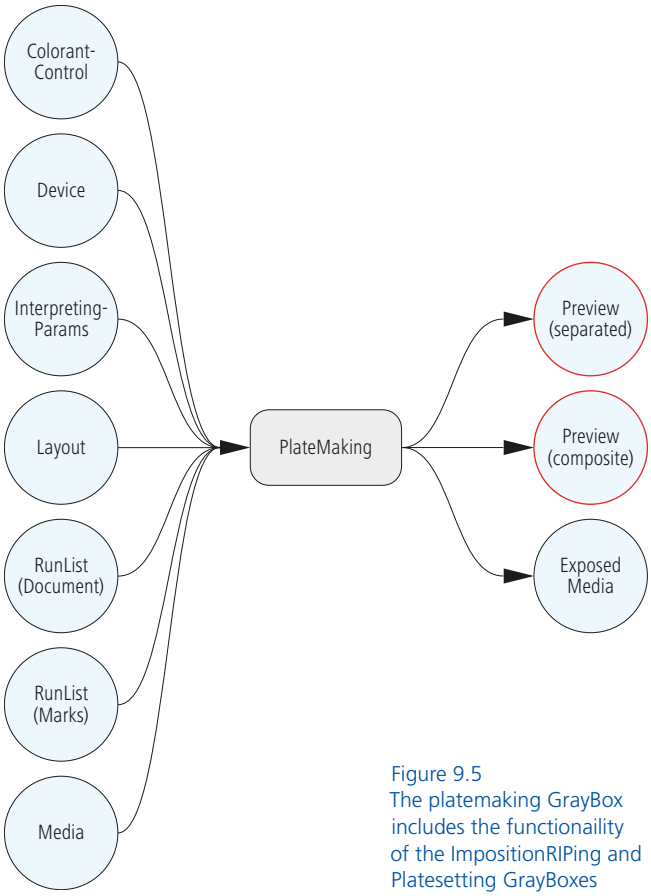


Figure 9.5
The platemaking GrayBox includes the functionaility of the ImpositionRIPing and Platesetting GrayBoxes

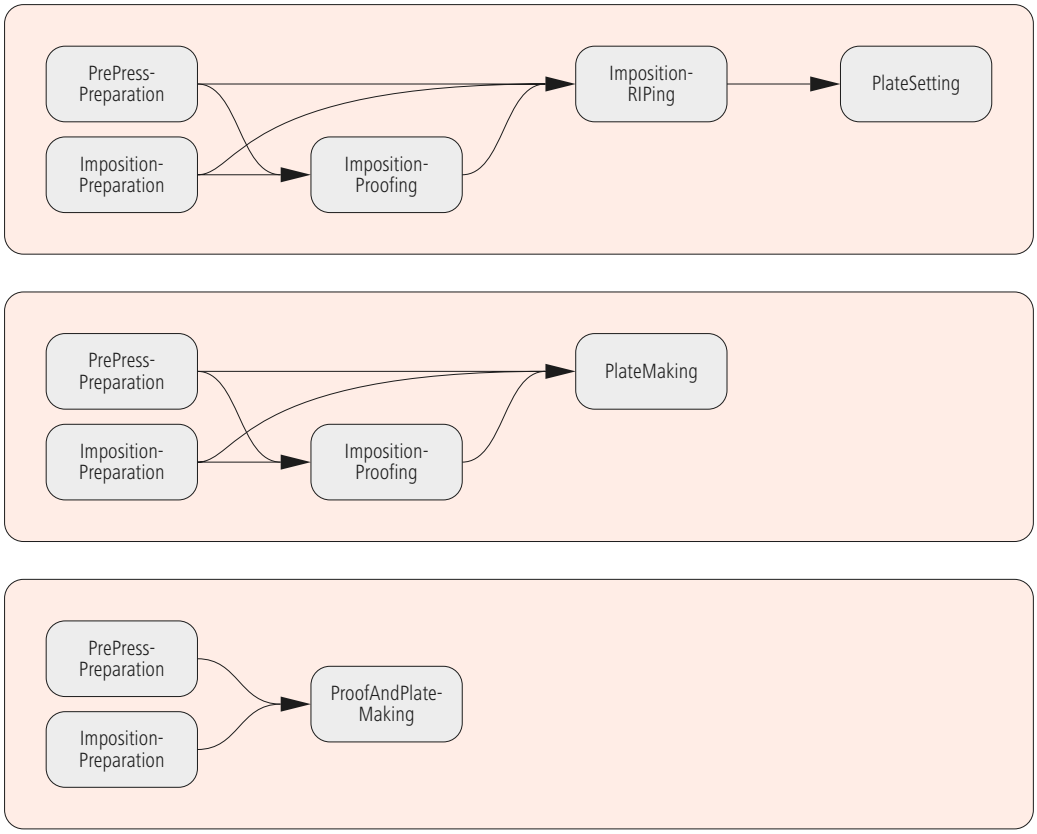


Figure 9.6
Three different options for linking GrayBoxes

ICS, which is likewise naturally a precondition for the *MIS to Prepress ICS,* the situation is actually as one would expect: Level 1 from *MIS ICS* is a prerequisite for Level 2 of *MIS to Prepress ICS,* and Level 2 of *MIS ICS* is a prerequisite for Level 2 from *MIS to Prepress ICS.*

9.2 Assembly

Montage consists of the following three steps that are executed in succession:

- Imposition layout creation
- Allotting the sides/tiling on the imposition layout,
- Imposition, that is the calculation of a data structure that contains all of the printing data for the exposure of one or more plates. The individual pages are also calculated together with the marks (typically into a PDF data structure).

All three steps were completed earlier in the assembly software. Meanwhile, it is common to carry out the imposed sheet creation in this software, while the mapping occurs in the WMS background and the imposition occurs in the background on a powerful workflow server. However, due to the ever-increasing integration of assembly and WMS software, the split is further blurred.

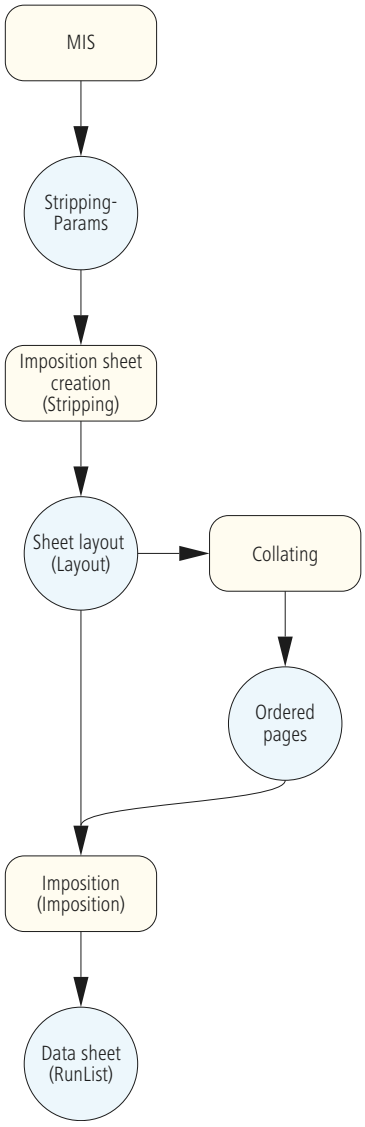
This section is mainly about the creation of the imposition layout, commonly also called the sheet layout. The tasks are typically:

- Placement of the print sheet on the plate
- Definition of sheet and page sizes
- Application of the imposition schema
- Definition of the margins and spacing (gripper edge, gluing edge, trim, etc.)
- Placement of control elements (color bar/control strips, register, fold, cutting, collating and page marks, sheet signatures, etc.) on the print sheet and plates

The order management system can already provide much of this information. We have already extensively analyzed the *StrippingParams* resources in the previous chapter which contained exactly this information. Figure 9.7 also gives a highly simplified workflow structure. We would like to stress that not all processes and resources necessarily represent the JDF structures, instead only the general tasks reflected in the workflow.

A sizable potential for automation is included in this sequence. Without JDF connections to an order management system, all of the specifications for the sheet layout must be entered manually in the assembly software, namely by the creation of a new imposition layout. Of course, the imposed sheet could be saved and reused with minor modifications for orders with the same structure. However, with a JDF integration between the order management system and prepress, all of the values can be directly transposed from the order management system, and the imposition layout can be generated automatically. In the offset world, admittedly, each press sheet layout created this way is controlled by an operator and has minor details changed. A prerequisite for this method is that the control marks must be positioned rel-

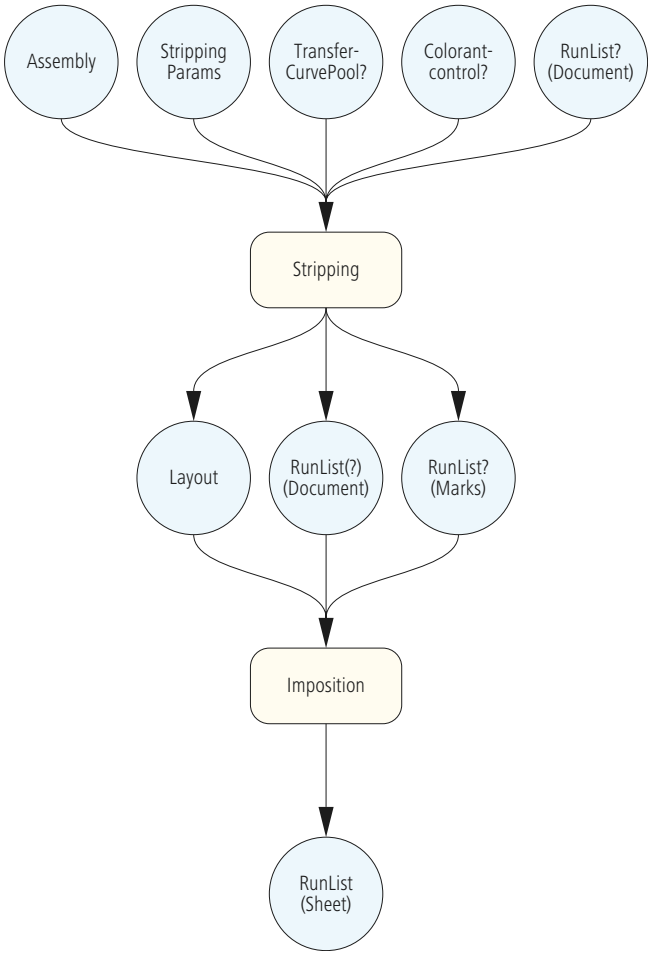
Figure 9.7
The assembly process resource model



ative to the sheet size and not, as was common earlier, with fixed coordinates placed on the sheet. Sheets of differing sizes require the marks in different positions, and since these positions are not available from the order management system (it only has specifications for the sheet size), the marks must be automatically adjusted. This process is actually defined and carried out by the assembly software. From JDF 1.4 and higher, dynamic marks can be characterized in the *StrippingParams* from the MIS and the stripping process must not have run yet. However, it is necessary that both the staff and also the software fill out the corresponding requirements when the order is generated.

We will now more closely analyze the JDF structure in Figure 9.7. In Figure 8.7 the process for the creation of the imposition layout, namely stripping, was already presented with a single-input and single-output resource. Figure 9.8 now reflects the situation more thoroughly. In this figure a "?" denotes optional resources. As you can see, we have now also assimilated the input resources *TransferCurvePool, ColorantControl,* and *RunList (Document)* for the *Stripping* process. The *TransferCurvePool* resource primarily contains the transfer curves, also known as tonal value passing curves or process calibration curves. They are really only important for the RIP process, and therefore we will describe them in more detail in Section 9.4. To this end, the resource is only used (if necessary) in order to receive information about the coordinate transformations between imposition layout and plate. In *ColorantControl* the colors of the separations are configured so that the corresponding color marks can be generated on the imposition layout as well. Finally, it is possible that a *RunList* resource, which describes the document to be printed, would

Figure 9.8
Assembly in the JDF model



be changed in the *Stripping* process when the pages are imported. Then one could build, or control as the case may be, the imposition layout under visual control of the print data on the screen.

The *RunList* resource could, in this case, be passed to the *Imposition* process. Which means that this resources can, but must not be, output from the *Stripping* process (because of the "?"). In any case, a *RunList* (wherever it comes from) must be an input resource from the *Imposition* process; otherwise it would have nothing to impose. In this respect we have again bracketed the question marks in the figure. Typically the *Stripping* process will also generate a file (or multiple files), which contains all of the control marks on the plate in the correct positions. These are described in the *RunList (Marks)* resource. Function and position of control marks can indeed be provided via JDF, but not their appearance. In this respect it must occur separately—in PDF, for example.

Finally we wish to show an example for a *Layout* resource in Figure 9.9. in which a placeholder for six pages is defined. Thus, a placeholder is a *ContentObject* in the language of JDF. As you can see, the layout is partitioned according to the key resource *SignatureName SheetName Side.* This must be because the placeholder for the sides and the position of the marks can be different from signature to signature, from sheet to sheet , and also between front and back. The layout of the example sheet is sketched out in Figure 9.10. Notice that in our example the size of the *SurfaceContentsBox* is exactly the size (*Dimension*) of the plate (*Media*). The zero point for both is at the bottom left. In general the *SurfaceContentsBox* could also be smaller, for example, the smallest rectangle that includes all of the printed elements on the sheet (= BoundingBox). The attributes of *ContentObject* are as follows:

CTM:

The *Current Transformation Matrix* was already introduced in Chapter 4.3 as a *Dictionary* entry in PDF. This idea, which originally stemmed from the PostScript world, has a similar function in JDF. It also defines the position, orientation, and size of a placeholder on the *SurfaceContentsBox* and therefore here on the plate. Also see the exercise at the end of the chapter.

ClipBox:

The size and position of a placeholder is defined here in DTP points. All elements of a page outside the *ClipBox* are cut off. The position refers to the *SurfaceContentsBox.*

```
<Layout Class="Parameter" DescriptiveName="Status" ID="_300"
 PartIDKeys="SignatureName SheetName Side" Status="Available">
   <Layout Name="Signature-1" SignatureName=" Signature-1">
      <Layout DescriptiveName="Cover" Name="Cover" SheetName="Cover"
       SourceWorkStyle="WorkAndBack"
       SurfaceContentsBox="0 0 2111.81 1714.96">
         <Media Class="Consumable" Dimension="2111.81 1714.96"
          MediaType="Plate" />
         <Layout DescriptiveName="Recto" Side="Front">
            <ContentObject CTM="1 0 0 1 316.06 188.50"
             ClipBox="307.55 180 1036.06 537.16"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 188.50" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 188.50"
             ClipBox="1075.74 180 1804.25 537.16"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 188.50" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 316.06 585.35"
             ClipBox="307.55 576.85 1036.06 934.01"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 585.35" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 585.35"
             ClipBox="1075.74 576.85 1804.25 934.01"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 585.35" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 316.06 982.20"
             ClipBox="307.55 973.70 1036.06 1330.86"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 316.06 982.20" TrimSize="711.49 340.15" />
            <ContentObject CTM="1 0 0 1 1084.25 982.20"
             ClipBox="1075.74 973.70 1804.25 1330.86"
             PositionX="Center" PositionY="Center"
             TrimCTM="1 0 0 1 1084.25 982.20" TrimSize="711.49 340.15" />
            <MarkObject CTM="1 0 0 1 0 0" ClipBox="0 0 2111.81 1714.96">
               <RegisterMark Center="1900.86 661.94" Class="Parameter"
                Rotation="0" />
               <RegisterMark Center="210.94 661.94" Class="Parameter"
                Rotation="0" />
               <ColorControlStrip Center="1059.44 1347.70"
                Class="Parameter" Rotation="0" Size="1486.18 28"
                StripType="FOGRA_6_F74_740x10" />
            </MarkObject>
         </Layout>
          ...
      </Layout>
   </Layout>
</Layout>
```

Figure 9.9
JDF code of a layout resource

PositionX, PositionY:

Nature of the content orientation within a placeholder.

TrimSize:

The size of a *TrimBox,* which means the pages after cutting or the final format of the printed product. The size is specified in DTP points.

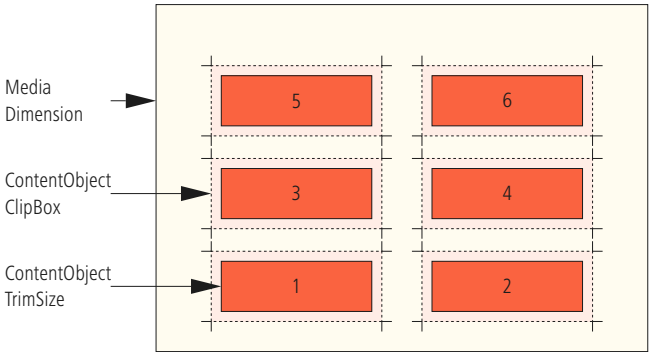


Figure 9.10
TrimSize and ClipBox as attributes of ContentObject

TrimCTM:

Positioning of a *TrimBox* on the *SurfaceContentsBox.*

Only one color bar "FOGRA_6_F74_740x10" with its position as well as two register marks are entered here as marks (*MarkObject*).

One thing that seems to continue to be important: The *Layout* resource was especially changed in the different JDF versions. As a result, at first glance the JDF descriptions from older versions appear totally different than they are presented here.

9.3 Trapping

In multiple-color offset printing, the overprinting may not end up being entirely in alignment. These registration variations mainly occur due to paper stretch, but also due to tolerances in the press. A consequence of this inaccuracy is a "misregistration," as shown in Figure 9.11. For this reason, owing to circumstances where trapping (overlapped) adjacent areas are enlarged; the general rule with translucent colors is that the lighter colors overlap the darker colors; see Figure 9.12. Then with small registration variations there is no more misregistration. The flip side of the coin is that, of course, the trap contours may be disturbingly noticeable. In this respect try to keep the trap width as marginal as possible, namely equal to the expected maximum register difference. Only when a black object is overlapped is the trap not visible and you can choose a larger trap width. With opaque colors like gold or silver, the lightness rule is no longer valid. Instead,

Figure 9.11
Misregistrations caused by register variations if no trapping exists

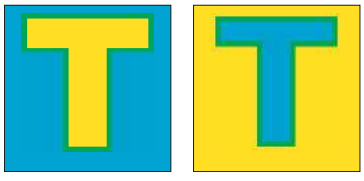


Figure 9.12
The lighter colors overlap the darker colors.

for obvious reasons, a translucent color should never overlap an opaque color, but rather the other way around. With two adjacent opaque colors, the first printed color overlaps the other.

Many trapping details, such as the trap width, must be defined. There is no universally valid value since it depends on many factors (paper, press, printing method, etc.). These *Trapping* details can be defined in the *TrappingDetails* resources which is an input resource of the Trapping process (Figure 9.13). In *ColorantControl* one finds the color assignments for the translucent and opaque colors in the print product. What should happen when the required fonts are no available (abort process, choose alternate font, etc.) is entered in the *FontPolicy* resource, and finally the *RunList* contains information about the print data. The result of the *Trapping* process is also a *RunList,* one that shows the trapped content data.

Figure 9.13
Trapping process resources

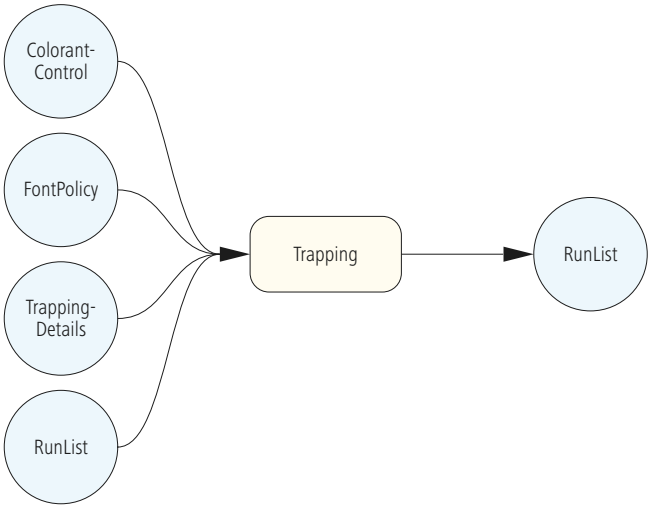


Figure 9.14 shows an example for a *TrappingDetails* resource, as they could be sent to a trapping engine. The practice shows that a trapping engine, similar to the RIP, cannot be configured independently of the WMS. In other words, the interface between the trapping engine and the prepress system is vendor-specific even if it is based on JDF.

Now we would like to explain the elements and attributes from Figure 9.14. *DefaultTrapping* can be adjusted as to whether the whole page (value equals *true*) or only certain zones (value equals *false*) are trapped. In the subelement *TrappingOrder,* the sequence is entered as to how the colors lay on top of each other—they should correspond to the print sequence. As was just explained, the sequence of opaque colors is of most importance. The many small details about trapping are stored in the *TrappingParams* element, of which we will only show a small portion.

The *BlackColorLimit* attribute specifies at which raster value a gray area is trapped according to the black color rules—here at 95% tonal value. *BlackDensityLimit* is at what certain density a color is considered analogous to black. The value of *BlackWidth* determines that the trap width for black is 1.3 times that of the trap-

ping width for the other colors. Whether adjacent images or images on top of other objects (graphics, fonts) should be trapped can be defined in *ImageToImageTrapping* and *ImageToObjectTrapping*. In the example the borders between images and objects are also trapped (value *true*); one specifies how it should happen in *ImageTrapPlacement*. *"Normal"* is chosen here, which mean that the usual rules apply, which are based on the lightness of the translucent colors and the print sequence order of the opaque colors. One can also choose, as in the example, whether the object is basically overlapped into the image (value equals *Choke*). A color separation should already have a significant tonal value difference with its two neighboring areas and is trapped there. Otherwise, it could even happen, in extreme cases, that a gradient would be trapped. In *StepLimit* the limit is set. Finally, the trap width is declared in *TrapWidth*. The value is given in DTP points.

9.4 RIPing and Platemaking

The production system has the task of resolving the *GrayBoxes* which were written by the MIS into individual processes like *PlateMaking* (see Figure 6.17) or *ImpositionRIPing,* and *PlateSetting* (see Figures 9.4 and 9.5) into one or more combined processes. Of course, combinations of the three possibilities are also allowed. The processes that arise are, in principle, put together like puzzle pieces in which an output resource of one process is the input process of the next process and so on. These are mostly handled as *RunList* resources; thus information about the intermediate re-

Figure 9.14
Excerpt from the TrappingDetails resource

```
<TrappingDetails Class="Parameter" DefaultTrapping="true" ID="_333"
 Locked="false" Status="Available">
   <TrappingOrder>
      <SeparationSpec Name="Black" />
      <SeparationSpec Name="Cyan" />
      <SeparationSpec Name="Magenta" />
      <SeparationSpec Name="Yellow" />
      <SeparationSpec Name="Gold-Spot color" />
   </TrappingOrder>
   <TrappingParams
    BlackColorLimit="0.95"
    BlackDensityLimit="1.6"
    BlackWidth="1.3"
    ImageToImageTrapping="false"
    ImageToObjectTrapping="true"
    ImageTrapPlacement="Normal"
    StepLimit="0.25"
    TrapWidth="0.25"
    ... />
</TrappingDetails>
```

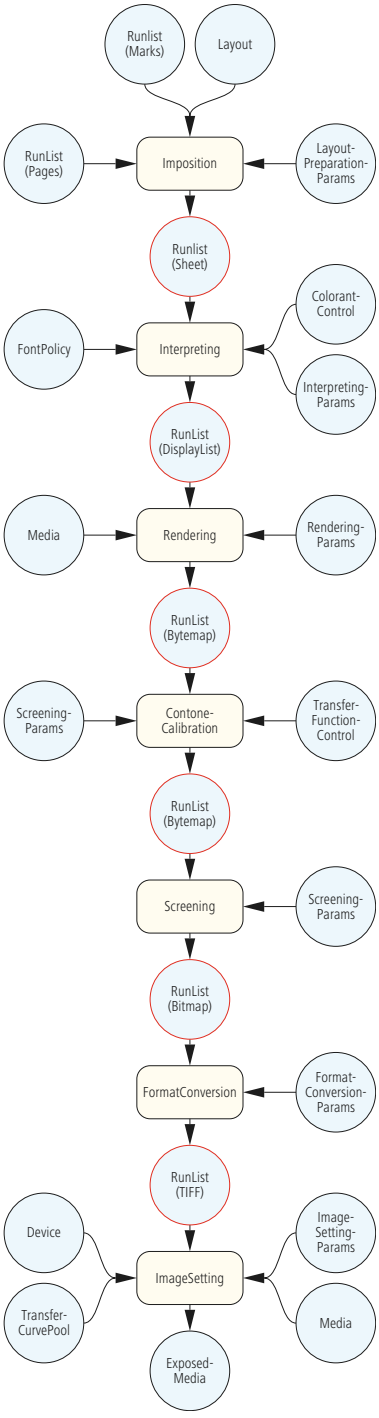


Figure 9.15
JDF model for platemaking

sults' data. It should be noted, however, that with combined processes these intermediate results may also be missing. Some of these puzzle pieces were already presented like *Interpreting, Rendering* and *Screening* (see Figure 3.10); others were at least mentioned, like *Imposition* and *ImageSetting* (Section 6.1). In Figure 9.15 a possible structure for the RIP process and for plate imaging are presented fully again. We will only briefly touch on the previously unhandled processes.

The *ContoneCalibration* process is responsible for the application of the transfer curve at the RIP. It contains a bytemap for each separation from the *Rendering* process, thus a halftone (all tonal values between 0% and 100%) in each process color or also spot colors. The outcome of the process is a data structure in the same form, but in this case only the tonal values would be changed. How the tonal values are changed is specified in the *TransferFunctionControl* resource. The description of the transfer curves is similar to the description in PPF (see Figure 4.11). In fact, more curves per separation can also be used here. So the plate output is often linearized with the help of a transfer function, which means it affects a tonal value adjustment that produces no or minimal difference between the input tonal values of the RIPs and the tonal values of the developed plate. Besides the linearization curve, which is only dependent on the platesetter, the plate and, under the circumstance the developing process, there is still the actual transfer curve for the press, which is dependent on many parameters like the output screen, the substrate, the printing press, the press conditions, and the colors. Both transfer curves are calculated against each other and run.

The *FormatConversion* process does exactly what its name says: it converts one data format to another. Here a vendor-specific bitmap is typically converted into a TIFF file, which is the input for the upstream control software of the CTP imagesetter. Because this data contains bitmaps, thus binary states of pixels (only black/white and no gray tones), the data format is called "TIFF-B." The format is a quasi standard between workflow management systems in prepress and imagesetters. This makes it possible to connect different imagesetters to a WMS

```
<JDF Activation="Active" ID="_001" JobPartID="_1003.I" Status="Part"
 Type="Combined" Types="Imposition Interpreting Rendering ContoneCalibration
 Screening PreviewGeneration FormatConversion ImageSetting" Version="1.3">
...
    <ResourceLinkPool>
        <RunListLink CombinedProcessIndex="0" ProcessUsage="Document"
         Usage="Input" rRef="_77" />
        <RunListLink CombinedProcessIndex="0" ProcessUsage="Marks"
         Usage="Input" rRef="_21" />
        <LayoutPreparationParamsLink CombinedProcessIndex="0" Usage="Input"
         rRef="_581" />
        <LayoutLink CombinedProcessIndex="0" Usage="Input" rRef="_99" />
        <ColorantControlLink CombinedProcessIndex="1" Usage="Input"
         rRef="_83" />
        <FontPolicyLink CombinedProcessIndex="1" Usage="Input" rRef="_583" />
        <TransferCurvePoolLink CombinedProcessIndex="6" Usage="Input"
         rRef="_71" />
        <InterpretingParamsLink CombinedProcessIndex="1" Usage="Input"
         rRef="_85 " />
        <RenderingParamsLink CombinedProcessIndex="2" Usage="Input"
         rRef="_86" />
        <MediaLink CombinedProcessIndex="2" ProcessUsage="Paper" Usage="Input"
         rRef="_79" />
        <ScreeningParamsLink CombinedProcessIndex="3 4" Usage="Input"
         rRef="_89" />
        <TransferFunctionControlLink CombinedProcessIndex="3" Usage="Input"
         rRef="_34" />
        <ImageSetterParamsLink CombinedProcessIndex="6" Usage="Input"
         rRef="_94" />
        <DeviceLink CombinedProcessIndex="6" Usage="Input" rRef="_286" />
        <MediaLink CombinedProcessIndex="6" ProcessUsage="Plate" Usage="Input"
         rRef="_3085" />
        <FormatConversionParamsLink CombinedProcessIndex="5" Usage="Input"
         rRef="_47" />
        <ExposedMediaLink CombinedProcessIndex="6" ProcessUsage="ExposedMedia"
         Usage="Output" rRef="_81" />
    </ResourceLinkPool>
...
</JDF>
```

Figure 9.16
Platemaking combined processes

without much effort. The result is that the imaging of plates is often completely separated from the WMS. The CTP imagesetter and its upstream software are then generally not JDF/JMF-capable. In this case the JDF workflow is already generating a TIFF-B file concurrently with the completion of the platesetting and issues a corresponding message. This is also explicitly allowed by the *MIS to Prepress ICS*.

Figure 9.16 presents a combined process matching Figure 9.15. However, only the *ResourceLinkPool* is depicted, not the *ResourcePool*. Notice that the red-outlined *RunList* resources in Figure 9.15 are not to be found in the *ResourceLinkPools* links, nor are the re-

```
<FormatConversionParams Class="Parameter" ID="_47" Status="Available">
    <FileSpec Class="Parameter" MimeType="application/octet-stream"
     ResourceUsage="InputFormat" />
    <FileSpec Class="Parameter" MimeType="image/tiff"
     ResourceUsage="OutputFormat" />
</FormatConversionParams>
```

Figure 9.17 FormatConversionParams resource

sources themselves in the ResourcePool which is permitted for combined processes. By way of example, the resource *FormatConversionParams* is only shown in Figure 9.17. There you can see that a related process should produce a TIFF format image out of an application-specific byte stream.

With the help of a similar JDF model the concept of a digital printing press can be taken on. In practice, however, it is common that digital print orders are collected in the front end of the digital printing press. Then someone, namely a machine operator, can intervene before printing in order to, for example, sort the orders according to paper type.

We also want to end this section with a general word of advice: as mentioned earlier, the JDF communication between the comprehensive WMS, the RIP, and a plate-setter is commonly not realized with truly open interfaces. This means a manufacturer may use JDF (and also other communications options such as databases), but only with the goal (quite legitimate) of controlling their own software modules with it. Third-party software is not able to be bolted on here. To this extent, the result of our research is not very surprising, through it we have ascertained that JDF is certainly employed "creatively" here. In particular one would find combined processes at this point in which the intermediate results of the process need not be performed. Then why, for example, should the storage positions of the intermediate results be entered in *RunList* resources, if the WMS has a set procedure and the jobs are deposited in a set folder or database structure or the information lies only in memory (RAM)?

9.5 Proof and Press Approvals

A proof should depict, in advance, as accurately as possible, the result of the print run. Different proof methods are used:

- Color proof, color binding proof, contract proof (digital)
- Imposition proof, also known as form proof
- Soft proof

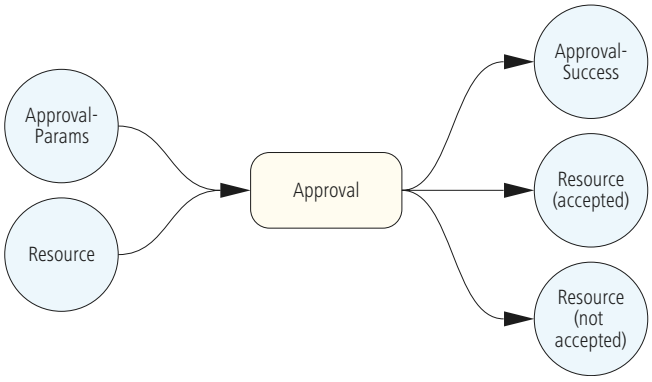
- Screen proof
- Machine proof or contact print (found, if ever, in gravure printing or even flexographic printing)
- Analog proof (based on film)

The first three technologies especially play an important role here, while the last two are only seldom employed today.

The creation of a proof is a part of comprehensive quality management because, naturally, other things are also checked such as printing plates, press sheets from the press run, or components of the print product. The purpose of this is to undergo a certain acceptance procedure by which a customer/client or also an internal employee decides if the test piece is acceptable or must be rejected. Resources for ongoing production are approved or rejected in the JDF language. Such resources typically, in this section in the book, are *ExposedMedia,* which represent the hard copy of a color proof or form proof, or also *RunLists*, which depict the image data for soft proofing.

Figure 9.18 illustrates this situation once again. Typically you would have one or more resources that should be approved. The *ApprovalParams* resource contains detailed information about the approval process, namely in the nitty gritty of who can issue the approval (described by the subelement *Contact*) and which roles this person or people take up there, for example, who can overwrite another person's approval and the like. The output of the *Approval* process consists of the approved, or as the case may be, unapproved, resources. The *Status* attribute has the value of *Draft* before the *Approval* process, followed by either a status of *Available* or *Rejected.* It is then noted in *ApprovalSuccess* exactly who signed off or rejected the proof, possible comments on the proof, and, if appropriate, a link to the file which contains the signature of the approving person.

Figure 9.18
Approval process in JDF



The *Approval* process can then be defined as the last stage of a combined process. A hardcopy proof from a digital printer is therefore a combined process, which ends with the processes *"Digital-Printing Approval,"* as is seen

in Figure 9.19. For a machine proof, the combined process would include *ConventionalPrinting* and *ImageSetting* instead of *DigitalPrinting.* The processes *ConventionalPrinting* as well as *DigitalPrinting* will be discussed in more detail in the following chapter.

Figure 9.19
Approval process for a proof

```
<JDF Category="DigitalPrinting" ID="_1" JobID="_2" Status="Ready"
 Type="Combined" Types="LayoutPreparation Imposition Interpreting Rendering
 ImageSetting Approval"...>
...
</JDF>
```

As a general rule for color proofs, but also of course for platesetting, color space transformations are required. Usually a color profile is defined which describes the color characteristics of devices. First, device-specific RGB data are available for the input data (digital camera, scanner) which must be converted into device-independent color values with the help of a profile. A second profile that describes the printing process is then still needed for the production of print forms. Yet another profile is required for the production of a color proof, namely the proofer profile.

The color space transformation takes place in JDF through the *ColorSpaceConversion* process, which typically occurs before the *Trapping Process.* The most important input resource is the *ColorSpaceConversionParams,* in which the location of the color profile is entered. Example 9.20 shows one such resource with its elements and attributes are explained as follows:

Figure 9.20
Information on color space transformation in a soft proof

```
<ColorSpaceConversionParams ColorManagementSystem="ADBE" ID="R6"
 Class="Parameter" Status="Available">
   <ColorSpaceConversionOp SourceCS="Gray" SourceObjects="All"
    Operation="Untag" />
   <ColorSpaceConversionOp SourceCS="RGB" SourceObjects="All"
    IgnoreEmbeddedICC="false" RGBGray2Black="true" RGBGray2BlackThreshold="1"
    Operation="Tag" RenderingIntent="Perceptual" >
      <FileSpec ResourceUsage="SourceProfile"
       URL="file://Server1/ICCProfiles/sRGBprofile.icm" />
   </ColorSpaceConversionOp>
   <ColorSpaceConversionOp SourceCS="CMYK" SourceObjects="All"
    Operation="Untag" />
   <FileSpec ResourceUsage="FinalTargetDevice"
    URL="file://Server1/ICCProfiles/CoatedFOGRA39.icc" />
</ColorSpaceConversionParams>
```

ColorManagement System:

Here the preferred vendor-specific system is defined for color space conversion ("Adobe").

FileSpec:

In this attribute the ICC profile defining the target values are entered.

ColorSpaceConversionOp:

The operations for color space transformation are described in more detail, where

- In SourceCS the output color space is established.
- In *SourceObject* the object type is specified as to which operation should be performed like *Text, LineArt, ImagePhotographic,* etc., or also *All* if all Objects are affected.
- In *Operation,* whether the profiles should be attached (*tag*) to the graphic elements or instead the attached profile should be removed (*untag*) is defined.
- *IgnoreEmbeddedICC* specifies whether embedded profiles should be ignored (*true*) or not (*false*).
- In *RGBGray2Black* it can be specified whether RGB gray should be (*true*) mapped to the black channel K with a CMYK conversion, or not (*false*), where in the case of true
- The attribute *RGBGray2BlackThreshhold* can limit the tonal value dependent process: with a value of 0, only the RGB black is in full tone; with a value of 0.5, all RGB gray tonal values go to 50%; with the value of 1 all RGB graytones are mapped to the K channel.
- In *RenderingIntent* the priority for the usual gamut mapping is determined (conversion from one color space to another).

In the example, all of the color profiles related to the CMYK or gray color space were therefore removed from the graphic elements, and all of the RGB elements were associated to a specific profile, provided that they do not yet have their own. The color space transformation describes a process for offset platemaking.

Exercise:

Draw an imposition layout with the data from Figure 9.9. Note the definition of the transformation matrices as shown in Figure 9.21. First, each point (x,y) in 2-dimensional space is formally written as 3-dimensional point (x,y,1) (to also allow for shifts). The transformation matrix is then consequently a 3 × 3 matrix where the right column has the fixed values 0,0,1. Therefore, these three values are always the same and not always incorporated in the **Current Transformation Matrix** (CTM) and there are only 6 values left. After the matrix multiplication, the third dimension can be removed and one can see that, for example, the CMT = "1 0 0 1 a b" really causes only a shift of coordinates around the value (a,b).

Figure 9.21
Current transformation matrix

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{21}y + a_{31} \\ a_{12}x + a_{22}y + a_{32} \\ 1 \end{pmatrix}$$

CTM = „a_{11} a_{12} a_{21} a_{22} a_{31} a_{32}"

CTM = „1 0 0 1 a b" corresponds to the offset of (x,y) at (x+a, y+b)

10 Press

At first glance it is paradoxical that there are only three different processes in the area of print: *ConventionalPrinting* (printing with physical press forms such as offset, flexo, gravure, and screen printing), *DigitalPrinting,* and *Varnishing*. In comparison, there are many more processes in prepress and in post processing. The reason for this situation is that, in the area of print, processes are independent of each other and can be carried out in different sequences (compare to Section 3.3), which differs from prepress and post processing where the work processes are very intertwined and not isolated.

In this chapter, we will only go into the JDF nodes that concern printing. Previously, especially in Chapter 9, we handled general communication between JDF/JMF devices (in this case the control station of a printing press) and Agents and Controllers (either MIS or production's workflow server). So that the new JDF nodes do not seem to be taken so completely out of context, we will briefly recount the typical communication scenario again: where a responsible Agent/Controller drops a JDF file in a hot folder for the printing press, which is regularly polled by the JDF module of the printing press. Alternatively, the Agent/Controller sends a JDF command (*SubmitQueueEntry*) to the control station, which, in turn, uses the included URL to download the JDF file from a file server which both parties have access to. The JDF file contains not only the usual things like customer name (*CustomerInfo*) and order number (*JobID*), but also all of the resource descriptions which are required for the printing process and will be discussed in the following sections. For data collections the Agent/Controller must request a persistent communication channel (*Persistent Channel*) with the control panel of the printing press either by means of JDF (*NodeInfo*) or with a JMF message (*Subscription*). It will "build up" the channel, confirm it with a JMF Response, and periodically send the desired JMF signals over HTTP to the Agent/Controller. This is done until the Agent/Controller sends a command to indicate that it now wants to stop receiving the signal (*StopPersistentChannel*). The signals are neither confirmed by the Agent/Controller nor cached by the control panel, which is referred to with a seemingly militaristic yet apt expression: "fire and forget." During the printing process, the JDF control station further writes its audit log as *Audit* elements in the *AuditPool* of the JDF file. Once the job is complete, the printing press control system finally returns the modified JDF file back to the Agent/Controller, which again can happen via special hot folder or using a JMF message.

10.1 Conventional Printing

Some conventional printing presses allow the processes of inline prepress or finishing, such as digital offset machines which can create their own plates or web offset machines which can also fold and cut. However, the JDF process *ConventionalPrinting* does not include such processes, and one must form combined processes with processes from prepress or from postpress. For digital printing it would be the combined process of *ImageSetting* and *ConventionalPrinting.* With web offset printing it would be the combination of both processes *ConventionalPrinting* and *WebInlineFinishing*.

Many parameters for the areas of printing and postpress are defined while in prepress; comparatively printing has very simple interfaces. It receives specifications from MIS and prepress, and outside of press sheets and status updates of its activities, it produces nothing for them. Therefore, it is not surprising that the *ConventionalPrinting* process has many (optional) input resources, but only one output resource.

Which type of information, specifications, and preparations from other departments (above all, MIS or prepress) are meaningful for the printing department? Primarily, regardless of whether they receive this data by means of JDF or in any other form:

JDF Systems for Offset Printing (2009)

Hersteller:	***Product:***
Akiyama	InkZone
Goss International	Goss Web Center
Heidelberger Druckmaschinen	Prinect Press Manager
manroland	printnet PressManager Sheetfed
Komori Lithographic Presses	K-Station
König & Bauer AG (KBA)	JDFLink for Logotronic
Mitsubishi Heavy Industries	IPC Server II
Ryobi Druckmaschinen	MIS Connection Software
Shinora Machinery Company	Shinora CIP4 Center / Station
Sakurai	InkZone Pefect

1. Printing forms
2. Ink
3. Information about the substrate (size, paper grade, etc.)
4. Determination about perfecting printing
5. Determination about the colors for the job
6. Administrative data (order number, customer, acceptance conditions, circulation, delivery terms, etc.)
7. Ink zone settings for offset printing
8. Proof (form proof, contract proof, prototype proof, preview, soft proof)
9. Positions of the control elements on the printed sheet
10. Determinations of the press conditions (Lab values or density of the spot colors, dot gain, ICC profile targets, etc.)

Only the first six points in this list are actually obligatory, the next two are widely used at least, but the last two play a special role. The positions of the control elements, in many cases, are actually determined by the printer and mandated according to the printing press and the press suppliers. The inline or the offline measuring systems can read and interpret the control elements only in certain places. Conversely, only a few modern machines can also receive the positions of the control elements as metadata from the printing form setup and position the measuring instruments accordingly. Also, the specification of the press conditions normally occurs in the pressroom, at least in the standard world of offset printing.

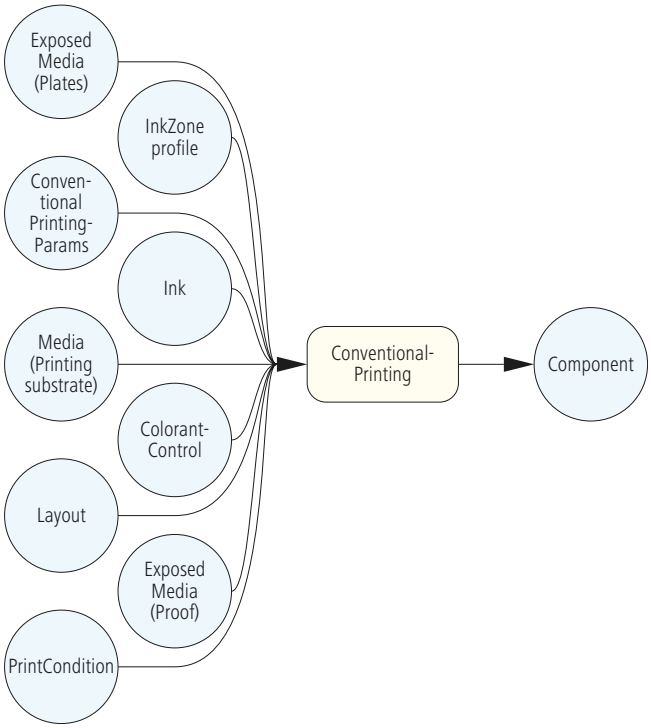
In point 6 we have also included the deadlines for delivery of the finished products, which is usually set with the incoming order. These still need to be converted by planners in shift planning for individual work steps during production. We want to address the following three possibilities:

- Scheduling is not based on JDF, but instead, for example, with a planning board or with software which sends deadline lists to the machine control panel over a vendor-specific protocol.
- The planning of the order sequence is performed in an MIS module, and the scheduler sends the technical and administrative JDF order dates under MIS control directly to the machine control panel.
- One of the material requirements planning departments assigned by production receives the key data which are necessary for machine scheduling from the MIS.

The first case is not of interest for our subject. In the second case the MIS sends the JDF job data in correct order straight to the control panel of the press, while in the third case the JDF goes first to the production scheduling software. The JDF from MIS, therefore, has different addressees and also different content.

In Figure 3.12 we have already presented the main inputs and outputs for offset printing. In Figure 10.1 we show the same picture again, but now with the names of the JDF resources which we have primarily introduced in the last chapters. We have not yet listed some possible JDF input resources because they are only necessary in very special situations. For example, if preprinted shells are processed, a *Component* resource is required at input. Or if double-sided adhesive tape should be specified in flexographic printing, which is required for the assembly of the cylinder or sleeve, an additional *Media* resource of the type *MountingTape* is required. Other inputs, like the *Preview* resource, are quite widespread in order to depict the preview of the sheet at the control panel of the printing machine. We will illuminate the different resources of the *ConventionalPrinting* process in more detail, even some of those that are not listed in Figure 10.1

Figure 10.1
Offset print with its resources



Print Parameters and Print Conditions

In the JDF model there is a distinction between print parameters (*ConventionalPrintingParams*) and print conditions (*PrintCondition*). There are some basic parameters for the setup of the printing press lodged in the obligatory resource *ConventionalPrintingParams.* Here the type of drying (UV, IR, heatset) can be given, determining the deactivation of the dampening system for waterless offset, or designating the maximum speed in sheets per hour for sheetfed offset or the revolutions per hour for web offset. Even the printing processes (offset, flexo, gravure, or screen printing) and the print-

ing press type (sheetfed, web with one or more plates per cylinder, etc.) are specified. This very general information is usually set to a fixed house standard or is implicitly indicated by the fact that a job is assigned to a particular machine. In this respect the attributes are not, in reality, playing a very large role. The usefulness is seen rather in connection with the query of devices about the capabilities of the associated machine. The so-called *DeviceCapabilities* will be presented at the end of Section 11.2. There are also order-related parameters for a printing press which are stored in this resource, such as *WorkStyle* —in which the type of perfecting printing is specified. In Figure 10.2 one sees that *WorkAndTurn* is entered as a value, which is often called "to turn" (in perfecting, the front arrangement is the same as the back, which the side arrangement changes; no plate change for the reverse printing). In the resource it could also be written that a customer or an internal quality assurance representative may give approval to print to the press. The status of the process *ConventionalPrinting* is then set at *Stopped* and the status details are at *WaitForApproval* until approval has given.

Figure 10.2
ConventionalPrintingParams resource

```
<ConventionalPrintingParams Class="Parameter" ID="_400"
 PrintingType="SheetFed" Status="Available" WorkStyle="WorkAndTurn" />
```

While *ConventionalPrintingParams* has very basic values available for the basic setup of a printing press, set points for color, density, and dot gain for the press are in *PrintCondition*. You can also specify spectrophotometric color measurements here (color temperature, filters, 2-degree or 10-degree viewing angle, measurement of wet or dry color, measure, measurement base, etc.). In practice this resource has been little used in offset printing in general since ISO Standard 12647 [24] was printed and so far no order-related definition is required. Fig-

Figure 10.3
SOLL curves for the dot gain in print

```
<PrintCondition Name=" Paper type_1_ISO_12647-2" Class="Parameter" ID="_4711"
 PartIDKeys="Side Separation" Status="Available">
   <PrintCondition Side="Front">
       <PrintCondition Separation="Cyan"
        AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
       <PrintCondition Separation="Magenta"
        AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
       <PrintCondition Separation="Yellow"
        AimCurve="0.0 0.0 0.4 0.56 1.0 1.0" />
       <PrintCondition Separation="Black"
        AimCurve="0.0 0.0 0.4 0.53 1.0 1.0" />
   </PrintCondition>
</PrintCondition>
```

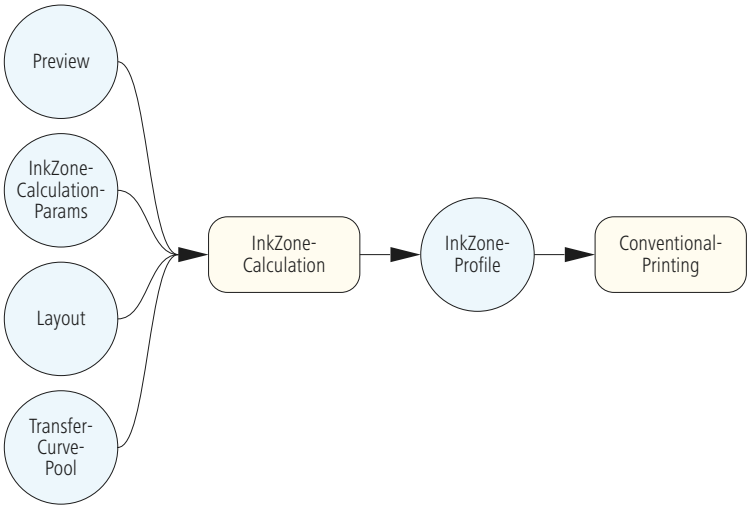
ure 10.3 shows but one example where the target curve for the dot gain was entered. The values to interpret are the same as Figure 4.11.

Presetting Ink Zones with Offset Printing

Classically, the ink supply of the ink fountain is adjusted to the print subject on an offset press (exception: anilox color inking units, also called short inking). For this purpose, individual color zones are defined across the width of the print. In each of these zones, an ink zone key, ink blade, or ink knife presses on the ink fountain so that only a defined amount of ink can flow. In general, the color-allotting slot is controlled electronically. The ink zone presetting is done as part of setting up a form.

As we have already mentioned in Section 4.2, the calculation of the ink presetting values using prepress data is already in Portable Print Format (PPF), the most widespread application. In fact, it is so common that in many cases PPF continues to be used and not JDF. Therefore, we must admit that the functionality of JDF is not superior. In this respect, switching from one file format to another has, at minimum, no advantages to the user. Overall, however, a JDF implementation has a structural advantage because these values can then be packaged and forwarded with all of the other values in a JDF file and no special handling is required. More important, it eliminates the need to license a PostScript interpreter in order to interpret the PPF file.

Figure 10.4
Calculation of the ink zone presets in offset



The calculation of the ink zone settings is seen as a JDF model in Figure 10.4. The *Preview* resource allows access to a preview image (usually only around 50 ppi) of the entire sheet. Because, simply put, the ink zone presetting must only count the pixels per separation and zone in the preview. In *InkZoneCalculationParams* the individual zones are defined, which have

```
<InkZoneCalculationParams Class="Parameter" ID="_25" Status="Available"
 ZoneHeight="1451.338583" ZoneWidth="92.125984" Zones="23" ZonesY="1" />
```

Figure 10.5
Ink zone parameters

different widths on various press types. In Example 10.5 the zone width is 3.25 cm (92.125984 / 72 × 2.54 cm) and there are a total of 23 zones parallel to the cylinder axis. It is formally correct that there is only one zone in the direction of the paper run and its height is equal to the maximum printing format in the unwinding direction, although the entry itself seldom appears. The zone positions are specific to the printing press; the preview image shows only the sheet, however. In order to be able to calculate the ink zone settings, the positions of the sheet with respect to the zones must be provided. This happens most simply by defining the position of the sheet on the plate during printing. You can find this entry in the attribute *SurfaceContentsBox* as a *Layout* resource, which has already been covered several times in this book. And finally, there are also possible transfer curves required in order to be able to carry out the tonal value changes with RIPing of the exposure data and also with the preview image (see Section 4.2).

In JDF the ink zone preset values are stored in the *InkZoneProfile* resource. Of course, the settings are not given in micrometers as ink-allotting slot openings, because that would be far too machine-specific. Instead, each of the preset values is a number between 0 and 1, where these represent the average surface coverage in a zone. One must multiply this factor by 100 in order to arrive at a percent value. From this, the calculation of the opening of the ink-feeding slot occurs with the help of an adaptation curve, which is determined at the press for the specific print conditions (ink, paper) and is no longer part of the JDF description. For an unchanged printed image, because of the same ink preset values, the ink supply must be higher on a natural paper than an art paper. These differences are identified on the press, stored in a file, and reused for the respective print order. Thus, the color work, without having printed even a single sheet yet, is adjusted to a substrate. Thus, so the control components always use the correct data for the corresponding printing conditions, they should always grab this information out of the JDF order description. In the event they have incorrect values (because they were not adjusted for a specific order) or if there are none at all, this leads to several rule cycles (copy making, sheet pulling, measuring and evaluating, readjusting, making new copies, etc.) having to be being carried out, which, in turn, requires more time and more makeready sheets.

```
<InkZoneProfile Class="Parameter" ID="_3" PartIDKeys="SignatureName SheetName
 Side Separation"...>
   <InkZoneProfile SignatureName="S1">
      <InkZoneProfile SheetName="Cover">
         <InkZoneProfile Side="Front">
            <InkZoneProfile Separation="Black" Status="Available"
             ZoneWidth="92.125984" ZoneSettingsX="0.004947916666669755
             0.008625565610862806 0.040955175339369494
             0.05258389894419598 0.036772011689294073
             0.04679616798642827 0.03305889423077225
             0.04669306184012362 0.023023897058826546
             0.03196920955882654 0.026367305335598803
             0.022763303638766233 0.01726845305430168
             0.027979296285825045 0.03681620003771036
             0.0802906414969863 0.07557833710407524
             0.07194204845399976 0.05891538225867557
             0.08456872171945984 0.08174302413273286
             0.029526241987182487 0.009106099170440482"
             ZoneHeight="1451.338583"
             ZoneSettingsY="0.04123003711309234" />
            <InkZoneProfile ... />
          ...
         </InkZoneProfile>
      </InkZoneProfile>
   </InkZoneProfile>
</InkZoneProfile>
```

Figure 10.6
Ink zone preset values

In Figure 10.6 example code for an *InkZoneProfile* can be seen. The resource is partitioned up to the separation because the ink zone preset values are generally different for each color. In the example only the values for the color black are given. The 23 values in the attribute *ZoneSettingX* give the coverage values for each zone; thus they begin on the left and are limited to two digits after the decimal point: 0.49%, 0.86%, 4.09%, 5.25%, etc. The *ZoneSettingY* contains the average value of the coverage area for all zones, in this case 4.12%.

Control Marks

Typically several control elements are burned on press forms. They can be categorized according to function:

- Control elements for the production of press forms
- Control and monitoring elements for printing
- Control and monitoring elements for postpress

In this chapter, only the control elements for print are relevant. The most important are:

- Registration or register marks
- Print control strips
- Pages or equipment marks, also known as push or pull marks
- Press sheet signatures

Naturally, marks should be placed in certain areas. In sheet printing the side marks are placed exactly on the sheet edge, the print control strip is placed parallel to the cylinder either in the middle or also on the end of the sheet, and so forth. However, unless the marks are also printed in order to be visually checked later by an employee or measured with a handheld device, the exact position for the verification process is unimportant. Only when automatic inline or automatically controlled offline measuring systems are used is the knowledge about the exact placement of the control elements important, so that the sensors can actually adjust themselves to them. And there are more and more such measuring systems: there are inline measuring systems for the color application, as measured in density, for offset, flexo, and gravure printing. At a minimum there are also spectrophotometric-based inline control systems. Often printed register marks are also read by inline CCD cameras and analyzed with the purpose of controlling the circumferential, lateral, and diagonal register.

A prepress worker sets the position of the control elements on the imposition layout in the assembly program. With "dynamic" marks as we have seen in Section 9.2, the exact placement is calculated only by software. The positions of the marks therefore represent a typical example by which information has to be passed across departments. This task can be taken on by JDF, and in Figure 10.7 shows several resources of the *MarkObject* type which are embedded in a *Layout* resource.

Press Forms and Substrates

There are two states of press forms (plates, gravure cylinders, sleeves, or screens): blank press forms and finished press forms which hold the press image information. In offset you speak in shorthand about exposed and unexposed plates where the term "imaged" would be correct since not all plates are sensitive in the

```
<Layout Class="Parameter" DescriptiveName="Lay" ID="_111" Name="Lay"
 PartIDKeys="SignatureName SheetName Side" Status="Available">
   <Layout Name="SIG1" SignatureName="SIG1">
      <Layout DescriptiveName="Sheet" Name="Sheet" SheetName="Sheet"
       SourceWorkStyle="WorkAndTumble" Status="Available"
       SurfaceContentsBox="0 0 2111.81102362 1714.96062992">
         <Media Class="Consumable" Dimension="2111.81102362 1714.96062992"
          ID="_3062" MediaType="Plate" Status="Available" />
         <Layout DescriptiveName="Recto" Side="Front"
          Status="Available"
          SurfaceContentsBox="0 0 2111.81102362 1714.96062992">
         ...
            <MarkObject CTM="1 0 0 1 0 0"
             ClipBox="0 0 2111.81102362 1714.96062992" Ord="0">
               <RegisterMark Center="2028.42519684 659.11417322"
                Class="Parameter" MarkType="8AR_R_106x164" Rotation="0" />
               <RegisterMark Center="83.38582677 659.11417322"
                Class="Parameter" MarkType="8AR_L_106x164" Rotation="0" />
               <ColorControlStrip Center="1059.44881889 809.11811023"
                Class="Parameter" Rotation="0" Size="1798 28"
                StripType="FOGRA_5_F74_740x10" />
         </MarkObject>
      </Layout>
   </Layout>
</Layout>
```

Figure 10.7
Marks object in a *Layout* resource

visible spectrum. The possibility of plate development here is simply added to the exposure process. Unexposed plates are described in the JDF resource *Media,* where the attribute MediaType must be the same as the *Plate.* Substrates are defined with the same resource. The type of substrate must then be specified under *MediaType,* such as *Paper, Foil,* or *CorrugatedBoard.*

For the printing process, however, exposed plates are required (once you have disregarded uncoated dummy plates in newspaper printing). These are listed in the *ExposedMedia* resource. In practice, print shops use only one plate type, and the size of the plate is statically defined for each printing press. The printer is actually interested in whether the exposed plates are available for an order. There is also automation which relays the order to the press automatically when all of the sheet's plates or forms are available. This exact information can be seen in Figure 10.8. You will also recognizes that there is a reference to the *Media* resource in which the plate type and the plate size are specified.

```
<ExposedMedia Class="Handling" ID="_123" PartIDKeys="SignatureName SheetName
 Side Separation"...>
    <ExposedMedia SignatureName="Signatur_1">
        <ExposedMedia SheetName="Cover" Status="Available">
            <ExposedMedia Side="Front" Status="Available">
                <ExposedMedia Separation="Cyan" Status="Available" />
                <ExposedMedia Separation="Magenta" Status="Available" />
                <ExposedMedia Separation="Yellow" Status="Available" />
                <ExposedMedia Separation="Black" Status="Available" />
        </ExposedMedia>
            <ExposedMedia Side="Back" Status="Available">
                <ExposedMedia Separation="Cyan" Status="Available" />
                 ...
            </ExposedMedia>
        <MediaRef rRef="_1234">
            <Part SheetName="Cover" SignatureName="Signatur_1" />
        </MediaRef>
    </ExposedMedia>
</ExposedMedia>

<Media Brand="745x605_Azura" Class="Consumable" Dimension="2111.811
 1714.961" ID="_1234" MediaType="Plate" PartIDKeys="SignatureName SheetName"
 Status="Available">
    <Media Class="Consumable" Dimension="2111.81102362 1714.96062992"
     SignatureName="Signatur_1" Status="Available">
        <Media Brand="745x605_Azura" Class="Consumable"
         Dimension="2111.811 1714.961" MediaType="Plate" SheetName="Cover"
         Status="Available" />
    </Media>
</Media>
```

Figure 10.8
Plate information for the printing process

We find in the JDF code in Figure 10.9 a bit about the substrate for the cover in the partitioned resource *Media*.

- Sheet size is 61 × 43 cm (*Dimension*)
- Paper Class 1 according to ISO 12647-2:2004 (*Grade*)
- Travel direction parallel to the long edge of the paper (*GrainDirection*)
- Paper thickness of 215 micrometers (*Thickness*)
- Paper Weight of 215 g/m² (*Weight*)
- Glossy front side coating (*FrontCoatings*)
- Matte back side coating (*BackCoatings*)
- Cromolux brand (*Brand*)

```
<Media Class="Consumable" ID="_111" MediaType="Paper"
 PartIDKeys="SignatureName SheetName" Status="Available"...>
   <Media Class="Consumable" SignatureName="Signature_1" Status="Available">
       <Media SheetName="Cover" Class="Consumable" ProductID="_999"
        Status="Available" Dimension="1729.1338582677165 1218.8976377952756"
        Grade="1" GrainDirection="LongEdge" Thickness="215.0" Weight="215.0"
        FrontCoatings="Glossy" BackCoatings="Matte" Brand="Chromolux" />
   </Media>
</Media>
```

Figure 10.9
Substrate information

However, additional attributes could be added to this resource like the Lab value of the paper or the degree of opacity. Unfortunately, these values are never (or rarely) provided by the paper supplier (for example, in the paper price list).

Colors and Color Control

Different specifications can be created under the term "color": the determination of the color per sheet side, the color sequence of the press, the colorimetric properties of the printing inks, the color names in the content data, the color model of the output device, and so on. This information will be stored in different resources in the JDF.

The printing inks are listed in the Ink resource. In the simplest case, these are only the process colors as seen in Figure 8.15. There is the front side CMYK, and the back side is only one color, black. Of course, special inks or varnishes can also be specified. However, the separation's names should correspond to the ink names in the *ColorPool* resource. It lists all of the colors which apply to the order, even those, under the circumstances, that are not printed on the printing press. Here the colors are more precisely specified, for example concerning the *ColorType,* whether opaque, transparent, or translucent (*Normal*) colors or also the neutral density of the colors (*NeutralDensity*). We have already seen in the prepress chapter (Chapter 9) that such color properties are important for the *Trapping* process. Also Lab-values of the colors or the CMYK representation of the colors—as well as special colors—may be entered here. Figure 10.10 provides a simple example of a *ColorPool* resource. The CMYK value of a color corresponds to the (approximate) tonal value of CMYK as a percent, but normalized to 1 instead of 100 as usual. With one exception, the *ColorType* in the example is always *Normal,* thus translucent. The exception color is only being used to outline the proof sheet. In particular this color is not allowed to be trapped with the other colors, no correspond-

```
<ColorPool Class="Parameter" ID="_500" Status="Available">
    <Color CMYK="1.0 0.0 0.0 0.0" ColorType="Normal" Name="Cyan"
     NeutralDensity="0.61" />
    <Color CMYK="0.0 0.0 0.0 1.0" ColorType="Normal" Name="Black"
     NeutralDensity="1.7" />
    <Color CMYK="0.0 1.0 0.0 0.0" ColorType="Normal" Name="Magenta"
     NeutralDensity="0.76" />
    <Color CMYK="0.0 0.0 1.0 0.0" ColorType="Normal" Name="Yellow"
     NeutralDensity="0.16" />
    <Color CMYK="1.0 0.0 1.0 0.0" ColorType="DieLine" Name="ProofColor" />
    <Color CMYK="0.2 0.3 0.4 0.5" ColorType="Normal" Name="PANTONEDeepBlue
     Lab="20. 30. 40."">
</ColorPool>
```

Figure 10.10
ColorPool resource

ing plate may be exposed, and of course it is not allowed to be printed on the printing press.

As a last resource that has to do with color definitions, we will once again list the *ColorantControl* resource (as already done in 6.4 with Figure 6.18, 8.2 with Figure 8.12, and 9.1 with Figures 9.8 and 9.13). These contain all of the information they need in order to map the colors of the content data to the colors of the output device. This information is completely irrelevant for a printing press, of course. Nevertheless this resource is necessary in the printing process because a subelement, namely *DeviceColorantOrder,* specifies the color sequence in the printing press, typically black, cyan, magenta, and then yellow in sheetfed offset (Figure 10.11).

Figure 10.11
ColorantControl resource

```
<ColorantControl Class="Parameter" ID="_2201" PartIDKeys="SignatureName
SheetName Side" ProcessColorModel="DeviceCMYK" Status="Available">
    <ColorPoolRef rRef="_500" />
    <DeviceColorantOrder>
        <SeparationSpec Name="Black" />
        <SeparationSpec Name="Cyan" />
        <SeparationSpec Name="Magenta" />
        <SeparationSpec Name="Yellow" />
        <SeparationSpec Name="PANTONEDeepBlue" />
    </DeviceColorantOrder>
     ...
</ColorantControl>
```

Proof and Preview

As a rule, the printer receives a proof, usually a form proof, which was output from a non-true-color inkjet printer. Additionally, in a networked configuration, as soon as a job is selected it is also displayed at the control panel as a small preview image of the sheet. In some cases (as with the printer Y in Section 2.2) Color-accu-

rate soft proofs are passed to the control station, which must be equipped with appropriately normalized light and calibrated monitors. Also color-accurate page proofs, color-accurate sheet proofs, and previously printed prototypes are used for an approval process. Both proofs and preview images can naturally be described using resources in JDF. A hardcopy proof is usually represented through a *Component,* and a preview image or a soft proof is represented through a *RunList* as already stated in Section 9.5. In some cases (especially in the packaging sector) hardcopy proofs are produced using a platesetter or special proof devices which can process the rasterized bitmap data (raster proofs). In such cases the proofs are represented through special variations of the *ExposedMedia* resource.

Component

The output of the printing process is press sheets, an intermediate product of the overall process. Which means, translated in the JDF language, that the output of the *ConventionalPrinting* process is a *Component* with the *ComponentType* equal to *Sheet.* Yet these components are not only partitioned up to press sheet, but also divided even after the *Condition,* as seen in Figure 10.12. In the reference to the *Component,* i.e. in *ComponentLink,* these distinctions are first clear: The press sheets are divided into *Waste* sheets and *Good* sheets. And for each of the two, a desired number of sheets are indicated. With 2040 Good sheets, 364 Waste sheets are planned.

Figure 10.12
Components of the print sheet

```
<Component Class="Quantity" ComponentType="Sheet" ID="_123"
 PartIDKeys="SignatureName SheetName Condition" Status="Unavailable">
    <Component SignatureName="Signature_1">
        <Component SheetName="Cover" />
    </Component>
</Component>

<ComponentLink Amount="1" Usage="Output" rRef="_012">
    <AmountPool>
        <PartAmount Amount="2040.0">
            <Part Condition="Good" SheetName=" Cover "
             SignatureName=" Signature_1" />
        </PartAmount>
        <PartAmount Amount="364.0">
            <Part Condition="Waste" SheetName=" Cover "
             SignatureName=" Signature_1" />
        </PartAmount>
    </AmountPool>
    <Part SheetName=" Cover " SignatureName=" Signature_1" />
</ComponentLink>
```

It is generally true that in a sense an *AmountPool* represents a count of resources, which can either be used or generated by a process. Strictly speaking, however, an *AmountPool* is only a type of "notepad" onto which the resource's counter can write its values.

One finds JDF implementations in offset print shops, and mostly within sheetfed print shops. In web offset, gravure, flexo, or screen printing there is a low penetration rate that is limited to MIS and prepress (see Chapter 12). A JDF integration of gravure, flexo, or screen printing machines is currently unknown to us. Indeed, the *ConventionalPrinting* process includes these printing processes, but, to date, the corresponding press manufacturers seem to have very little interest in the subject. Many of them are not even members of the CIP4 Organization. There could be several reasons for this:

- The original initiators of JDF are manufacturers of offset machines and the demands of the other print technologies were fewer and only considered later.
- In gravure and flexographic print, the prepress and printing companies are often separate firms, making it difficult to integrate.
- The transfer of the ink zone preset values has a particularly high cost/efficiency, which is not there for the other print technologies.
- There are essential printing standards only for offset, while other technologies mostly print to house standards or even standards specific to certain customers.
- The variability of substrates is higher than with offset, which complicates automation.
- With the print volumes in offset, which are mostly lower than gravure or flexo printing, longer makeready times come into play.
- Especially with gravure sector, there are only very few large printing firms which have individual order control systems.

Nevertheless, it is conceivable that this situation could change in the near future. Because, at the very least, the JMF messaging about material consumption, the current status of the printing presses, and print orders are naturally of interest for orderly production in each case.

10.2 Digital Print

Digital print is well known for its strengths, especially short runs, short production times, and personalized printing. Thus, a possible end-to-end automation is an important prerequisite for the cost-effectiveness of this print technology and JDF is, therefore, a potentially relevant data format.

One can divide digital print into two applications areas, although their borders are not clear but rather fluid:

- Office and copy shop printing
- Professional printing with a digital print system

In particular, digital printing systems differentiate themselves from copy-shop printing through more imposition options, higher color fidelity (color management), and expanded inline finishing options. These categories again mirror each other in the corresponding two ICS papers, titled the *Office Digital Printing ICS* and the *Integrated Digital Printing (IDF) ICS*. JDF-networked, integrated digital print systems are often part of a "hybrid workflow" in which the data for both offset and digital print is prepared in a WMS. Not only can offset plate production be automated, but the cylinder engraving in gravure printing can also be more strongly JDF-supported. Even in this area there are implementations where JDF data, exported by the MIS, are taken from the gravure system and from production reports and returned to the MIS.

Table 10.13

No.	Integrated Print System	Digital Printer in Office/Copy-Shop
1	LayoutPreparation	LayoutPreparation
2	Imposition?	Imposition
3	ColorSpaceConversion?	
4	Interpreting	Interpreting
5	ColorSpaceConversion?	
6	Rendering	Rendering
7	ColorSpaceConversion?	
8	Screening?	
9	Imposition?	
10	DigitalPrinting	Digitalprinting?
11	Folding?, Stitching?, Trimming?, HoleMaking?, CoverApplication?, SpineTaping?	Folding?
12		Stitching?

Compared to sheetfed offset, digital print differentiates itself especially in technical process points:

- In digital print, no physical printing form is produced; the JDF process *ImageSetting* is unnecessary.
- A digital printing press often has integrated inline finishing components.

The second point leads to digital printing being represented in JDF via a combined process, for which the intermediate results of the individual processes need not be described. The possible processes are presented in order from the two ICS papers in Table 10.13. That both processes Imposition and *ColorSpaceConversion* are listed several times and marked with a question mark as optional in the "Integrated Print System" is immediately conspicuous. This means that both processes can exist at different positions in the workflow sequence, but also that they must always exist at exactly one position. The finishing processes are all optional and may be carried out in more or less arbitrary sequences. The "office digital printer" is expected to have no dedicated color management control and fewer options in finishing processes.

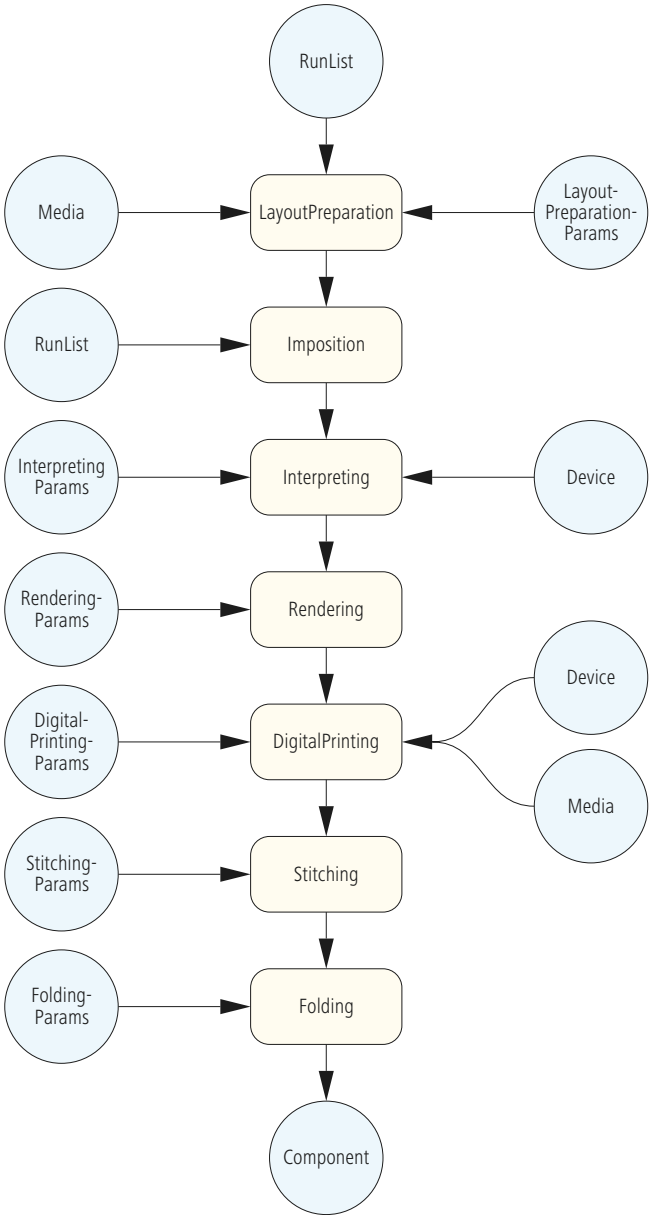


Figure 10.14
Practical example of digital print

Due to the variety of the processes involved, such a combined process can have many input resources. As the output of the combined processes, one always receives a *Component* resource. Also intermediate outputs of the processes are resources of the type *Component* or *RunList,* if they are even ever listed. In Figure 10.14

a very simple example is provided, which, indeed, does not give all of the options, but comes from practical experience. The corresponding code is listed in Figure 10.15.

The *LayoutPreparation* process is similar to the *Stripping* process which was handled in Section 9.2. Namely, both produce a Layout as an output resource, i.e., an imposition layout, which is necessary for imposition. Both also have input resources, through which they are supplied with the information they need for layout generation. For the *Stripping* process, the input resource is called *StrippingParams;* for the *LayoutGeneration* process it is referred to as *LayoutGenerationParams.* The difference between the two processes lies in the application, because *LayoutPreparation* is used in digital printing, while Stripping is used in offset prepress in concert with an MIS. There is the possibility here that there will be changes in the next JDF version, because the description of the *LayoutPreparationParams* resources is already stated in version 1.4, which could be abandoned again in future versions.

Figure 10.15
Practical example of digital print

```
<?xml version="1.0" encoding="UTF-8"?>
<JDF Status="Ready" Type="Combined" Types="LayoutPreparation Imposition
 Interpreting Rendering DigitalPrinting Stitching Folding" Version="1.3" …>
    …
   <ResourceLinkPool>
      <ComponentLink Amount="65" CombinedProcessIndex="5" Usage="Output"
       rRef="_2503">
         <Part SignatureName="Sig1" />
         <Part SignatureName="Sig2" />
         <Part SignatureName="Sig3" />
      </ComponentLink>
      <LayoutPreparationParamsLink CombinedProcessIndex="0" Usage="Input"
       rRef="_2504" />
      <MediaLink CombinedProcessIndex="0 4" Usage="Input" rRef="_0918" />
      <DigitalPrintingParamsLink CombinedProcessIndex="4" Usage="Input"
       rRef="_2507" />
      <DeviceLink CombinedProcessIndex="2 4" Usage="Input" rRef="_2509" />
      <RenderingParamsLink CombinedProcessIndex="3" Usage="Input"
       rRef="_2511" />
      <InterpretingParamsLink CombinedProcessIndex="2" Usage="Input"
       rRef="_2510" />
      <StitchingParamsLink CombinedProcessIndex="5" Usage="Input"
       rRef="_2513" />
      <StitchingParamsLink CombinedProcessIndex="6" Usage="Input"
       rRef="_2514" />
      <RunListLink CombinedProcessIndex="0 1" ProcessUsage="Document"
       Usage="Input" rRef="_9437" />
   </ResourceLinkPool>
</JDF>
```

Personalization

Personalized printing, also called variable-data printing (VDP), means that individual elements (text, graphics, images) can be different for each side/piece for each press sheet. Usually there is a static piece with VDP which is identical on each sheet and a variable part which is controlled by a database or a control file. One such order usually consists of the following sub-tasks (Figure 10.16):

- Preparation of a data system (databases or tables) with information about the addressee
- Design of a template for each side/piece where the static and variable elements are positioned, commonly accomplished with plug-ins from layout programs
- Definition of the rules for linking the variable part of the template with the data system
- Merging all of the information for output to the RIP of a digital print system

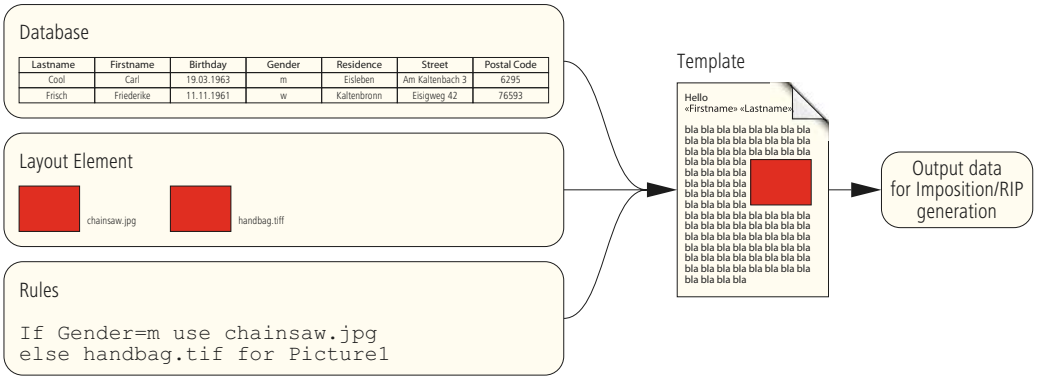


Figure 10.16
Subtasks for personalized print

For the output of the personalized data to a RIP one could use a normal PDF. Then each side of each sheet would have to be individually RIPed, even the static elements. The variable portion is also typically only a selection from a collection of a few static elements. As a result, this approach would be very ineffective. Instead, a better special language and corresponding RIPs are used in order to make it possible to transmit and RIP graphic elements only once but to use them repeatedly and variably.

First and foremost is the XML-based **Personalized Print Markup Language** (PPML) [11] [36] [37] [38] with the subspecifications

PPML/GA (Graphic Art); this is in addition to the many vendor-specific formats which will not be discussed here. The data format **PPML/VDX (Variable Data Exchange)** which is based on PPML and PDF and the relatively newly specified format from Adobe, **PDF/VT (Variable Transactional),** are other options. Some specialists in the industry foresee that the PDF/VT-Format [25] will replace PPML in the future, especially since it is integrated into **Print Engine2** (the OEM RIP technology from Adobe Systems [1]).

PPML was presented in exactly the same way as JDF was in 2000. This vendor-independent language was developed by the PODi (Digital Printing Initiative) organization, founded in 1996. Is PPML now a competing format to JDF in digital printing, or does it supplement it? To answer this we will first examine the most important properties of PPML.

PPML allows digital objects like images, text, and pages which exist in different data formats (EPS, PDF, JPEG, TIFF, etc.) to integrate into one PPML file, whereby these objects are not necessarily embedded into the PPML file but instead can be referenced. In PPML, these objects can then be positioned, transformed, and trimmed. In PPML one can also define objects to be reusable so they can be RIPed only once and subsequently stored in a cache for later uses. Graphic objects are also not stored in PPML, but in the usual formats. Instead, PPML describes only how these objects must be combined in order to make individual pages.

It is sometimes argued that PPML simply describes only the VDP pages, like PDF, but more robustly structured, and the JDF workflow begins with the completed pages, hence PPML and JDF are only supplementary formats. But this is not completely right for several reasons:

- JDF also describes page creation, represented by the *LayoutElementProduction* process.
- PPML also includes optional workflow metadata, such as the description of the imposition layout [37].

Consequently, although both formats overlap each other in some things, they, in turn, complement each other in many other things. For example the properties of the substrate in PPML may not be precisely specified; in JDF they can be very well defined (in the *Media* resource). Indeed, a PPML file could even contain JDF data or a reference to a JDF file, but this was withdrawn with version 2.2 since PPML in the future shall only serve as content description. Conversely, a *RunList* resource can reference PPML data. This

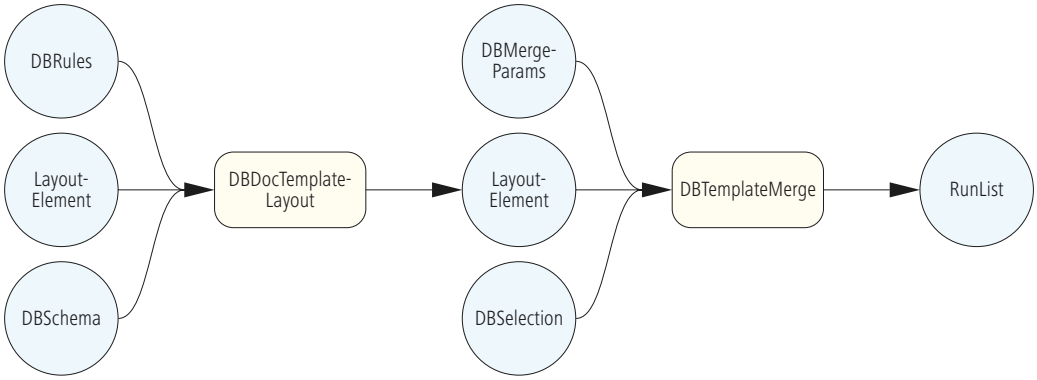


Figure 10.17
JDF model for personalized print

resource is then input for a PPML consumer. In [39] the conventions between the two formats are covered in detail. For example, the attribute *IgnorePDLImposition* is given in the JDF resource *LayoutElement* if the sheet layout information should be taken out of the PPML data or out of the JDF data.

Figure 10.17 again gives the JDF model for VDP. The *RunList* resource on the right side represents a PPML or another VDP language. On its part, it is typically input from an *Imposition* process. On the left side we have the process *DBDocTemplateLayout,* which creates a template for the static and variable elements whereby the link to the database is already defined. For this task, rules are needed as to which text or graphic element should be incorporated into the template. Since the implementation of the rules is very disparately handled from applications, they are only inserted into the *DBRules* as readable comments. In the *DBSchema* resource, the database type is given (SQL, XML, or comma-separated text entries) as well as a readable comment about scheme. In *LayoutElement* the individual content elements which are required for the template and as variable elements are described and their URL is given. The *DBDocTemplateLayout* process generates a new *LayoutElement* resource, which represents the template. In order to really identify as a template, the value *True* must be entered in the *Template* attribute.

Finally the *DBTemplateMerge* process has the task of generating a VDP language like PPML out of the template, such as information like where in the file system the file(s) are stored, and what is to be found in the *DBMergeParams.* The URL of the database, the indices of the database records, and the database access in the database-specific language (like SQL) is held in the *DBSelection* resource.

11 Postpress

There were few networked postpress finishing machines before JDF was published. For example, high-speed cutters received their data over PPF, as described in Chapter 4.2. Or proprietary solutions were developed with which the most important data for the folding (sheet size and fold location) of a job was coded in an EAN code, printed in a job jacket, and later read on the folding machine with the help of a barcode reader. There was, and still is, another solution: for example, when a job starts a cutting program, a folding plan is defined for the cutter or folder, respectively. However, in order not to tie up these (expensive) special machines with setup work, work preparation computers are set up to be able to transfer the programs over the network to the finishing equipment. This leads to the reduction of setup times for the special machines. In addition, these computers collect order- and machine-related data from the production equipment (especially the folder and saddle stitcher) such as setup times, machine status, production rate, and so on. The machine manufacturers have developed proprietary solutions for this and the protocols between the job scheduling software and the equipment are manufacturer-specific (as between the control centers of the printing machines and the printing machines themselves). But such networked configurations are not widespread, and connecting via JDF is still very scant in finishing.

There are two areas offered by the industry for potential JDF/JMF interfaces:

- The console terminal of the finishing machine, and
- The work preparation stations for one or more finishing machines

To revert to CIP4 terminology, in one the JDF device communicates directly over the local network to the production machine or machines, and in the other the JDF device communicates to a dedicated machine or machines directly over the local network. In both cases the imported JDF values are then able to be used for the default setting, which the user is still able to modify in the software. In the second case the finished production programs can then be passed over the network to the machines in the proprietary format. Conversely, the production data must be translated into JMF messages and JDF structures from the proprietary protocol. Figure 11.1 shows both configurations as a flowchart. The difference between both solutions is contained in the diagrams, where a separate job preparation station can sort orders in sequence and dis-

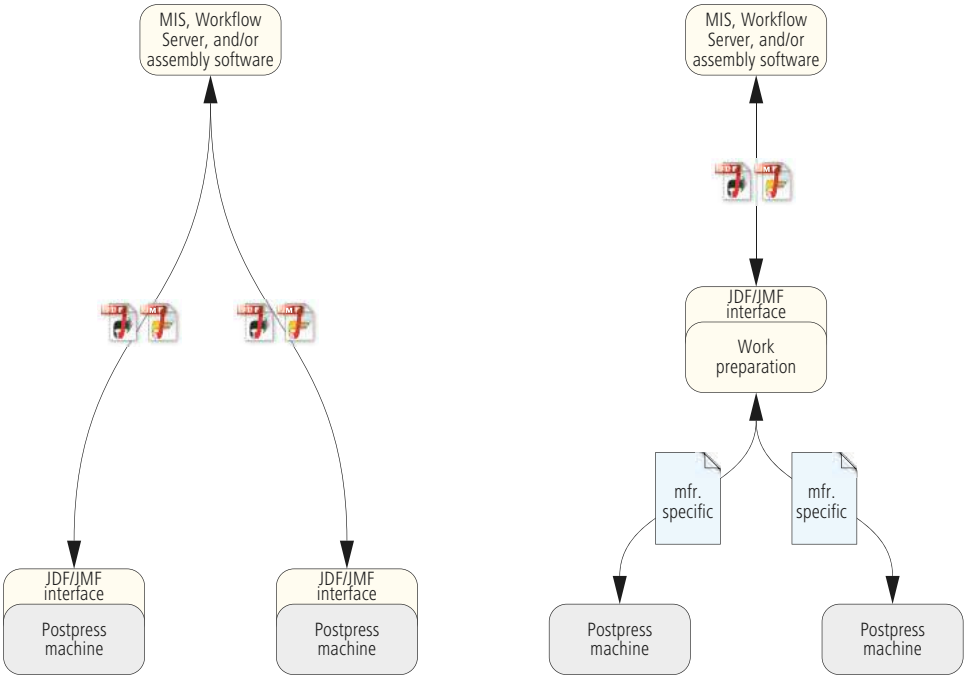


Figure 11.1
Two different configuration principles of JDF/JMF interfaces in postpress

tribute them to different machines if necessary. The operator of the machine(s) is then relieved of that burden. With a JDF/JMF interface integrated into a machine, either the MIS or the user specifies the production sequence to the machine and makes sure that the changeover times are as minimized as much as possible. In this case, several JDF/JMF interfaces must also be maintained if they reside with the production machines, and software updates can be difficult. On the other hand, the solution really only makes sense with a work preparation solution if several production machines from one manufacturer can be controlled with it.

Examples of JDF-compatible Postpress Connectivity (2009)

Manufacturer:	***Main Product:***
Ferag AG	iQ
Heidelberger Druckmaschinen	Prinect Postpress Manager
Hohner Maschinenbau GmbH	ixFrame (der ixact GmbH)
Horizon International Inc.	i2i
MBO	Datamanager
Müller Martini	Connex

JDF devices in finishing receive JDF/JMF information either from MIS, from a workflow server in production, or directly from an assembly program. The latter are those which actually specify the positions of the cut and fold marks in as much as the information is able to be relayed. Indeed a configuration is also possible in which the assembly program returns the information to the MIS or a workflow server that will in turn serve finishing. Another possibility is that an assembly program completely controlled by the MIS creates the imposition layout and has no facilities for altering position on a page. The MIS can then directly determine the cut and fold positions for finishing.

11.1 Guillotine Cutter

Raw sheets, which are also intermediate products (as, for example, are press sheets), are usually cut with a high-speed cutter, also called a guillotine cutter. The cutting of raw sheets before printing not only serves the purpose of obtaining the correct sheet size but also to obtain squared sheets. Conversely, press sheets are cut in order to get multiple signatures from a press sheet or also in order to reduce the size of a signature since the edges are not allowed to be too large at the three-knife cutter after folding.

In the JDF notation, raw sheets are a resource of the type *Media;* intermediate or end products are of the type *Component.* The cutting process (*Cutting*) therefore receives either *Media* or *Component* resources as input and also generates the same resource type as output. The model can be seen in Figure 11.2. The question marks, which are at the input resources and indicate optionality, are interpreted here to mean that exactly one of the two options is mandatory. The output resources are marked with an asterisk, which stands for zero or multiple numbers of resources. We must also point out that exactly one resource may be listed here which is identical to the input resource, and this must occur at least one time.

The most important information for the cutting process is in the *CuttingParams.* Similar to PPF, cut blocks (*CutBlock*) may be defined with JDF (see Section 4.2 and Figure 4.12 in particular). Where cuts must be is the only thing specified, but the cutting sequence is not defined. Alternatively, they can be defined in the *CuttingParams* resources. Each cut specification is an element of the type *Cut.* In older JDF versions one could also have defined the positions of the cut marks in the *CuttingParams* resources, but since Version 1.3 this information has been stored in the *Layout*

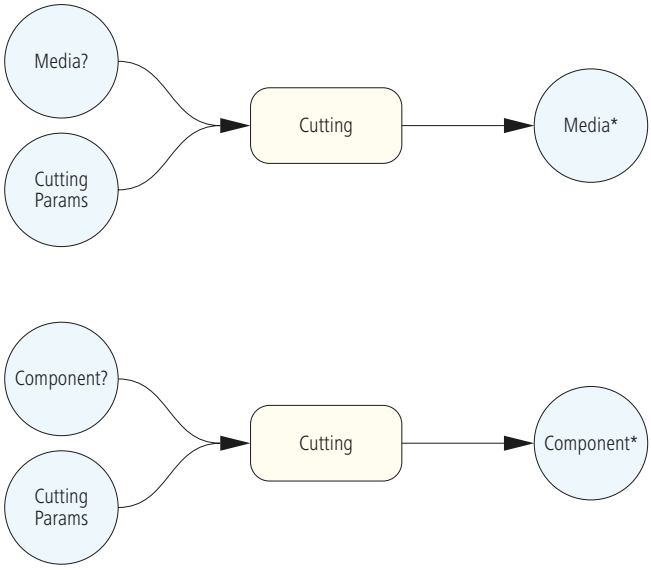


Figure 11.2
JDF model of the cutting process for raw sheets (top) or intermendiate or end products (bottom)

resource, similar to what we saw in Figure 10.7 for the print control element.

Figure 11.3 shows the JDF code of a *CuttingParams* resource, which contains six cut blocks, specifically one for each cover block. For clarity we have removed a few attributes and limited all of the values to two places after the decimal point—in reality you would enter the values with more precision. The block size of 768.18×396.84 DTP points corresponds to 271×140 mm. The block size is cut initially and is comprised of the recto, verso, and spine of the booklet and a one-centimeter bleed around the final format. The position of each block is determined with the *BlockTrf* attribute. The definition of the matrix, which holds the values of the attributes, is implemented as outlined in Figure 9.21. One con-

Figure 11.3
Definition of cutting blocks

```
<CuttingParams Class="Parameter" ID="_77" SheetName="Cover"
 Status="Available">
   <CutBlock BlockName="Cover_1" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 14.17"/>
   <CutBlock BlockName="Cover_2" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 14.17" />
   <CutBlock BlockName="Cover_3" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 411.02" />
   <CutBlock BlockName="Cover_4" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 411.02" />
   <CutBlock BlockName="Cover_5" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 96.37 807.87" />
   <CutBlock BlockName="Cover_6" BlockSize="768.18 396.84"
    BlockTrf="1.0 0.0 0.0 1.0 864.56 807.87" />
</CuttingParams>
```

verts the DTP points into centimeters in the configuration from Figure 11.4. A three-knife trimmer cuts the booklet into the final format at the end. However, this operation is not described by a *Cutting* process, instead through the *Trimming* process. Accordingly, there are no *CuttingParams;* instead there are *TrimmingParams.* These would be entered as 120 × 120 mm final format in our example.

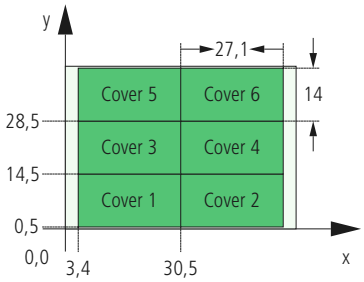


Figure 11.4
Position of the cutting block on the press sheet

11.2 Folder

With modern folding machines, many things that were able to be set only manually may now be entered into a dialog box and then controlled automatically. The side fence for the sheet feeder, the roller settings based on paper thickness, or the stop in the plate folder would be examples here. In contrast, other parameters are not automatically controlled, such as the air supply to the sheet separator. But it would be possible here that defaults for certain types of paper or even specialty papers could be stored and recalled with the help of the order information or with the JDF, or in the future a "learning" software could determine, collect, and return appropriate settings for specific conditions.

Typically in these folding machines with default settings, sheet size and folding type are entered in the menu navigation whereupon the electronic setup system takes over the motorized adjustments. After a test folding the fine adjustments occur, which means it takes several iterations to proof and correct the deviations from the fold marks. But since at the beginning only the sheet size and fold type are available as parameters for processing, printing must not cause paper distortion and errors in the previous work operations; instead differences in individual piece lengths based on corresponding positioning or by the imposition applied before or after the fold (pre- or post-applied folio) are to be corrected. In other words, the information from the folding catalog number and folding sheet are not sufficient (see Figure 11.5), and fine adjustment can therefore be considerably more complex than the default setting.

Likewise, the same readjustment is also necessary if only the fold pattern and the sheet size are given in the JDF folding resource. However, some systems give the precise folding position and folding sequence as JDF, wherein then the individual existing pre- or post-folio is already calculated. With this, the post adjustment effort and also the error rate are lower.

Figure 11.6 shows a folding type for a 24-pager with a folding catalog number F24-8 entered in the *FoldingParams* resource. In

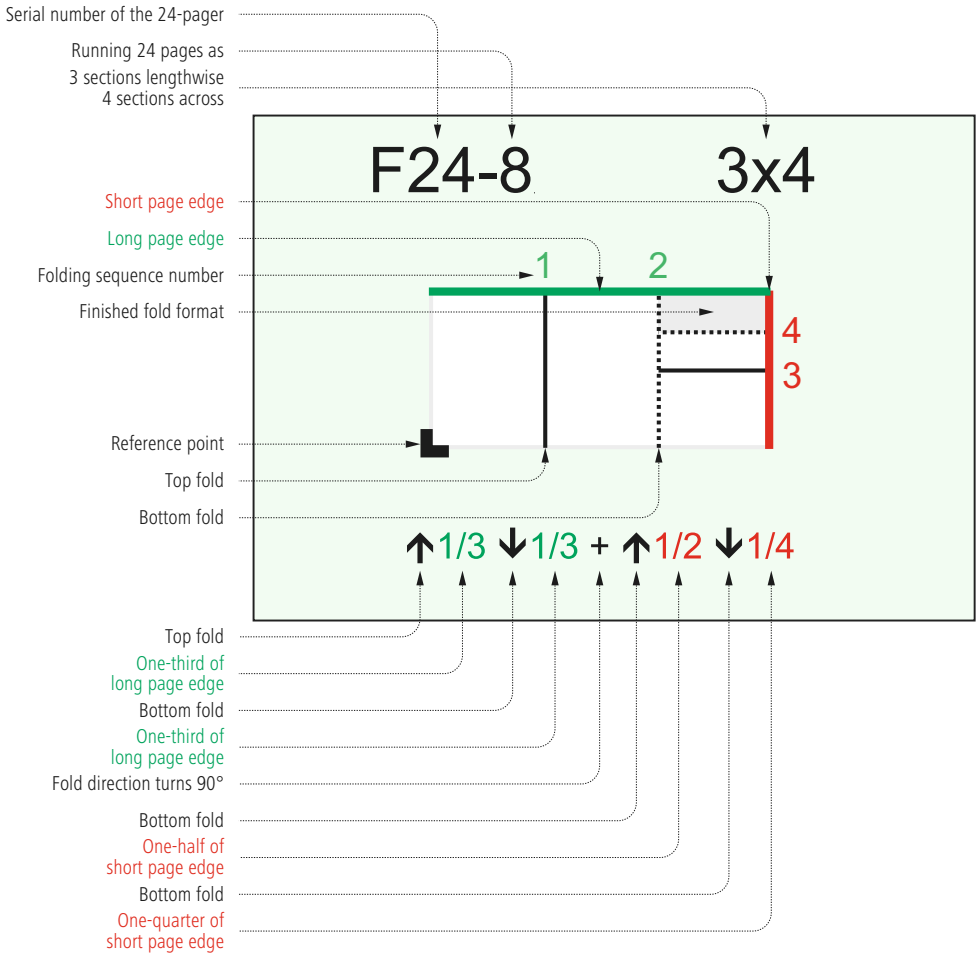


Figure 11.5
F24-8 from the folding catalog of specification 1.4

many cases this information is all of the information that is passed along. But additionally in the previous example the fold position was specified in the *Fold* element. Each *Fold* resource defines a folding operation wherein the sequence of the fold elements also defines the folding sequence. If, as in our example, both the folding catalog number and the individual folding positions are noted from an agent/controller, the *Fold* elements are produced from

Figure 11.6
Folding parameters

```
<FoldingParams ID="FOLD_A-1" Class="Quantity" Status="Available"
 FoldCatalog="F24-8" SheetLay="Left">
    <Fold From="Front" To="Up" Travel="943.9370079811025"/>
    <Fold From="Front" To="Down" Travel="1887.8740161811025"/>
    <Fold From="Left" To="Up" Travel="907.0866141732283"/>
    <Fold From="Left" To="Down" Travel="1360.6299241732283"/>
</FoldingParams/>
```

the JDF device. At a minimum, the JDF specification requires it. The *From* attribute provides the corner from where it is folded; the *To* attribute provides the fold direction. The edge descriptions are visible in Figure 11.7; also the reference point is front-left. The travel values are given, as usual, in DTP points. With the first folding operation in Figure 11.5, the sheet is folded at x = 33.3 cm (equivalent to 2.54*943.9370079811025:72) in the y-direction whereby the front part is laid over the back part. The third folding operation is at y = 32 cm, where the left part of the sheet comes to lie on the right part of the sheet.

Figure 11.7
Designation of the edges on a signature

Folding machines are often put together from different modules dependent on the order. If a folding type is provided as an input into the JDF device as a *FoldingParams* resource, which the current configuration cannot fold, the device gives an error message directly to the folding machine. The customer service representatives or project managers must therefore be aware of the capabilities of the folding machine and the schedulers also must know the current configuration in order to determine the order sequencing. If you want to deploy a software-supported solution at this point, monitoring of the machine configuration with respect to a query about the capabilities of the JDF device is necessary. This is not as trivial as it may seem at first glance, because a product has a combination of properties which are important for the decision of whether it can be folded on a specific machine or not. So, for example, a folding machine can work with paper from 250 g/m^2, and also process a 6-fold z-fold, but not the two together. This subject is dedicated to the JDF specification under the term *Device Capabilities.* With it, a device can define which nodes, elements, attributes and their values, resources, and JMF messages are supported. So the maximum folding sheet size can, for example, be transmitted to the MIS or a production controller. The possibilities are still very far-reaching, and it can even describe throughput parameters of a machine like sheet count per hour or setup times. JDF-specific details, namely whether the device software *GrayBoxes*—combined processes or general process group nodes can be combined—are also able to be communicated. Figure 7.6 shows the situation in principle.

11.3 Saddle Stitcher

A saddle stitcher generally consists of three modules:

- A collecting machine
- A saddle stitcher
- A three-knife trimmer

With collecting, different signatures are inserted into each other so that an inserted block is created. For this purpose several feeders, mostly built in series, manually or automatically feed different signatures with rods or rolls, with attention to the correct sequence (which means from inside to outside in the final product). In production, the signatures are then separated, opened in the middle, and laid on a saddle. As the sheets are transported from one feeder to another, the individual signatures are laid over each other. In the last process, where necessary, the cover is brought to the block. The number of different signatures which are collected depends, naturally, on how big the print product is, but also on the number of available feeders. Therefore, the setup of a saddle stitcher includes the adjusting of the different feeders based on the signature format and the substrate.

In a saddle stitcher, also known as a wire stitcher, or just "stitcher" for short, metal staples are stuck through the spine of the block and bent over. The number, position, widths, and shapes of the wire staples may vary from order to order. With a block, each staple is set from a single head set, which cuts the correct length wire from a coil; the staple bends, pokes through the gutter, and finally bends and closes.

The trimmer (three-sided cutter or also three-knife trimmer) is responsible for cutting the stitched print product into the end format and, if necessary, at the same time for cutting the fold in order to unbind the pages. For this purpose, all of the pages outside of the bound edge are cut (head, foot, leading edge, face, recto). Some trimmers allow also for a middle separation cut so that two-up production is possible.

As expected, three processes for stitchers are definitive in JDF: *Collecting, Stitching,* and *Trimming.* Since all three modules of a saddle stitcher are usually strongly coupled, it is therefore advantageous to describe these three processes through a combined process or a process group. In Figure 11.8 a corresponding *GrayBox* is de-

picted with its input and output resources. The resources are described briefly below:

- The *Device* resource contains the device identifier and the cost center of the saddle stitcher.
- In *NodeInfo* the estimated setup and production times are entered.
- *Component* contains the description of the signature.
- *Assembly* defines the sequence with collection of the signatures.
- *StitchingParams* notes the number and shape of the staples.
- *TrimmingParams* contains as information as to the final format size of the print product as it will be eventually trimmed.

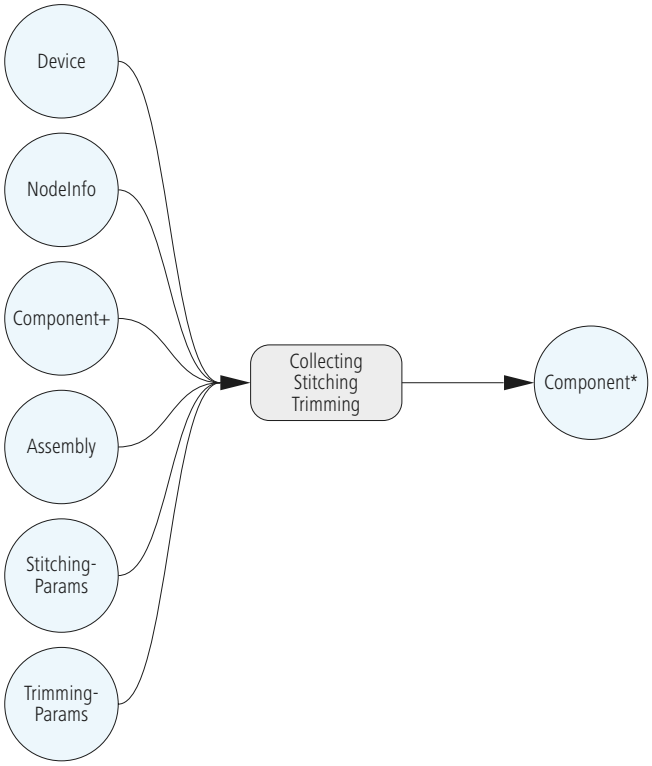


Figure 11.8
GrayBox for saddle stitching (collecting, stapling, and three-side trimming)

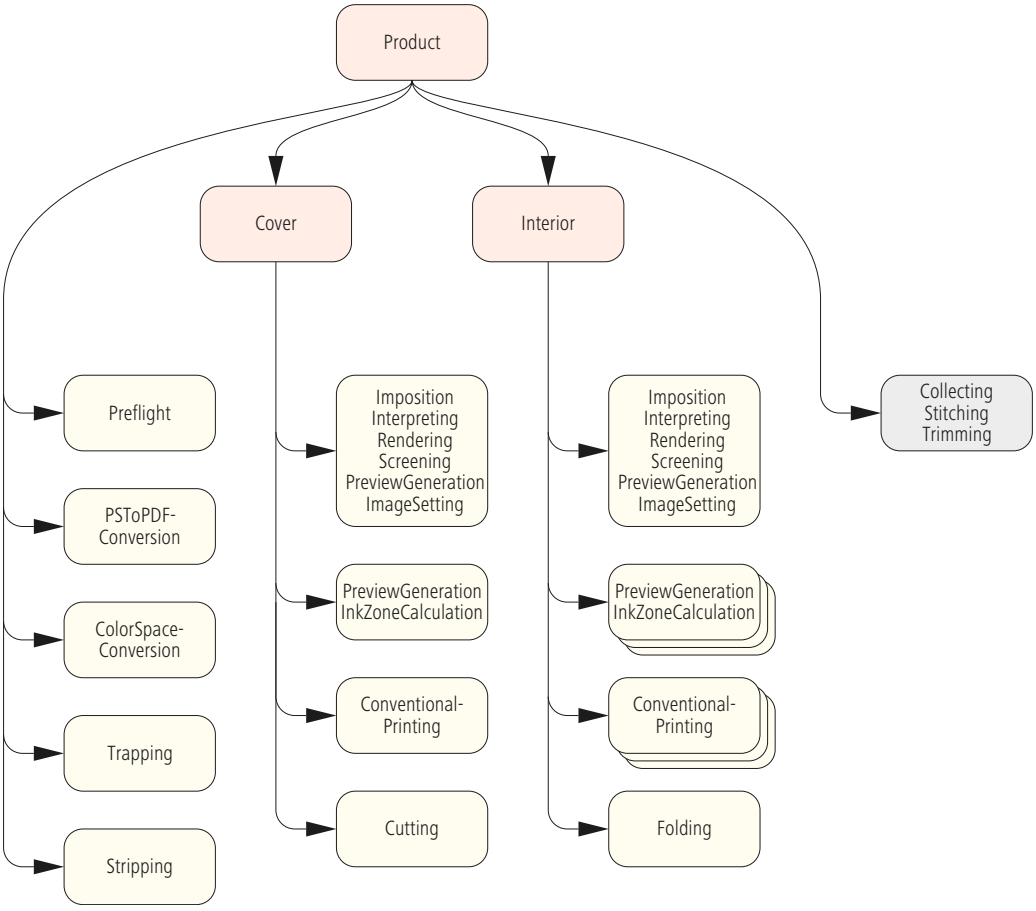
Actually one would also expect a *CollectingParams* resource for the *Collecting* process, and in fact it is listed in the JDF specification—but without attributes and subelements. It is only a container for extensions and not expanded upon here. The sequence of signatures is also not defined in this resource, instead in the *Assembly* resource. The *AssemblySection* elements are entered there with their order and also their sequence given, where the first element describes the outer position and the last element describes the innermost position. Figure 11.9 describes this situation with three signatures for the contents and once for the cover. With selective binding, where signatures are collected in a variable way, the *Collecting* process receives further input resources (*DBrules* and *DBSelection*). Thus, it is possible to produce for different versions of the print product for different addressees.

```
<Assembly Class="Parameter" ID="_019069" Order="Collecting" "
 Status="Available">
    <AssemblySection AssemblyIDs="Cover_B_1"
     DescriptiveName="F04-01_ui_2x1_1" />
    <AssemblySection AssemblyIDs="Text_1_B_2"
     DescriptiveName="F08-07_li_2x2_2" />
    <AssemblySection AssemblyIDs="Text_2_B_3"
     DescriptiveName="F08-07_li_2x2_3" />
    <AssemblySection AssemblyIDs="Text_3_B_4"
     DescriptiveName="F08-07_li_2x2_4" />
</Assembly>
```

Figure 11.9
Collecting sequence of the signatures

In conclusion, the example from which the *Assembly* resource in Figure 11.9 was taken should be described in whole (although somewhat simplified). The JDF tree is shown in Figure 11.10. In the left column the central prepress processes are marked: proofing the data on production suitability (*Preflight*), distillation of the PostScript data to PDF (*PSTToPDFConversion*), the color space

Figure 11.10
JDF process tree



transformation (*ColorSpaceConversion*), over or under filling (*Trapping*), and the creation of the imposition sheet (*Stripping*). Often these processes are initially outlined only through the *GrayBoxes PrepressPreparation* and *ImpositionPreparation*. The middle two columns in Figure 11.10 show the different processes, or rather combined processes for the partial products "Cover" and "Contents." Both of the combined processes above contain the process name *PreviewGeneration,* but their functions are different. While a preview image is defined in the combined process *ImpositionImageSetting,* which will be visible later on the monitor of the printing press control panel, the preview image of the combined process *PreviewGeneration InkZoneCalcuation* is created only for the calculation of the ink zone presets (typically separated images in 50.8 ppi). Since multiple cover "ups" are found on the press sheet, a cutting process follows the printing process. The content sheets, however, are folded.

Figure 11.11 outlines yet again a part of the production process where only the most important "transfer resources" are shown: The pages (*RunList*) for imposition (*Imposition*), the plates (*ExposedMedia*) as interface between prepress and print, and finally the different component resources press sheets, signatures, and the finished product.

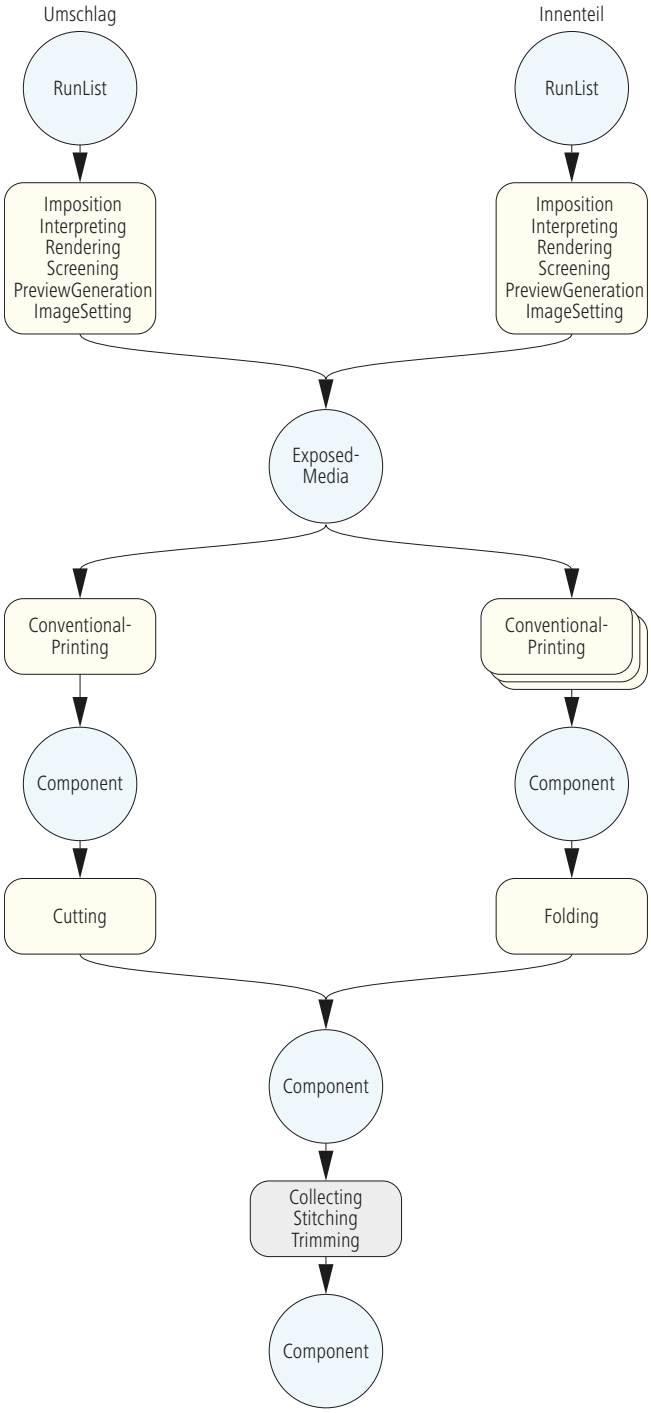


Figure 11.11
Production process

In the glossary of process names in the appendix, 50 different processes of print production are briefly shown, but only five were handled in more detail here (Cutting, Folding, Collecting, Stitching, Trimming). The process of die making (*DieMaking*) is presented in the next chapter. All of the others, unfortunately, must remain unconsidered.

Exercise:

Make a process chain for a perfect-bound brochure similar to Figure 11.11 where the cover is embossed with a gold foil. Look for the necessary processes within the JDF specification and also list the "Transfer Resources."

Likewise, create, similar to Figure 11.10, a potential JDF tree for a perfect-bound brochure.

12 Packaging Printing

From the perspective of the JDF/JMF model there is no fundamental difference between packaging printing, commercial printing, and other areas of a print application. Indeed, other substrates (such as paper board, corrugated board, film, or composite) are used in packaging printing, as are other print and finishing machines, yet there is no change here to the JDF process point of view. Only a few new attributes and attribute values must be defined for some resources. The repeatedly used attribute *MediaType* therefore uses values such as *CorrugatedBoard* or *Foil.* Outside of this, attributes for describing these materials, such as *Flute* for corrugated board, are introduced (for example, in the resources *Media* and in *MediaIntent*). Since JDF 1.4 there is also growing support for flexo printing. For example, the values *Sleeve* and *MountingTape* are defined for the attribute *MediaType* and for flexo plates the values *PlateTechnology* and *ReliefThickness* are defined.

However, there are processes that are mainly used in the area of packaging. In this chapter only a few of them will be presented and will be divided into the three following areas:

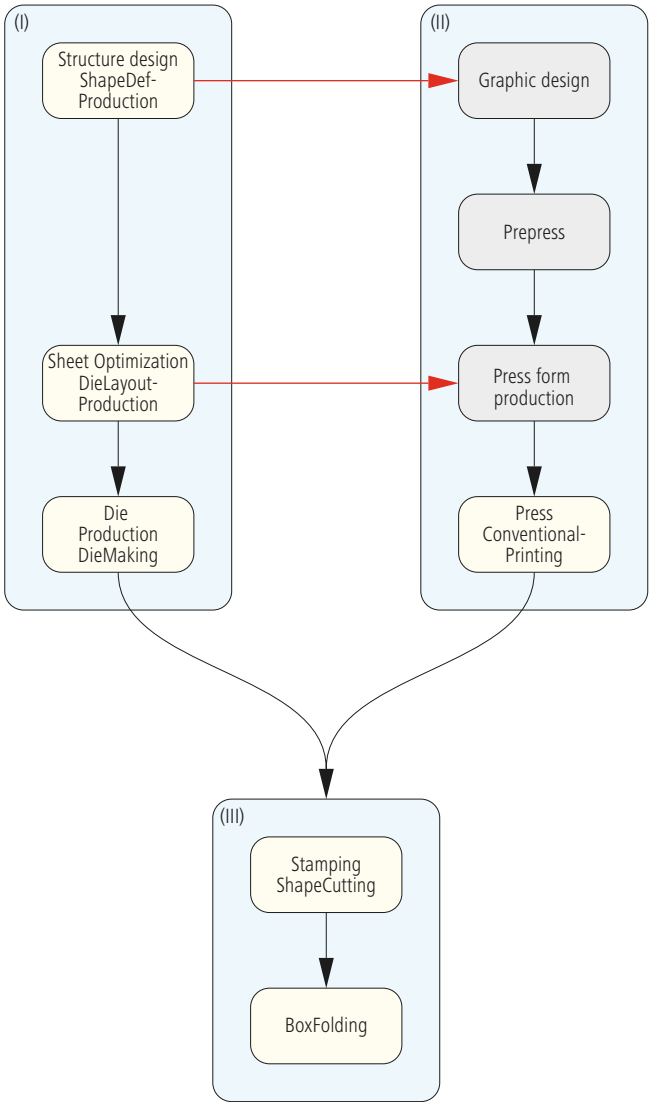
a) Folding box design (structure design), sheet optimization, die making

b) Stamping and folded box gluing

c) Barcode management, including stroke-width compensation, especially in flexographic printing

Figure 12.1 shows an activity diagram where in the example A and B the listed processes are

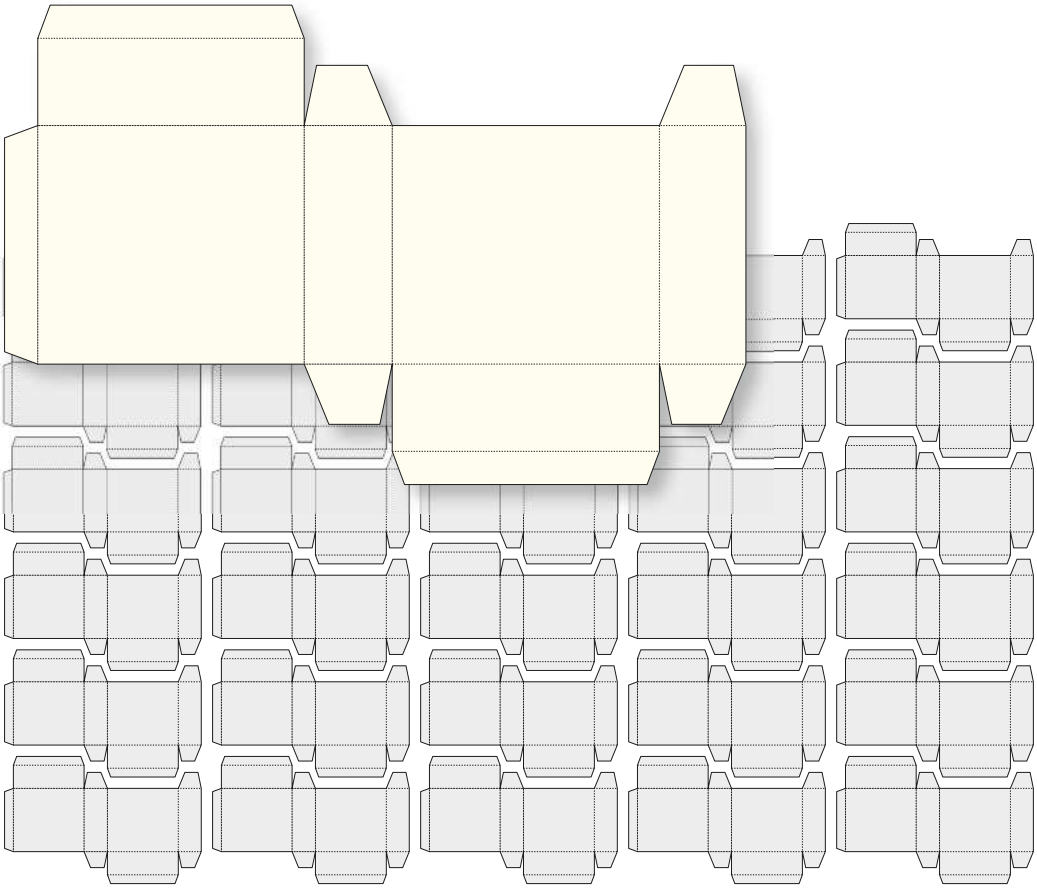
Figure 12.1
"Folding carton" activity diagram



seen in the context of folded box production. Line width compensation in C is a partial process of the "PrePressPreparation" activity.

The activities highlighted with yellow rectangles correspond to a JDF process. With construction design (*ShapeDefProduction*), the outline, scoring, and perforation lines for one of the units is usually defined with a CAD system. With layout optimization (*DieLayoutProduction*), one or more units are subsequently placed on the sheet multiple times so the least possible amount of material is wasted (see Figure 12.2) and at the same time so that production can run quickly and easily. As a rule, this procedure is carried out with the same CAD system. Additionally, even more things which are important for cutting die construction are laid down, such as the register holes for the intake of the tool in the punching machine. With cutting die construction, the exported CAD data can then be imported back again into the corresponding system for production of the punching tool, whereby a punching tool can consist of multiple parts (see Figure 12.3):

Figure 12.2
Single unit and unit optimization



- Die (*CutDie*)
- Counter Finishing/Counter cutting die (*CounterDie*)
- Stripping tool (*LowerStripper, UpperStripper*)
- Blanking tool (*Blanker*)

Counter Finishing is required for forming the creases and the stripping board for separating the punching waste. Die-making production (*DieMaking*) creates one or more tools (Tool) for the diecutting operations (*ShapeCutting*). In a stamping machine, often the cut pieces are broken and the units are separated. Both can also occur offline, thus taking place beyond the punch. The carton blanks then go into the feeds of a carton gluing machine, in which the cartons are folded, glued, and closed. These operations are combined under the carton gluing process (*BoxFolding*).

The red line in Figure 12.1 denotes special data flows. Namely, for a graphic design of carton information about diecutting, or rather crease curves, is required so that the design fits the carton. Often a corresponding EPS file or the PDF paths are simply exported from the CAD system and then read back in the graphics software so that the designer can orient him or herself on it. In addition, with imposition (*Step&Repeat*) the units are so positioned in the press form creation that punching and creasing the press sheet exactly

Figure 12.3
Elements of a die: rubberized die in closed frame (right), die plate with counter punch (top left), press sheet (bottom left), punched and broken-out single unit (center), and finished carton

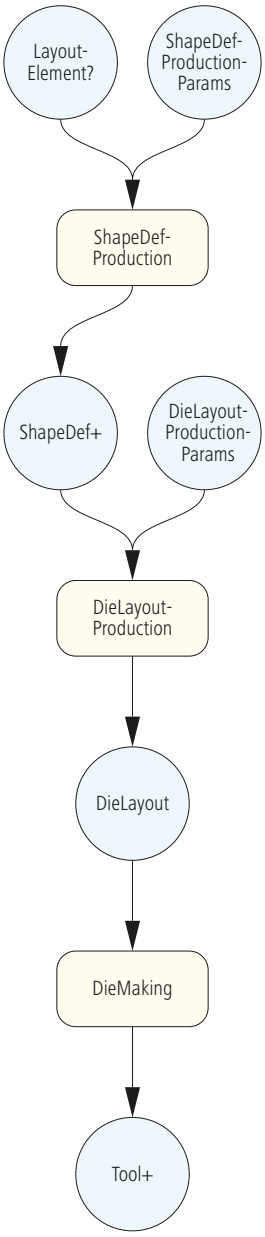


Figure 12.4
Die making

match the unit arrangements. Since the positions of the units are mostly set up early with the CAD system and not in the press form creation, position data from the CAD system is commonly passed in the form of CFF2-, DXF, or in DDES files. These are provided to both the die mold manufacturer as well as the press form manufacturer. Therein lies the reason why the press form manufacturer doesn't determine the position of the units; the production of a die takes a longer time, especially since it is frequently outsourced. And as a rule, the CAD drawings come at the beginning of the manufacturing process during the press form manufacturing; shortly before the press date the data for the set of plates is prepared.

With simple labels (*Labels*), which are bounded by a rectangle (*BoundingBox*), MIS sometimes provides the information for *Step&Repeat* via instructions and communicates these both to the die making as well as the press form preparation. For folded cartons, which are assembled and interleaved in a complex way on a press sheet, a CAD system or other production software must take over the sheet optimization.

12.1 Die Design, Sheet Optimization, and Die Manufacturing

In Figure 12.1 Box I shows the structural design with CAD and the die form construction while Box II shows the work steps of the graphics production up to printing, and Box III represents the print finishing work. The processes in I were first introduced with the JDF 1.4 specification 1.4 and are therefore not yet currently implemented in a finished JDF workflow. The examples shown here are therefore not based on production data. The processes in Boxes II and III were, however, already set in the specification several years ago, and there are implementations in the market (albeit only a few) which read and execute these JDF processes. In Figure 12.4 Box I from Figure 12.1 is no longer shown as a JDF process-resource model. For the *ShapeDefProduction* process, i.e., the CAD construction of a carton (or label, cardboard displays, bags, etc.), there are two input resources available in which the optional *Layout-Element* is typically entered as a sketch once the construction is performed. The entry could be a URL of an EPS file, for example, in which the sketch is saved. The design itself is mostly provided by the client or an (internal or external) advertising agency. In the *ShapeDefProductionParams* resource, the URL and the extent of a 3-D model of the packaging can be listed. Furthermore, it is possible to enter the name of the packaging standards, for example from FEFCO or from the ECMA catalog. The output *ShapeDef* of

the *ShapeDefProductionProcess* describes the construction of the carton where either an external file which holds the contours can be referenced or the contours themselves can be entered into the resource. In the latter case, straight lines and Bezier curves are given as values of the CutPath attribute. The coding of the lines and curves are carried out as with PDF (see Section 4.4.1 in [5]). Figure 12.5 once again clarifies both possibilities. In the first case a CFF2 file is given as a reference; in the other a path is specified. The process resource *Shape* is namely of the path type in which the PDF path is defined in the *CutPath* attribute. With the operator *m,* the cursor is moved to a specific position. With the operator *l,* a line is drawn from there. And finally with the operator *c,* a curve is appended at the end of the line (*m* for *moveto, l* for *lineto, c* for *curveto*). The operands are always listed before the operator. The attribute *DDESCutType* gives the type of the cut or perforation according to the ANSI Standard DDES3 [7].

A package can naturally consist of multiple parts as, for example, in the case of lidded boxes. To that extent multiple ShapeDef resources can be input to the *LayoutProduction* process. In addition, different packaging may naturally be imposed onto a sheet. In the resource *DieLayoutProductionParams* some *Step&Repeat* parameters can be defined, such as the order and the gaps between the units. The *DieLayoutProduction* process then determines the sheet optimization of one or more contours and generates the *DieLayout* resource in which the CAD data is described, which is necessary for the die form construction and for imposition. Basically only the location of the external file is registered here. Finally, a physical die form (Tool) must be produced, which is described in the *DieMaking* process. As mentioned, there will be dies, counter finishing and stripping, and blanking tools made, but also embossing tools for blind or hot foil stamping, which we prefer not to elaborate upon here.

With the workflow of graphic production, a layout for imposition and plate burning must also be created to some extent as a counterpart to the DieLayout of the die form creation. For this purpose

Figure 12.5
Two different possibilities for describing folding carton contours

```
<ShapeDef Class="Parameter" ID="_4711" Status="Available">
    <FileSpec URL="file://Fileserver1/CFF2/Folding.cf2"/>
</ShapeDef>
```

```
<ShapeDef>
    <Shape ShapeType="Path" DDESCutType="101"
     CutPath=" 28 28 m 10 72 l 20 140 144 150 144..."/>
</ShapeDef>
```

the necessary stripping process was presented in Chapters 8 and 9 (see Figures 8.7, 8.9, and 9.8). In the area of commercial print, the position of a single folded sheet with respect to the position of several folded sheets is fixed and is also still noted in the subelement. In contrast, with the production of packaging, one uses instead an imposition scheme; either the optimized die form contours or a step-and-repeat pattern for arranging the content data. For this purpose, this information is able to be provided by direction of certain MIS systems which in turn receive the information from the CAD department (though not yet via JDF).

The JDF code in Figure 12.6 shows an example in which a reference to the external CAD data is entered in the *BinderySignature* resource. In the *StrippingParams* the position of the die lines on the press sheet is given first, similar to the example in Figure 8.8, only that there were more folded sheets placed on a sheet. Also, as before, the example contains the subelement *StripCellParams.* It contains the end format box as well as an indication about the clip path (*Mask*), which is identical to the cutting contour of the die. The *BinderySignature* resources can either be a direct child of the *StrippingParams* resources (as in Figure 8.9) or alternatively defined outside the StrippingParams. In the latter case the StrippingParams must naturally contain a reference to the BinderyParams, as seen in Figure 12.6. The *BinderySignature* has three subtypes (BinderySignatureType), namely:

- Folding Pattern (*Fold*)
- Step and Repeat (*Grid*)
- Dieline (*Die*)

Figure 12.6
Unit optimization by CAD data

```
<StrippingParams Class="Parameter" ID="_001" Status="Available"
WorkStyle="Simplex">
    <Position MarginBottom="36.0" MarginLeft="36.0"
     RelativeBox="0.00000 0.00000 1.00000 1.00000" />
    <BinderySignatureRef rRef="_4711" />
    <StripCellParams Mask="DieCut" TrimSize="744.12 752.04" />
    …
</StrippingParams>

<BinderySignature BinderySignatureType="Die" Class="Parameter" ID="_4711"
 Status="Available">
    <DieLayout>
        <FileSpec URL=" file://Fileserver1/DDES3/6-Folding.dd3" />
            <Station StationAmount="6" StationName="DES1" />
    </DieLayout>
</BinderySignature>
```

where *Fold* is defined as the default value. Insofar as this attribute was not necessarily included in the previous chapters, it is now. If the resource is from the type *Die,* it must contain (a reference to) the *DieLayout* element. The URL of a file is specified there ("6-fold-box.dd3"), which serves to describe the die lines. Moreover, in the example the number of stations is given, which corresponds equally to the number of units on the press sheet.

Figure 12.7 shows an example of a *BinderySignature* resource in which the unit arrangement is defined via a step-and-repeat matrix. As a result, the *BinderySignature* type is also *Grid.* The number of columns and rows of the matrix is limited with the attribute *NumberUp.* So each matrix entry corresponds to a folding box cutout. The same attribute is also applied to commercial printing when an imposition pattern is to be defined. Then each matrix entry corresponds to a pattern page. See Section 8.2, Figure 8.11. With the *FrontPages* attribute, each 0 represents a unit of the same orientation whereby the sheet contains only one type of label (otherwise multiple *BinderySignatures* would need to be defined).

```
<BinderySignature ID="_4712" Status="Available" BinderySignatureType="Grid"
 NumberUp="2 3"...>
   <SignatureCell FrontPages="0 0 0 0 0 0 " Orientation="Left" />
</BinderySignature>
```

Figure 12.7
Step&Repeat for labels

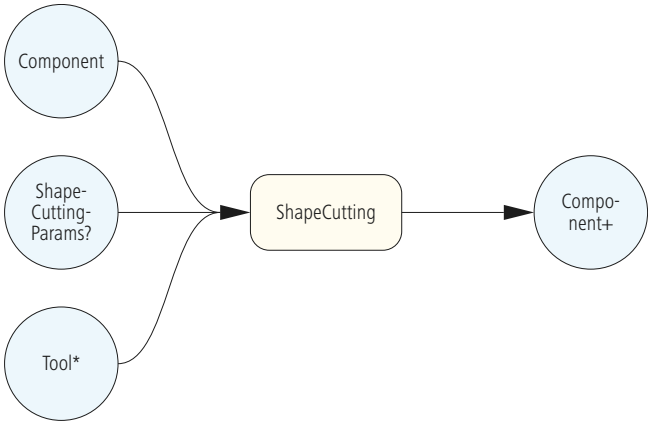
12.2 Punching and Folded-Carton Gluing

With punching, shaped parts, the so-called cutouts, are stamped out of a press sheet by means of an automatic punch. Then, depending on the configuration in a break-apart station, the second station of the punch automation, the punch waste is eliminated via the break-apart tool. Finally the units are separated, which can be done inline or offline.

These operations are collected together in JDF under ShapeCutting. The tool (*Tool*) discussed in the last section is the most important, as is the press sheet (*Component*). Figure 12.8 shows the model. Naturally such input resources, which are permissible for each process node, could, for example, come along as *NodeInfo* or *Device*. Furthermore it should be pointed out that blind embossing can be carried out with punch automation, and even hot foil stamping modules could be integrated into the machines. In these cases the entire process could be described through a combined process which includes *Embossing* and *ShapeCutting*.

The example of a ShapeCutting process, which can be seen as a fragment in Figure 12.9, is even simpler to build than the Figure 12.8 model. Namely, it contains no *Tool* input resource. As already stated in the beginning of Section 12.1 there are currently no production systems that make this information available. In the commercial print area, a *Blockname* is identified in a *CutBlock* for the *Cutting* process (see Figure 11.3); for the packaging sector it identifies a unit that is punched out. The printing process *Conventional-Printing* produces a *Component,* which means a stack of press sheets with the *SheetName* "Folded Carton – Press sheets," out of which then the six folded cartons (blocks) are punched.

Figure 12.8
JDF die/punching/stamping model



The cutouts are then placed in production in the feeder of the folder gluer. Then the crease lines are folded by means of a primary crusher, folded back and finally laid flat. To this end, the primary crusher serves to make it easier to open the cartons later in the packing machines. Finally, most of the glue tabs or glue edges are glued. Thereafter, the cartons are closed. Finally they are pressed together, glued, and collected and delivered as a unit where the cartons are repacked into boxes.

It is common for carton gluing machines to be individually made to order for the user. So there are special modules for cartons which in the same sense are unique such as those with conical forms or for chocolate boxes. In this case, components are built from entirely different manufacturers. Often machines are even specially constructed for an intended purpose. Otherwise as a rule the conversion of a folder gluer is connected to many mechanical settings that the user has to perform. For the potential case of a repeat order, the setting values of the machine are simply noted. Nevertheless, there are already machines that can automatically preset themselves to a certain degree through servo motors and software, though not by hand wheels, levers, and hex keys, which was traditionally the case.

The adjustment of a more typical box gluer machine goes roughly as follows. Depending on the format and the thickness of the working cutouts, there must be a variety of settings made at the feeder. Furthermore, one or more crush stations are set up accord-

```
<JDF ID="_343" JobPartID="_1002.0" MaxVersion="1.3" Status="Part" Type="
 ShapeCutting " Version="1.3" >
    <ResourceLinkPool>
        <NodeInfoLink Usage="Input" rRef="_344" />
        <ComponentLink Usage="Input" rRef="172"…>
        <DeviceLink Usage="Input" rRef="_339" />
        <ComponentLink Usage="Output" rRef="_152"…>
        <ShapeCuttingParamsLink Usage="Input" rRef="_772">
            <Part SheetName="Folding Sheet" SignatureName="SIG1" />
        </ShapeCuttingParamsLink>
    </ResourceLinkPool>
    <ResourcePool>
        <ShapeCuttingParams Class="Parameter" ID="_772"
         PartIDKeys="SignatureName SheetName BlockName" SheetLay="Left"
         Status="Available"…>
            <ShapeCuttingParams SignatureName="SIG1">
                <ShapeCuttingParams SheetName="Folding Sheet">
                    <ShapeCuttingParams BlockName="Block1" />
                    <ShapeCuttingParams BlockName="Block2" />
                    <ShapeCuttingParams BlockName="Block3" />
                    <ShapeCuttingParams BlockName="Block4" />
                    <ShapeCuttingParams BlockName="Block5" />
                    <ShapeCuttingParams BlockName="Block6" />
                </ShapeCuttingParams>
            </ShapeCuttingParams>
        </ShapeCuttingParams>
    …
    </ResourcePool>
</JDF>
```

ing to cutout type and thickness. Then width and length of the glue strip and the adhesive applications rate must be adjusted in the gluing units. Naturally the positions of the glue units are also defined. The subsequent folding then requires a specific setting, exactly like the subsequent pressing and collecting of the cartons.

Figure 12.9 Die/punching/stamping process

A JDF/JMF network of carton gluing machines can—as with the other machine—bring several benefits:

- The order data is delivered directly to the device.
- The manufacturing data is directly reported back to the production system.
- The default data can be taken up so that the setup times are shortened.

The third point, however, still plays a relatively minor role since there are only a few machines on the market with automatic presets. Looking ahead to the future should nevertheless highlight the JDF possibilities.

The *BoxFolding* process represents work running on a carton gluing machine. *BoxFolding* input resources are essentially the *Components* (cutouts) which arose as a result of the stamping process, and the *BoxFoldingParams* are parameters which can preset the machine. An example can be seen In Figure 12.10. In it are folding actions (*BoxFoldActions*) related in the previous case to a standard folding carton. Altogether there are ten standards predefined in the JDF specification; beyond that, individual folded cartons can also be defined. The BoxFoldingType here has the value *Type01,* which corresponds to a carton construction as seen in Figure 12.11. The values of the attribute BlankDimensionsX and BlankDimensionsY are also marked in Figure 12.11. The zero point is at the bottom left and the numbers are given in DTP points. Naturally, these values can vary and here describe only a very specific carton of this type. With the carton type, the folding operations are also already known; attention should be paid to the cutouts being laid down in the feeder with the printed side toward the bottom:

- Creased on line L0 from left to right
- Creased on line L2 from right to left

And after the glue application from the bottom on the left tab

- Folded on line L1 from left to right
- Folded on line L3 from right to left

The glue line (*GlueLine*) is not defined in this example, but the *BoxFoldAction* element corresponds exactly to the order of the fold operations. With *FoldIndex,* only the first value here is critical, the

Figure 12.10
Preset data for a carton gluing machine

```
<BoxFoldingParams BlankDimensionsX="43 388 502 848 960"
 BlankDimensionsY="43 78 156 465 543 578 621" BoxFoldingType="Type01"
Class="Parameter" ID="_771" PartIDKeys="SignatureName SheetName BlockName"
Status="Available">
    <BoxFoldingParams SignatureName="SIG1">
        <BoxFoldingParams SheetName="Folding Sheet">
            <BoxFoldingParams BlockName="Block1" />
                <BoxFoldAction Action="LongPreFoldLeftToRight"
                 FoldIndex="0 -1" />
                <BoxFoldAction Action="LongPreFoldRightToLeft"
                 FoldIndex="2 -1" />
                <BoxFoldAction Action="LongFoldLeftToRight" FoldIndex="1 -1" />
                <BoxFoldAction Action="LongFoldRightToLeft" FoldIndex="3 -1" />
        </BoxFoldingParams>
    </BoxFoldingParams>
</BoxFoldingParams>
```

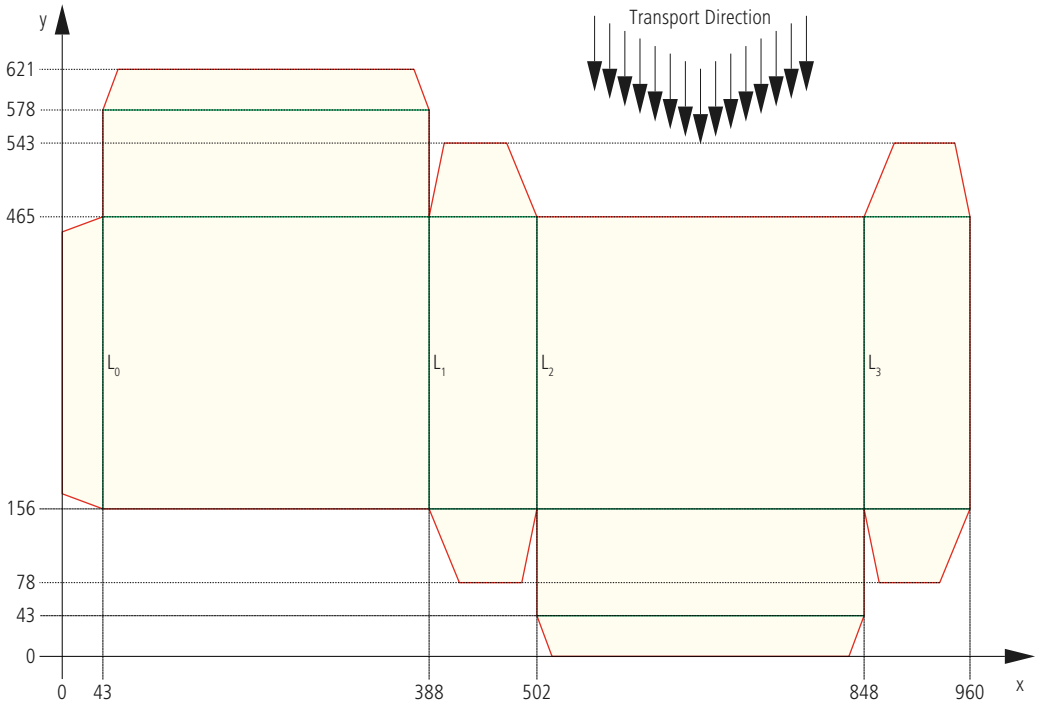


Figure 12.11
Standard folding carton form of Type 01

value "-1" indicates that the folds should follow along the entire length in the direction of travel.

12.3 Barcode

At first glance, a barcode seems to be only another graphical element on the package, which is allowed for and naturally also realized with the design. In fact, with standard business layout programs, one can create and place barcodes on the graphic design. The generation of barcodes is indeed not directly supported by the layout programs, but there are plug-ins for this purpose. Alternatively, one can use standalone programs, in which the digits for a barcode can be entered then a corresponding barcode font is created. Such a font then contains only a single character that corresponds to the previously defined barcode pattern. Finally the font can be installed in an operating system and subsequently used by any layout program. Since in the packaging sector, all of the fonts are commonly converted into general graphic paths, then the barcode font must not be passed on to the print provider so the barcode is already compiled with the graphic design and the printer will have nothing to do with it.

```
<LayoutElementProductionParams Class="Parameter" ID="_300" Status="Available">
    <LayoutElementPart>
        <BarcodeProductionParams>
            <IdentificationField Encoding="BarCode" EncodingDetails="EAN_13"
             Value="0123456789128" />
        </BarcodeProductionParams>
    </LayoutElementPart>
</LayoutElementProductionParams>
```

Figure 12.12
Definition of a barcode

This practice has one drawback: Especially in flexographic printing there is a considerable line broadening which can occur that could cause the barcode to no longer be readable electronically. This can lead to immense spoilage because not only is the packaging unusable but also the packaged product is unusable if the mistake is noticed after packing. So a line width compensation for the barcode must be in the calculation beforehand. The level of compensation depends, however, on different factors (print technology, substrate, ink, waste, etc.), so that only technical packaging prepress specialists can make the requirements. Therefore, as a rule, barcodes are not set in graphic design; instead they are added later with plates or printing services with special packaging prepress programs.

So that barcode details are depicted, information from the client must be passed through to production. This communication can be made easier with the help of JDF.

The printer receives the EAN code as a number sequence from the client and enters it into a suitable MIS System. From there the information is passed on in JDF to packaging prepress software so that the employee there no longer has to enter the codes him/herself (which always carries the risk of mistaken input). A corresponding entry is found in Figure 12.12. The *LayoutElementProductionParams* form an input resource of the *LayoutElementProduction* process. In this process either the production of layout components like images, graphics, text, or barcodes are described or created as complete page/units in a layout program. On the other hand, an MIS will not define such a detailed process, but rather a *GrayBox* is described like the previously mentioned *PrePressPreparation* or the *ContentCreation*. In the subelement—a grandchild in the sequence of generations—one finds then the *BarcodeProductionParams* which contains the parameter for the production of barcodes. Each barcode is then listed in the subelement *IdentificationField*. In the present example, the *Type* of the barcode (*EncodingDetails*) and the value (*Value*) is given. This resource may optionally include additional subresources and accord-

ingly create references to additional resources as can be seen in Figure 12.13. Then the *BarcodeReproParams* resource can, for example, contain information about height and width of the barcode and, as soon as they are known, the necessary line width compensation is written in *BarcodeCompParams.*

A further advantage of barcode metadata potentially lies in quality assurance. The value can be compared with a (product) database, so that it is guaranteed that the barcode actually matches the product. That would not be so easy with a placed graphic element.

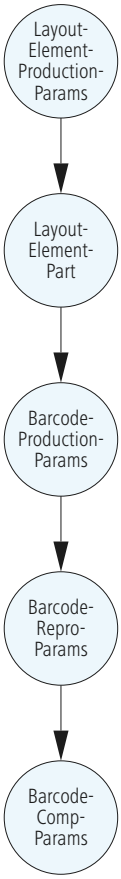


Figure 12.13
"Resource tree" for the LayoutElementProduction process

13 JDF/JMF Projects

While the last twelve chapters could be reasonably backed up by facts, we now move ourselves onto very thin ice. JDF projects are very unique, and the project management methods are not always the same. So it is difficult to give general rules or even tips for a new implementation or upgrading of JDF workflows. Also are our observations are surely coined at least in part from our personal experiences. Finally, we also want to portray the implementation as neither too rosy nor too gloomy. In order to make it easier for us, we divide JDF projects into the three types:

- Workflow implementation with modules from a single vendor
- Workflow implementation with modules from multiple vendors
- Workflow implementation with modules from several vendors and proprietary software customizations

But before we discuss these three cases in the following sections, we want to make a few general remarks even if it can be perceived as the proverbial "Sunday sermon." Above all, at the beginning of such a project, it is important to minimize the expectations of a JDF workflow. Namely, what one can and can't expect from a JDF workflow must be considered and depend largely on the requirements of the particular operation. The less one knows about the topic the harder it naturally is to correctly evaluate the complex situation, especially as so often the devil is in the details. Sometimes miracles are expected, like, for example, that the current business organization suddenly steers itself into orderly paths. Or the employees in the pressroom are disappointed that the much-discussed "JDF Workflow" is not even noticed. Sure, now they receive a couple of default values, but was that all? And one must admit that many of the "beautiful improvements" which are reached with JDF/JMF are also previously available with vendor-specific solutions. Then there was already plant data collection and also cross-departmental exchange of data from order data and preset values. Nevertheless, it is our opinion the JDF/JMF concept is forward looking and much more broad and yet, for that purpose, as vendor overlapping as anything previously existing. The potential is surely not yet exhausted.

To be clear about the expectations means, in other words, defining the arguments for the engagement in a JDF/JMF project. Should the cost transparency be increased, the internal and/or external communication simplified, the failure rate reduced, the production

steps improved and possibly even automated, planning optimized, and the costs reduced? Or everything all at once? The problem is the same here for all, that one is glad to maintain these buzzwords and not go into the details—where, and how exactly, which communications should be improved and so on. The reasons are too abstract and nebulous hopes are tied to it, which then of course cannot be fulfilled.

It must also clearly define goals both in technical as well as from a business point of view. The basis is the analysis of the actual state. The examination of the workflow can even be valuable alone, even if a result may show that the JDF networking for a business is not suitable. Because in this case the analysis uncovers deficiencies and corresponding solutions are found which go in a totally different direction. Apart from that, the analysis helps to find realistic goals for a JDF engagement. In short, the printer should write a functional specification on the proposed JDF interfaces and confirm the compliance of the demands from the potential suppliers.

A JDF/JMF project must be viewed as long-term (key word "scalability"). First it is simply just more work and only later brings benefits for the company. Therefore, all also agree that one should approach such a networking project in multiple phases and define milestones. This is how the milestones could be made out (which naturally must be much more concretely defined):

- Networking of MIS to prepress
- Networking of MIS/prepress to the pressroom
- Plant data collection from production to MIS
- Integration of selected finishing machines in the network
- Customer connection

As a rule MIS forms the beginning of a JDF network. Where exactly one then proceeds depends largely on the available equipment. Because, for example, one will not purchase a new folding machine with a servo motor only because this (sensibly) fits in a JDF environment. That would certainly not be right economically. Instead, it is more likely existing machines, to which JDF interfaces are available, should be upgraded or JDF compatibility should be valued when overdue on a new purchase.

In all three of the cases presented, it requires the active participation of the users, whereby the degree of necessary participation

increases in accordance with the task list. We would like to quote James Harvey, the executive director of the CIP4 Organization [21]:

> Implementing process automation, even with JDF, requires active participation by the printer's staff. Different vendors have different philosophies for how they are implementing process automation, and printers may find themselves acting as project managers to work out issues that may arise when multiple systems are integrated.

Participation should therefore not only be ordered from above, but the work atmosphere should preferably be characterized by team spirit and joy of innovation. Then workflow management systems (with or without JDF) change and blur the boundaries of the responsibility and departments. So typically we find in a JDF environment that a skills shift occurs from prepress through order management, and, where need be, employees must then change their position. The increase of transparency naturally also results in the objective evaluation of job performance. Employees will not support the project if a climate of mistrust and competition prevails. Just the actual analysis can cause resistance with the employees. Finally it must also be decided by consensus who has access to which digital data and also who has read access only or may also have write access.

13.1 Workflow Implementation with Modules from a Single Vendor

Naturally the effort to set up a JDF workflow is reduced when only one manufacturer is involved. Nevertheless it is not to be neglected because the adoption to the production methods and devices of a business must naturally take place. This also can and should not be carried out by the manufacturer under its own auspices because this means that with each small change in the service one must take it.

A WMS administration in the print shop must also network, for example, new devices in the workflow, manage users, and set default values for production. Since the different JDF modules must run off of several distributed servers, an IT manager (or several) must be responsible for these computers and for the network and control their access rights to one another. In practice, things may happen like some required service suddenly quitting or a computer, due to automatic operating system update, may reboot. Only if each employee knows how to respond to such disruptions will you be able to enjoy the JDF workflow for a long time. For a stable configura-

tion, an IT administrator and a workflow administrator are required, though it is better if this knowledge lies in hand. Since this position is extremely important, a second person (redundancy) at least should be available to control these two systems. Depending on the scope of the JDF network this can be a very difficult task. For a far-reaching workflow system, even complete oversight at the administrator level is no easy task for the responsible employee. And even with the manufacturers one tends to meet specialists rather than technically accomplished generalists.

Even if only one manufacturer is responsible for the workflow installation, a JDF implementation changes over time and is also usually implemented in stages. Because the JDF workflow systems are still in the flux of constant changes and software patches, updates and extensions must regularly be recorded and subsequently tested. It can be helpful here to work with virtualization or disk images so that in the case of error it is easier to go back to an earlier version.

When a new JDF workflow is to be installed in a business, the configuration should be reviewed by the vendor as soon as possible or one of its reference customers should be reviewed and tested with the typical orders of the print shop. This is because automation might function very well for one product or product group, but not for others. This can be for a variety of reasons. For example, an MIS that can access only one fixed folding catalog, not a specific imposition template, naturally cannot pass the information on to production. Or certain details that describe a type of production are simply not passed along or will be faulty. Some matrices that show which systems are compatible with each other are only an initial, rough assessment. Nevertheless, compatibility must still be tested in detail. Insofar as perhaps one will use workflow automation for only one part of the orders, others will still require much manual intervention.

The introduction of a JDF workflow requires, as already mentioned, well-defined production rules. But these could also inhibit short-term changes in orders from being easily quickly and quickly transacted. Realizing subsequent changes in a strongly regulated and automated workflow is, at times, a contradiction in terms, however, a frequent necessity in everyday practice. The change effort with the JDF workflow can be higher than before with a run ticket depending on how far the order has already been processed. For a complex JDF workflow system, many computers, the network, and various programs must function simultaneously. Then, naturally, there are failures, and failure containment is not easy, since a hardware defect on a network switch, for example, can cause an

error in the application and a corresponding (misleading) failure message. Thus it may be a long time until an error can be remedied, but in order to keep production running, it can be helpful to have an emergency workflow defined. For example, the press in this circumstance should be able to print with one set of plates without that preset data. That may sound obvious, but it is not, because the impact is sometimes quite hidden. If, for example, the positions of the register marks are not provided in the JDF, the inline registration in the press may no longer work properly, and so on. Or one of the JDF workflow attached digital presses should, if necessary, be able to process directly if the JDF communication causes problems.

Not every automation scenario that is conceivable and offered always make sense. That goes, of course, not only for the economics but also for technical aspects; there are limitations in this regard. If the start of subsequent processing is dependent on approval from the customer, or if the person responsible for production is made dependent on certain events, this can unnecessarily hamper the production process.

So, for example, the preparation of a job in the press control panel could be made to depend on the completion of the printing plates, which means the JDF data with the parameters for the print job would then first be sent or copied to the hot folder of the MIS, or alternatively the productions system holds information about the completion of the corresponding press plates from the plate burner. This entirely sensible arrangement, however, then creates problems if the response of the plate burner—for whatever reason—is disrupted. Then, indeed, the set of plates is available and in principle all of the preset data are also available for the press and nevertheless it goes no further because either the system or at least the press did not receive the appropriate information.

13.2 Workflow Implementation with Modules from Multiple Vendors

Everything that was mentioned in the last section naturally applies in the case that multiple providers participate in a JDF workflow. However, a few additional considerations arise.

The interface problem is well known in digital technology: if a software module from one vendor generates a data format for a program of another, there is no clear responsibility if the communication doesn't function. This also goes for JDF as well as for any other format. Since JDF and JMF are still comparatively new (for exam-

ple, in comparison to PDF) you will encounter incompatibilities with these formats. Above all, an in-depth analysis is recommended with the new installation and also with version updates. But this is actually only possible for the JDF data exchange and also only then if the JDF is passed via a hot folder and not sent in a MIME packet over HTTP. Indeed, you could intercept the packets, but this is somewhat cumbersome. The same goes for JMF messages. Sometimes you can adjust with workflow components whether the communication should run via files or HTTP. You should decide here on the file type when you wish to investigate interfaces.

If you don't yet have the JDF devices installed, you want to check their communication among one another so you must ask to create JDF from one vendor and ask another to read it. In another case, you can analyze the interface directly. Many vendors also have white papers that specify which requirements a JDF file must satisfy in order for the software to be read and processed. If not, the vendor can provide sample data which can be read successfully for the software that imports JDF.

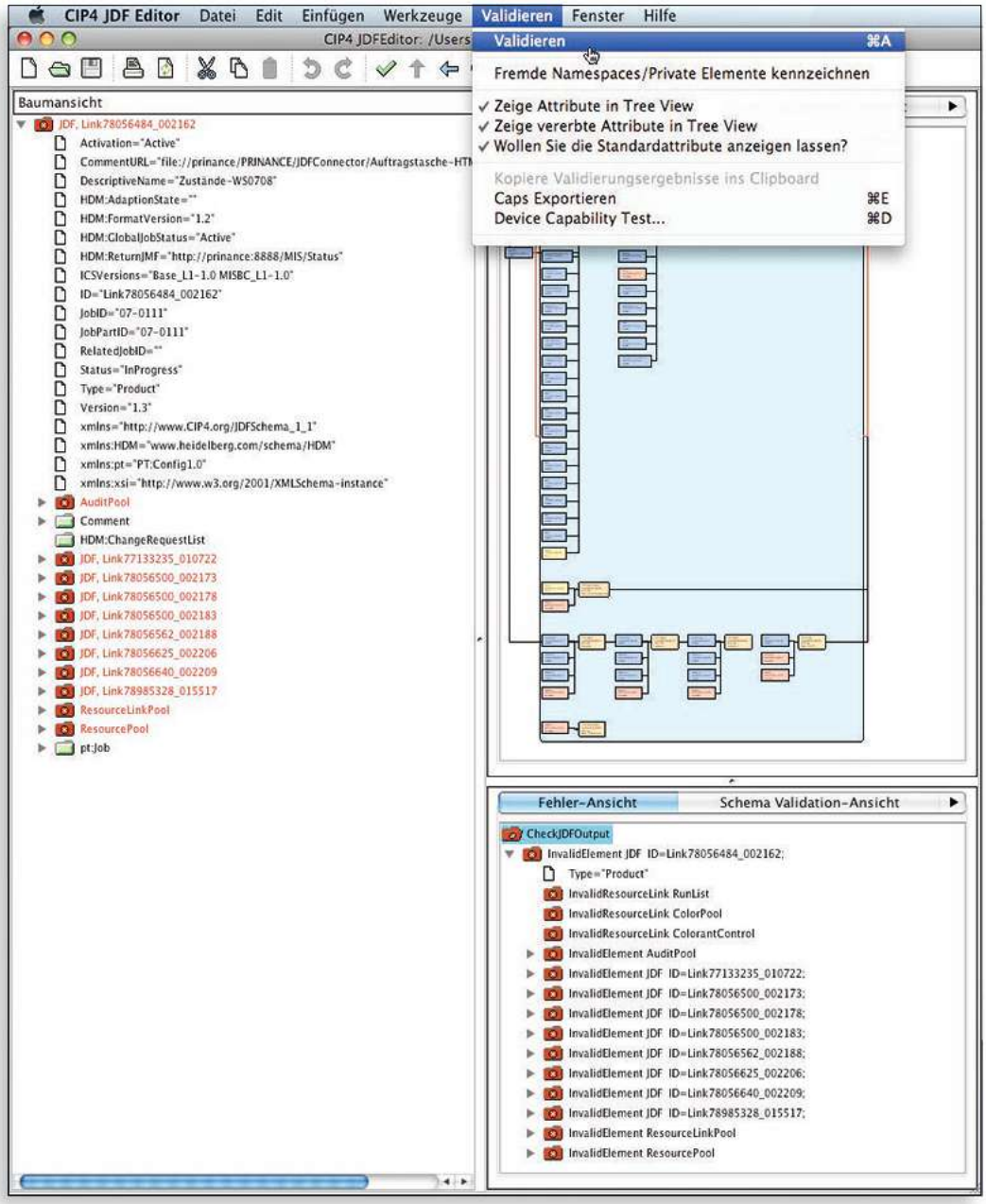
An analysis of the JDF file could begin with a validity check (see Section 5.2). This can be done on the CIP4 website under "Technical Resources," choose the menu item "Check JDF Server," and then load the JDF file with the file service "Check JDF." A report then shows possible errors first, for example missing attributes which are obligatory or links that are not able to be resolved. Optionally, private extensions can also be marked as errors. At the same site you can choose the "Fix JDF" service and a JDF file is repaired and brought to a defined version. As expected, the automated repair has its limitations. If you are a CIP4 member, then you can download the program "CheckJDF.exe" and an additional DLL file under "Technical Resources," "Downloads," "Internal Source." You can also validate JDF files with the "JDF Editor" which can be downloaded free from the CIP4 website (See Figure 13.1).

It can also be helpful to show the values of the *ICSVersions* attributes and perhaps to consider if the JDF program is based on different ICS versions. The same goes also for the JDF versions listed in the *Version* attribute. However, experience (or also a healthy mistrust) shows that the specified version levels do not always agree with reality.

There can be many different reasons, when reading a JDF file, a file aborts because of a failure. Perhaps a resource is not directly on the JDF node that is required, instead—fully legally—it stands only at the root level and is not found (see Figure 6.5 in Section

6.1). Such problems are difficult to analyze without the source code of the importing software to debug it. It helps the most to begin first with tests with very simple JDF files and gradually become more complex. On the other hand, it is easier if only single values that should actually be transmitted are not shown in the JDF reading software. Then one can easily see whether the cor-

Figure 13.1
Validation function in the CIP4 JDF Editor



responding element/attribute exists therein and can at least pass the buck to the two vendors involved. Although, of course, this not always enough to correct the manufacturer's defect in a reasonable amount of time.

JDF files can, by the way, be quite large. The examples in this book are always only excerpts. In reality, in certain circumstances, JDF files can be many A4 pages long and contain several dozen JDF nodes, which, of course, complicate the analysis.

We wish to clearly say yet again: Although JDF is an industry standard, unfortunately the interfaces do not function without friction. This may be due to private data which are transported in the JDF (see "Standardization" in Chapter 2); as a rule it is more likely the JDF structure and the missing or false information is in the JDF data. JDF interfaces between modules of different vendors do not function by themselves alone, but must be adapted. Indeed these are tested between the manufacturers, but there are always situations, especially with complex jobs, where errors appear. And these errors remain inexplicable to users since they have neither the time nor the tools and perhaps also not the know-how to satisfactorily analyze a problem.

13.3 JDF/JMF Programming

This section is addressed to the people who will be starting a software project and also have a little experience with the JAVA or C++ languages, but not yet with the programming of JDF applications. It is not directed at professional software developers and experts who are interested in special tips and tricks.

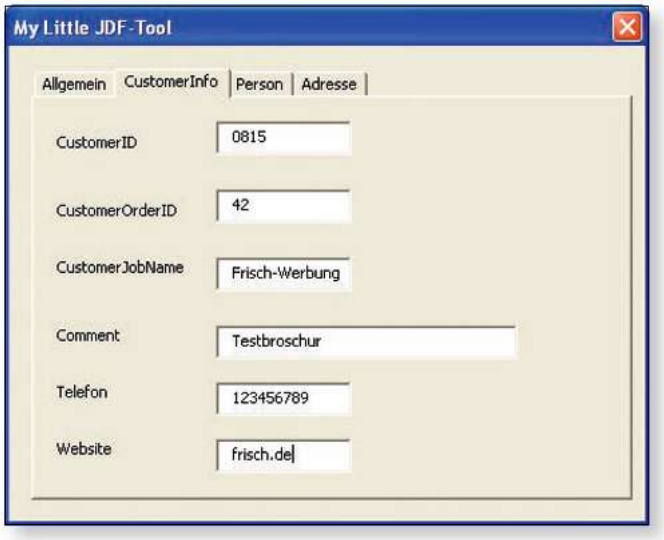


Figure 13.2
GUI from a simple JDF application

Naturally, JDF importing or exporting software can be created in any language. For example, you can easily create a user interface like in Figure 13.2 with Visual Basic for Applications and also write a small program that takes the input vales from the dialog box and creates a JDF file with *CustomerInfo* as a resource (corresponding to the example in Figure 6.8). Only then you can not resort to a library and the code is

laborious to create and very prone to errors. An excerpt of such a quick-and-dirty program is seen in Figure 13.3. In fact, this example program creates no fully correct JDF code because, for example, both of the obligatory attributes *Class* and *Status* are missing. Of course, it would be extended accordingly. It is even more arduous to write software that reads this type of JDF if no suitable library is available. In all, such action is discouraged if one wants to develop reliable software.

Fortunately, CIP4 has created exactly such libraries, but only for JAVA and C++. Therefore, you should develop JDF software in one of the two languages. We will only discuss the JAVA library in the following section. First, decide on an (integrated) programming environment. The list of possibilities is long—we programmed the example specified here with the help of the Eclipse development Environment [17], but sometimes we also worked with NetBeans IDE.

In Eclipse you first define a new "Project," where libraries can be bound in the form of jar files. The following libraries are required for the development of JDF applications—they can be downloaded from the Internet:

Figure 13.3
Cumbersome VB program that can create JDF

```
Private Sub write_CustomerInfo()
    CustomerInfoID = 11101
    Print #1, " <ResourcePool>"
    Print #1, " <CustomerInfo CustomerID=" & """ & CustomerID & """ & "
     CustomerJobName=" & """ & CustomerJobName & """
    Print #1, " CustomerInfoID=" & """ & CustomerInfoID & """ & "
     CustomerOrderID = " & """ & CustomerOrderID & """ & " > "
    Print #1, " <Comment>" & Comment & "</Comment>"
    Print #1, " <Contact Class=" & """ & "Parameter" & """ & "
     ContactTypes=" & """ & "Customers" & """
    Print #1, " ID=" & """ & ContactRef1 & """ & "
     Status=" & """ & "Available" & """ & ">"
    Print #1, "<Person FirstName=" & """ & FirstName & """ & "
     FamilyName=" & """ & FamilyName & """ & "
     NamePrefix=" & """ & NamePrefix & """ & ">"
    Print #1, " <ComChannel ChannelType=" & """ & "Phone" & """ & "
     Locator=" & """ & Telefon & """ & "
     Status=" & """ & "Available" & """ & "/>"
    Print #1, " <ComChannel ChannelType=" & """ & "WWW" & """ & "
     Locator=" & """ & Website & """ & "
     Status=" & """ & "Available" & """ & "/>"
    Print #1, "</Person>"
    Print #1, "<Address City=" & """; City & """ & "
     Street=" & """ & Street & """ & " Country=" & """ & Country & """ & "
     PostalCode=" & """; PostalCode & """ & "/>"
    Print #1, " </Contact>"
    Print #1, " </CustomerInfo>"
    Print #1, " </ResourcePool>"
End Sub
```

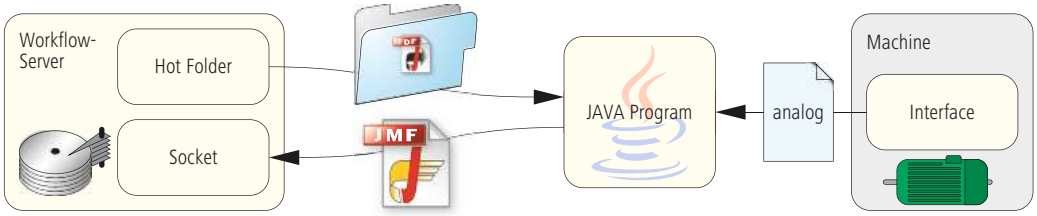


Figure 13.4
Project to connect JDF/JMF to a machine

- XercesImpl.jar (our Version 2.9.0),
- Commons-lang.jar (our Version: 2.3),
- JDFLibJ.jar (our Version 2.1.3.4—see www.cip4.org).

In our small example we have programmed an application which includes the following features (see Figure 13.4):

- JDF files are read and displayed from a selectable hot folder.
- With a double-click on a job, some details of the job are extracted and displayed (JobID, Order Name, Sheet Size, Run Length).
- In this window, four command buttons are defined: "order begin," "waste paper," "good sheets," "end of job."

Figure 13.5
Realization of a hot folder

```
public Main window (){ // Constructor
    ...
    ListWin = new Thread(this);
    ListWin.start();
    ...
}
public void run(){
    while (true){
        try{
            jobliste.removeAll();
            dateien = ordner.list();
            for (int i =0; i< file.length; i++){
                jobliste.add(dateien[i]);
            }

            this.add(jobliste);
            repaint();
            Thread.sleep(10000);
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

- After clicking the "order begin" button, all sheets are counted with a photo sensor which all go through a printing press without a JDF interface; one chooses "waste paper" or "good sheets" so the appropriate meter is set.
- If the "end of job" button is pressed, the program sends a JMF message to a configurable IP address and TCP port.

Naturally, we cannot show the entire source code here, instead only a couple of interesting snippets.

In Figure 13.5 one can see a thread with the name *ListWin,* which regularly reads file names out of a hot folder and then goes back to sleep. It is started in the main window. If the thread is activated, the entries from the job list are first removed, then the names of the files in the hot folder are reentered into the list and, finally, the result is displayed in the main window.

The parsing and reading of attribute values is shown again in Figure 13.6. First the JDF document *JDFDoc* is parsed and the root element is identified with the name Root. Attributes of the root element, like the *JobPartID,* are very easily pulled out. Also the *ResourcePool,* which we have called RSP, can be found via the root and from it the specific resources like *CustomerInfo.* One can clearly see here how easy the whole thing is; we have partly left out a couple of security questions for clarity. So, for example, we simply assume that the resource *CustomerInfo* exists and this, in turn, contains the attribute *CustomerJobName.*

Figure 13.6
Parsing and reading of attribute values

```
JDFDoc JDFDocument = JDFDoc.parseFile(pfad + dateiname);
final JDFNode Root = JDFDocument.getJDFRoot();
...
String JobPartID = Root.getJobID(true);
...
JDFResourcePool RSP = Root.getResourcePool();
final JDFResource CustomerInfo = RSP.getResource("CustomerInfo",0,"");
CustomerJobName = CustomerInfo.getAttribute("CustomerJobName");
```

The last thing we want yet to introduce is the source code in Figure 13.7 for sending a JMF signal. The JMF message and the HTTP header are assembled therein. A *DeviceInfo* and a *JobPhase* node element are created and filled with attributes. Finally, everything is written in the string *Content.* There is nothing more to see here, like a new socket is created with the help of IP and port address, and into it the JMF message and HTTP header are written. The result of this small program is a JMF signal of the type in Figure 13.8.

```
JMFMessage(String JobID,String JobPartID, Long goodCount){
    String DeviceStatus = "Running";
    //Construct a JMF-Message, i.e. the content of the http-package
    JDFJMF JMF = JDFJMF.createJMF(EnumFamily.Signal,EnumType.Status);
    KElement DI = JMF.appendElement("DeviceInfo");
    DI.setAttribute("DeviceStatus", DeviceStatus);
    KElement JP = DI.appendElement("JobPhase");
    JP.setAttribute("TotalAmout", goodCount.toString());
    JP.setAttribute("JobID", JobID);
    JP.setAttribute("JobPartID", JobPartID);
    String content = JMF.toXML();

    //construct a http-header
    long length = content.length();
    String header = "HTTP/1.1 200 OK" + "\nContent-Length: " + length +
     "\nContent-Type: text/html";
...
}
```

```
HTTP/1.1 200 OK
Content-Length: 509
Content-Type: text/html

<?xml version="1.0" encoding="UTF-8"?>
<JMF xmlns="http://www.CIP4.org/JDFSchema_1_1"
 TimeStamp="2008-12-02T18:32:24+01:00" Version="1.3">
 <!--Generated by the CIP4 Java open source JDF Library version : CIP4 JDF
 Writer Java 1.3 BLD 40-->
    <Signal ID="m081202_063224609_000000" Type="Status"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="SignalStatus" />
        <DeviceInfo DeviceStatus="Running">
            <JobPhase JobID="_08-0157" JobPartID="_08-0157" TotalAmout="300" />
        </DeviceInfo>
</JMF>
```

Figure 13.7 (top) Source code for the sending of JMF signals

Figure 13.8 (bottom) JMF signal with HTTP headers

In certain cases one would perhaps not like to write an actual JDF/JMF application but instead maybe only transform a vendor-specific software XML interface to a JDF interface or maybe only slightly modifying JDF files. Then an XML document must also be transformed into another XML form. This task can also be done using the **Extensible Stylesheet Language Transformation** (XSLT). This requires you to write an XSLT document, which then controls the document transformation. However, since there are many books (for example [9]) on this topic, we do not wish to go farther at this point.

Bibliography

[1] Adobe System:
Adobe PDF Print Engine 2, White Paper,
http://www.adobe.com/de/products/
pdfprintengine/ (2008)

[2] Adobe System Incorporated:
XMP Specification,
http://www.adobe.com/devnet/xmp/ (2004)

[3] Adobe System Incorporated:
TIFF Revision 6.0,
http://partners.adobe.com/public/developer/en/
tiff/TIFF6.pdf (1992)

[4] Adobe System Incorporated:
PostScript Reference Language, 3. Ausgabe,
Addison Wesley Publishing Company,
ISBN 0-201-37922-8 (1999) oder
http://www.adobe.com/products/postscript/
pdfs/PLRM.pdf

[5] Adobe System Incorporated:
PDF Reference sixth edition, Adobe Portable
Document Format, Version 1.7,
http://www.adobe.com/devnet/pdf/pdf_
reference_archive.html (2006)

[6] Adobe System Incorporated:
Portable Job Ticket Format, Version 1.1,
Technical Notes # 5620 (1999)

[7] ANSI:
IT8.6-2002 (R2007) Grafic technology –
Prepress digital data exchange – Diecutting
data (DDES3)

[8] Apple Inc.:
AppleScript Overview,
http://developer.apple.com/documentation/
AppleScript/Conceptual/AppleScriptX/
AppleScriptX.html (2007)

[9] Behme, Henning und Mintert, Stefan:
XML in der Praxis:professionelles Web-
Publishing mit der Extensible Markup Language,
Addison Wesley, ISBN 3-8273-1636-7 (2000)

[10] Bohan, Mark et al:
Automation and JDF Workflow, 2006 TAGA
Proceedings

[11] CGATS.20:
Variable printing data exchange using PPML
and PDF (PPML/VDX),
www.npes.org (2002)

[12] CIP4:
Interoperability Conformance Specifications,
Version 1.3 www.cip4.org/ (2007/2008)

[13] CIP4:
JDF-Specification Release 1.4,
www.cip4.org (2008)

[14] CIP4:
The JDF Marketplace;
http://www.cip4.org/marketplace/

[15] Commerce XML (cXML) User's Guide, Version
1.2.019, http://www.cxml.org (2008)

[16] Dolin, Penny Ann:
Exploring Digital Workflow,
Thompson Delmar Learning,
ISBN 1-4018-9654-5 (2006)

[17] Eclipse Foundation:
Eclipse IDE for Java Developers,
http://www.eclipse.org/downloads

[18] Freund, Jacob; Götzer, Klaus:
Vom Geschäftsprozess zum Workflow,
Carls Hanser Verlag München,
ISBN 978-3-446-41482-2 (2008)

[19] Ghent PDF Workgroup:
PDF/X Plus
www.gwg.org

[20] Hamilton, Eric:
JEPEG File Interchange Format, Version 1.02,
http://www.jpeg.org/public/jfif.pdf (1992)

[21] Harvey, James:
About the JDF User Group
http://www.jdfusergroup.org/
unter: Grafic Arts Information Network
http://www.gain.net [Zugriff April 2009]

[22] Hoffmann-Walbeck, Thomas:
Lehrbuch Digitale Druckformherstellung,
dpunkt Verlag, ISBN 3-89864-182-1 (2004)

[23] International Organization for Standardisation
(ISO) 15930: PDF/X Teil 1 bis Teil 8,
Beuth Verlag (2001 – 2008)

[24] International Organization for Standardisation
(ISO) 12647: Teil 1 bis Teil 7,
Beuth Verlag (2001 – 2007)

[25] International Organization for Standardisation (ISO) 16612-2: Grafic technology – Variable data exchange – Part 2: Using PDF/X-4 and PDF/X-5 (PDF/VT-1 and PDF/VT-2), under development (2008)

[26] International Press Telecommunication Council (IPTC): http://www.iptc.org

[27] Japan Electronics and Information Technology Industries Accociation(JEITA): http://www.jeita.or.jp/english/

[28] JEITA: Exchangeable image file format for digital still cameras: Exif Version 2.2, JEITA CP-3451 (2002)

[29] Kipphan, Helmut: Handbuch der Printmedien, Springer Verlag Berlin, Heidelberg, New York, ISBN 3-540-66941-8 (2000)

[30] Kodak Grafic Communications: "Prinergy 4 Software and Rules-Based Automation", White Paper, http://grafics1.kodak.com/documents/Release%20date.doc (2007)

[31] Koster, Kai: Informations- und Kommunikationstechnologien für Unternehmen, Carl Hanser Verlag, ISBN: 3-446-2118-3 (1999)

[32] Kühn, Wolfgang; Grell, Martin: JDF: Prozessintegration, Technologie, Produktdarstellung, Springer Verlag Berlin Heidelberg, ISBN 3-540-20893-3 (2004)

[33] Kular, Christopher: The Job Definition Format and Print Media, International Journal Of The Book, ISBN: 1447-9516 (2007)

[34] Microsoft: Windows Script Host, http://www.microsoft.com/downloads

[35] Mittelhaus, Michael: Automatisierung in Druckunternehmen, Bundesverband Druck und Medien e.V (bvdm), Art-Nr. 83107 (2005)

[36] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Functional Specification, Version 2.2 (2006)

[37] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Imposition Specification, Version 2.2 (2006)

[38] PODi - the Digital Printing Initiative: Personalized Print Markup Language – Grafical Art Conformance Specification Specification, Version 2.2 (2006)

[39] PODi - the Digital Printing Initiative: Digital Print Ticket – Printing with PPML and JDF, Version 2.0 (2005)

[40] PrintTalk Version 1.3, http://www.cip4.org (2007)

[41] Schwabe, Gerhard et al (Hrsg.): CSCW-Kompendium, Springer Verlag, ISBN 3-540-67552-3 (2001)

[42] St. Laurent, Simon; Fitzgerald, Michael: XML, O'Reilley Verlag Köln, ISBN: 13 978-3-89721-516-0 (2006)

[43] Workflow Management Coalition, www.wfmc.org

[44] W3C (Dave Beckett): Resource Description Framework (RDF), http://www.w3.org/RDF/

[45] W3C: Web Services Description Language (WSDL) Version 2.0, http://www.w3.org/TR/wsdl20-primer/ (2007)

Glossary

AM/FM screen AM is the acronym for "amplitude modulation," a type of plate screen based on halftone dots that vary in size to create the perception of highlight-to-shadow areas of the print. FM stands for "frequency modulation," a halftone technique where the dots are of equal size and tones are created by varying the concentration of dots (highlight areas have sparse concentration of dots while shadow areas have dense concentration of dots).

ASCII85 Coding one Byte stream with 85 different bit patterns, in the ASCII character set with all printable characters represented.

Assembly Placing film or paper elements together in order on a suitable substrate.

Bitmap An image represented by an array of picture elements (pixels), each of which is encoded as one or more binary digits.

Bytemap Data structures for images with (at least) 8-bit color separation (256 shades).

Color depth Number of bits needed to store one pixel or tone.

Color management The use of software to automatically determine the color reproduction characteristics of scanners, monitors, and output devices, and then to automatically make the image settings necessary for optimal color reproduction. Color management attempts to simply color reproduction by putting color expertise and science into software.

Commercial printing Business and private printed matter, such as business cards, stationery, promotional brochures, and catalogs. Periodicals such as magazines or newspapers, books, and packaging are not generally included.

Displacement Also called "shingling." Condition that results when the outside edges of the pages in a saddle-stitched book project out, nearer to the center.

Form proof Also called a "page proof." Usually not a color-accurate printout from one press sheet on a large-format inkjet press to check a printed form with regard to page position and control marks.

Gripper margin The area of the paper on a sheetfed press that is drawn through the press by mechanical fingers (grippers). This margin can contain no image.

High/low folio Gripper fold. Overhang on a folded sheet (front or back) for the automatic opening of a folded sheet that is used by a saddle stitcher with grippers.

Hot folder Unidirectional interface between two software modules, realized with a folder. One software module writes data in the folder on a regular basis while the other checks on the arrival of files.

Imposition Arranging the pages of a job on a printing form during the prepress stage of production so that when printed, folded, trimmed, and bound, all content appears in the proper sequence and orientation. Control marks needed by the press, finishing, and binding operators are also placed on the imposition. Page layout is the process of defining where repeating elements such as headlines, text, and folios (page numbers) will appear on multiple pages throughout a document, while imposition can be thought of as defining where these completed pages will appear on much larger sheets of paper. The binding method affects the imposition and assembly order. Imposition involves not only pages but other print job graphics as well, such as multiple copies of a label. Imposition in a digital print workflow is done using special imposition software.

Imposition layout	A guide that indicates folding sequence, number of pages and signatures used, guide and gripper edges, and cutting and scoring lines for a specific job.
Ink zone presetting	Values for presetting the ink feed supply quantities in offset inking, which can be set by zones.
Inline finishing	Manufacturing operations such as numbering, addressing, sorting, folding, diecutting, and converting that are performed as part of a continuous operation right after the printing section on a press or on a single piece of equipment as part of the binding process. In-line finishing is common in web printing operations.
Interpretation	Transforming a page description language in the RIP into an internal data format, called the "Display List."
Normalizing	Converting customer content data into a unified PDF format.
Nutzen	Positioning the same products, such as page sides or folding cartons, on a printed sheet.
Order management system	Also known as "industry software" or "management information system." Software for costing and job management.
Papierbeginn	Distance from lower edge of plate to the paper of the printed sheet.
Paper class	Classification of printing papers according to ProzessStandard Offset (PSO). ... Or ISO??
Preflight	In prepress, an orderly preventive procedure using a printed checklist or special software to verify that all components of a job (e.g., digital text files and high-resolution image files) are present and correct prior to submitting the document for high-resolution output and to identify potential problems that could cause rework or even rerunning a whole job.
Pressrun	After the machine is set up, the print run that produces good sheets.
Rendering	Conversion of the mathematical descriptions of objects as described in page description languages (for example, PDF) into halftone images.
RIP	Raster image processor. In computer graphics and imaging, the hardware and software configuration used in output devices to determine what value each pixel or spot of output should possess, driven by commands from a page description language such as PostScript. This bitmap is used to guide the laser source in an imagesetter or other recording device (like a platesetter) to image the film, paper, or plate.
Run-length compression	Lossless compression method in which the number of repetitions is stored as a bit value.
Running direction	Fiber direction (grain) of the paper.
Screen frequency	A measurement equaling the number of lines or dots per inch (lpi) on a halftone screen. The higher the screen ruling, the greater the detail in the halftone. Screen rulings range from only a few per inch for large billboards to over 200 lpi for high-quality printing on coated paper.
Screen type	Divided into AM or FM screens.
Screening	The process of converting a continuous-tone photograph to a matrix of dots in sizes proportional to the highlights and shadows of the continuous-tone image.
Setup	Preparation of the machine.
TIFF-B	Common file format for storing bitmaps.

Transfer curve	Also called "tone compensation curve" or "calibration curve process." Curve or table for altering the tonal values in the RIP.
Trapping	(1) Printing a wet ink over a previously printed dry or wet ink film. (2) Creating thin overlaps between adjoining colors to compensate for misregister in prepress and press operations. If the colors are not trapped, the misregister may result in small areas of unprinted paper where colors would normally abut.
Trimbox	Auch Endformatrahmen genannt. Größe des beschnitten Formats des Druckproduktes
Web-to-print	Layout of a printed product via Internet browser, with standard templates used as the basis.

Abbreviations

B2B	Business-To-Business
CFF2	Common File Format, Version 2
CIE	Commission Internationale de l'Éclairage
CIM	Computer Integrated Manufacturing
CIP3	International Cooperation for Integration of Prepress, Press and Postpress
CIP4	International Cooperation for Integration of Processes in Prepress, Press and Postpress
CSCW	Computer Supported Cooperation Work
CTM	Current Transformation Matrix
CtP	Computer-to-Plate
cXMP	Commerce Extensible Markup Language
DDES	Digital Data Exchange System
dpi	Dots per Inch
DTD	Document Type Definition
DXF	Drawing Exchange Format
ECMA	European Carton Maker Association
EPS	Encapsulated PostScript
ERP	Enterprise Resource Planing
Exif	Exchangeable Image File Format
FEFCO	Fédération Européenne des Fabricants de Carton Ondulé
GPS	Global Positioning System
HdM	Hochschule der Medien
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICC	International Color Consortium
ICS	Interoperability Conformance Specification
IPTC	International Press Telecommunication Council
JDF	Job Definition Format
JFIF	JPEG File Interchange Format
JPEG	Joint Photografic Experts Group

JEITA	Japan Electronics and Information Technology Industries Association
JMF	Job Messaging Format
JTP	JobTicket Prozessor
MIME	Multipurpose Internet Mail Extensions
MIS	Management Information System
OMS	Order Management System
PDF	Portable Document Format
PDF/VT	PDF Variable Transactional
PDL	Page Description Language
PJTF	Portable Jobticket Format
PPF	Print Production Format
ppi	Pixel per Inch
PPML	Personalized Print Markup Language
PS	PostScript
RBA	Rules-Based Automation
RDF	Resource Description Framework
RFQ	Request for Quote
RIP	Raster Image Processor
SOAP	Simple Object Access Protocol
TIFF	Tag(ged) Image File Format
UCS	Universal Multiple-Octet Coded Character Set
URI	Uniform (Universal) Resource Identifier
URL	Uniform Resource Locator
UTF	UCS transformation format
VDP	Variable Data Printing
W3C	World Wide Web Consortium
WfMC	Workflow Management Coalition
WMS	Workflow Management System
XSD	XML Schema Definition
XML	Extensible Markup Language
XMP	Extensible Metadata Platform
XPS	XML Paper Specification
XSLT	Extensible Stylesheet Language Transformation

Index

About the Authors

Professor Dr. Thomas Hoffmann-Walbeck: Mathematician, Software developer, project manager. Professor since 1998 at the University of the Media in the Printing and Media Technology degree program. Teaching areas: prepress, applied computer sciences, JDF networking.

Sebastian Riegel: Engineering Graduate. Graduate degree at the University of the Media in Stuttgart. Research assistant since 2003 in the Printing and Media Technology degree program, technically responsible for the CtP laboratory and for the JDF integration of the university.

About Printing Industries of America

Printing Industries of America, along with its affiliates, delivers products and services that enhance the growth, efficiency, and profitability of its members and the industry through advocacy, education, research, and technical information.

Printing Industries of America developed from the 1999 merger of the Graphic Arts Technical Foundation (GATF), founded in 1924, and Printing Industries of America (PIA), founded in 1887. This consolidation brought together two powerful partners: the world's largest graphic arts trade association representing an industry with more than 1 million employees and $156 billion in sales and a nonprofit, technical, scientific, and educational organization dedicated to the advancement of the graphic communications industries worldwide.

Printing Industries of America's staff of researchers, educators, and technical specialists helps members in more than 80 countries maintain their competitive edge by increasing productivity, print quality, process control, and environmental compliance and by implementing new techniques and technologies.

In addition to striving to advance a global graphic communications community through conferences, Internet symposia, workshops, consulting, technical support, laboratory services, and publications, Printing Industries of America promotes programs, services, and an environment that helps its members operate profitably.

Many of Printing Industries' members are commercial printers, allied graphic arts firms such as electronic imaging companies, equipment manufacturers, and suppliers. Its special industry groups, sections, and councils were developed to serve the unique needs of specific segments of the print and graphic communications industries and provide members with current information on their specific segment, helping them to meet the business challenges of a constantly changing environment. These groups focus on web offset printing, label printing, binding, financial executives, sales and marketing executives, and digital printing.

Printing Industries Press publishes books on nearly every aspect of the field; training curricula; audiovisuals and digital media; and research and technology reports. It also publishes *Printing Industries of America: The Magazine,* providing articles on industry technologies, trends, business management practices, economics, benchmarks, forecasts, legislative and regulatory affairs, human and industrial relations issues, sales, marketing, customer service techniques, and management resources. The magazine represents the consolidation of *GATFWorld* and *Management Portfolio,* formerly bi-monthly publications of the association.

For more information about Printing Industries of America, special industry groups, sections, products, and services, visit www.printing.org.

Printing Industries of America Affiliates

Canadian Printing Industries Association
Ottawa, Ontario
www.cpia-aci.ca

Graphic Arts Association
Trevose, PA
www.gaa1900.com

Pacific Printing and Imaging Association
Portland, OR
www.ppiassociation.org

Printing & Graphics Association MidAtlantic
Columbia, MD
www.pgama.com

Printing & Imaging /Association of MidAmerica
Dallas, TX
www.piamidam.org

Printing & Imaging Association of Georgia
Smyrna, GA
www.piag.org

Printing Association of Florida
Orlando, FL
www.pafgraf.org

Printing Industries Alliance
Amherst, NY
www.pialliance.org

Printing Industries of Arizona/New Mexico
Phoenix, AZ
www.piaz.org

Printing Industries Association of San Diego
San Diego, CA
www.piasd.org

Printing Industries Association Inc. of Southern California
Los Angeles, CA
www.piasc.org

Printing Industries of Ohio • N. Kentucky
Westerville, OH
www.pianko.org

Printing Industries of the Gulf Coast
Houston, TX
www.pigc.com

Printing Industries of Michigan, Inc.
Southfield, MI
www.print.org

PINE
Southborough, MA
www.pine.org

Visual Media Alliance
San Francisco, CA
www.visualmediaalliance.org

Printing Industries of St. Louis, Inc.
Maryland Heights, MO
www.pistl.org

Printing Industries of Utah
West Jordan, UT
www.piofutah.com

Printing Industries of Virginia
Ashland, VA
www.piva.com

Printing Industries of Wisconsin
Pewaukee, WI
www.piw.org

Printing Industry of Illinois/Indiana Association
Chicago, IL
www.pii.org

Printing Industry Midwest
Roseville, MN
www.pimn.org

The Printing Industry of the Carolinas, Inc.
Charlotte, NC
www.picanet.org

Printing Industry Association of the South
Nashville, TN
www.pias.org

Selected Titles from Printing Industries Press