JDF Specification

Page Intentionally Left Blank.

JDF Preface and User Overview

This specification is immense … there little doubt about that … but it is also a keystone standard for the future of graphic communications. The members of CIP4 believe that users and developers alike should have a clear understanding of what the objectives of the Job Definition Format (JDF) are as well as an understanding of its value and purpose. To that end we thought you would find a "non-standard" preface and user overview helpful.

Before we get into the overview, we remind you that JDF is a living specification. We would value your comments and input. There are several ways to contact the International Cooperation for the Integration of Processes in Prepress, Press and Postpress (CIP4) association and to receive ongoing information about CIP4 activities. To get a list of contacts, join the JDF developers form, or sign up for email updates, visit the contact page at http://www.cip4.org/. (*Of course, we'd love to have you as a CIP4 member too! Be sure to review the membership page when you visit the CIP4 Website.)*

You will also find callouts throughout this document that are identified by three different icons. These callouts, provided for your convenience, are not normative parts of the standard (i.e., they're not technically a part of the *standard*). They provide references to external sources, executive summaries of complex technical concepts, and some thoughts or strategies you may want to consider as you formulate your JDF implementation plan. Look for these callout icons:

Icon	Callout Type
	External references to online resources, related standards, tutorials, and helpful information.
	Executive-style summaries of technical concepts in easy to understand language.
	Thoughts to ponder and strategy ideas for formulating JDF implementation programs.

Value. This revision of JDF is significant because it builds upon the first version of JDF (v.1.0) to deliver a fully functional and mature standard. As such, this revision includes elements from which executives, shop managers, and technicians will all benefit equally, though in different ways. In the next few years it is our belief that this specification will positively effect everyone involved in the creation and production of printing; regardless of form (offset, digital, flexographic, and so on) or function (direct mail, periodical publication, packaging, and so on). Furthermore, JDF will be of value to companies both large and small. Some of the benefits that JDF may provide include:

- A common language for describing a print job across enterprises, departments, and software and systems;
- A tool for verifying the accuracy and completeness of job tools;
- A systems interface language that can be used to benchmark the performance of new equipment (hardware and software) and that can reduce the cost of expensive custom integration for printers, prepress services, and others;

- A basis for total workflow automation that incorporates all aspects of production: human, machine, and computer;
- A standard that can be applied to eliminate wasteful rekeying and redundancy of information; and
- A common computer language for printing and related industries as well as a platform for more effective communication.

Most importantly, JDF provides an opportunity for users of graphic arts equipment to get a better return on their technology investment and an opportunity to create a print production and distribution workflow that is more competitive with broadcast media in terms of time-to-market.

> **Implementation Strategy**
>
> As you read this standard, consider how to make JDF a part of your equipment evaluation and purchasing procedures. Should you add JDF enabled systems slowly with equipment replacement and upgrades, or aggressively as part of a plant reengineering process? What's your desired competitive position?

XML and Schema: Why? The Extensible Markup Language (XML) is the standard language that is employed by JDF. JDF is also constructed to the World Wide Web Consortium's (W3C) recommendation for the construction of schema. Why is this important and, in layman's terms, what does it do for you?

First of all, it is helpful to understand how MIS professionals around the world use XML today. Although there are some systems that manage and process XML directly, it is primarily used as an exchange language or "middleware" element to create the "glue" that ties integrated systems together.

For instance, complex systems such as enterprise resource planning (ERP), data warehousing, or E-commerce systems often tap into numerous legacy databases and application environments. A manager may wish to have a "view" of corporate information that is actually an aggregate of information that may come from various sources such as billing and invoicing, sales management, inventory, and other systems. Rather than merge these systems into a single, monstrous and centralized system, an operator queries the legacy systems and the results are wrapped in XML. This allows programmers to deal with one exchange language or data format instead of a multitude of proprietary data formats.
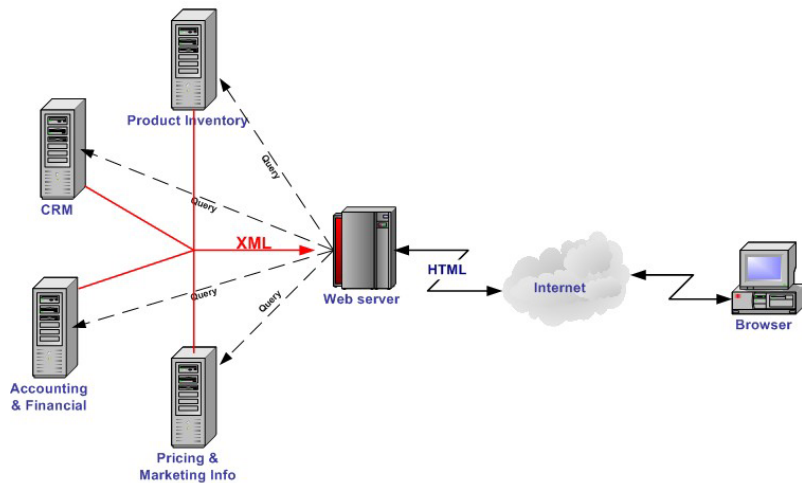
XML is not a *functional* computer language like JAVA, C++ or FORTRAN — it is incapable of manipulating data in anyway; rather, it is a *descriptive* computer language that can be used to describe your information including its structure, interrelationships, and to some extent, its intended usage. For this reason, modern program languages such as JAVA provide intrinsic support for XML processing. Most modern database applications also provide methods for receiving and delivering XML.

Early XML, based solely upon the XML 1.0 specification, had a few limitations that prevented it from being used widely as a transactional data format *across* enterprises, as opposed to *within* enterprises (where it found its niche as described above.) For example, there is probably a database behind each of your major systems and applications. If your database has reserved a fixed space a data particular field and a supplier provides a transaction with a data element larger than that field, you have a problem. The data limitations of XML 1.0 cannot effectively deal with this. The XML Schema specification solved this problem and others.

X M L
S c h e m a

To learn more about XML Schema, including tools, usage, tutorials, and other resources visit http://www.w3.org/XML/Schema

The Pluses of Parsing. Schemas also provide one other feature that is perhaps the greatest benefit. Tagged documents or transactions (called "instances" in XML parlance) are *parsible*. Schemas, such as JDF, establish rules for structuring your information. A parser is a software application that reads those rules, checks documents and transactions, and then validates that they conform to the rules as established in your schema … sort of like preflighting but for XML instances rather than your layout pages.

Parsers can play many roles. Like preflighting software, parsers can be run as standalone applications, but they can also be found embedded into other applications. Some of the roles parsers may play in your JDF-enabled workflow include:

F r e e
P a r s e r s

The JDF schema was validated with the Xerces parser. This parser, as well as other XML tools, is available for free from The Apache Software Foundation open source software community at http://xml.apache.org/

1. Acceptance checking of client job tickets.
2. Validation of JDF prior to or following transformation of data into and out of databases.
3. Ensuring that source job information is collected as a document is created. (Embedded in document layout software.)
4. Determining if equipment reads and writes Job Messaging Format (JMF) commands, a subset of JDF, as part of equipment benchmarking and testing software.
5. Controlling the movement of workflow information and controls within workflow software, from process to process and as a specific JDF job ticket requires.
6. Working as a middleware component to communicate between JDF-enabled software and systems and your legacy Management Information System (MIS) and corporate applications environments.

It is worth mentioning that parsing can be time consuming and computer intensive. But parsers don't have to be the gatekeepers everywhere in a JDF-enabled workflow. Equipment that is JDF-enabled and part of a company's internal production operations need not parse every communication. It can be limited to equipment evaluation and problem solving applications. The role of JDF parser-enabled software in a printing plant that uses tightly coupled JDF-enabled print production equipment might look like this:
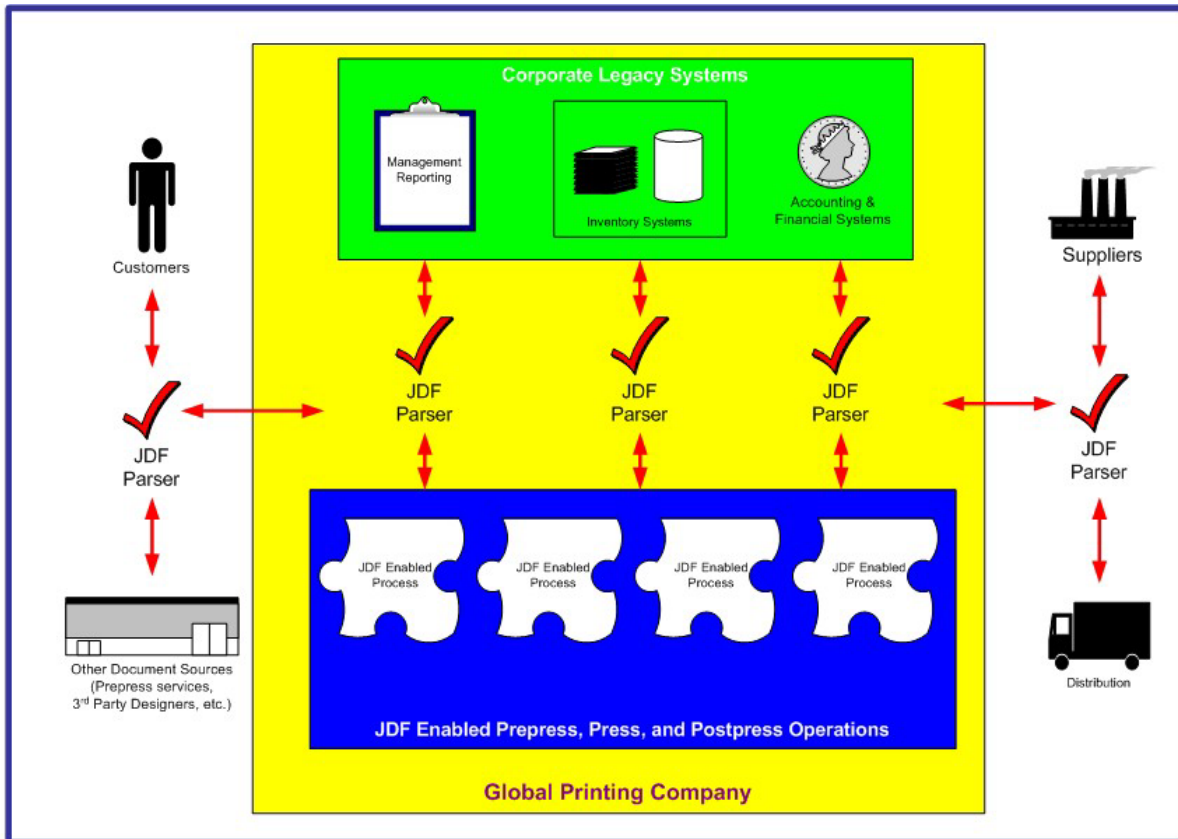
The JDF Concept. The JDF schema is quite complex and detailed — something best left to programmers, MIS personnel, and XML experts. But the language and concepts behind JDF are quite simple and straightforward. The schema itself can be downloaded from the CIP4 Website, but is not part of this specification. Instead, this is your "cookbook." It provides an explanation of each of the components of JDF, its meaning, and intended usage. You will want to use the components of JDF that fit best with your workflow and the needs of your customers. To start, a basic understanding of the concepts behind JDF is in order. There are three primary components to JDF:

1.   JDF itself,
2.   The Job Messaging Format (JMF), and
3.   The MIS system.

JDF is simply an exchange format for instructions and job parameters. You can use PDF, or its standard variant (PDF/X), to relay production files from one platform to another. You can do the same with JDF to relay job parameters and instructions. JDF can be used to describe a printing job logically, as you would in exchanging a job description with a client within an estimate. It can also be used to describe a job in terms of individual production processes and the materials or other process inputs required to complete a job.

There is no such thing as a standard print workflow. In fact, printing is the ultimate form of *flexible manufacturing*. This makes process automation quite a challenge for our industry. What you'll find in this standard are XML element definitions that describe all the production processes and material types you're likely to encounter, regardless of your workflow. These are the building blocks that you can use to emulate your workflow with JDF. As a matter of convention, processes such as preflighting, scanning, printing, cutting, and so on are referred to as process

nodes. Every process in the print production workflow requires input *resources* starting with the client's files or artwork and ending with the final bound, packaged, and labeled print product. For example, before you can print, you need paper, ink, and plates, and before you can send a document to a bindery line, you need printed and cut signatures.

Process *nodes* and *resources* are the basic elements within JDF. They can be strung together to meet the requirements of each job. The output of one process becomes the input of the following process, and a process doesn't begin until its input resources are available:

Node 1 — output — Resource — input — Node 2

Example:
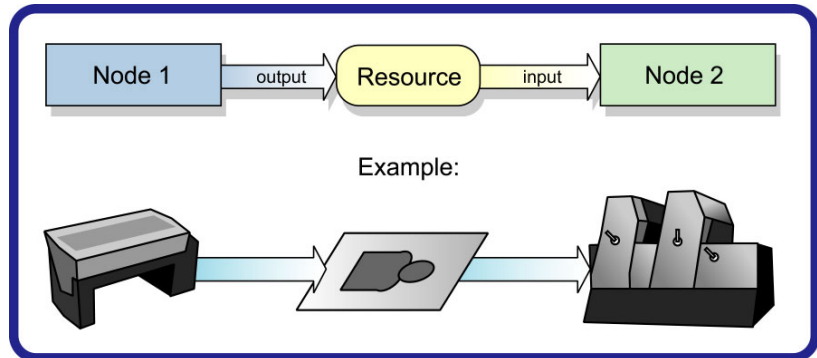
This specification provides details on how to use these building blocks to describe concurrent processes, spawned processes, dynamic processes, and so on. To realize the capabilities of JDF, there are two other things you will need: a way of controlling the flow of process and a way of communicating commands to equipment on the shop floor.

JMF is a subset of JDF that handles communication with equipment on the shop floor. This may include major equipment, such as platesetters, or subsystems, such as in-line color measurement devices. JMF can be used to establish a queue, discover the capabilities of a JDF-enabled device, determine the status of a device (e.g., "RIP'ing," "Idle"), and so on.

Although, theoretically, you can string together equipment that supports JMF directly to one another, in almost all cases you will want your production equipment to communicate with your MIS system. This way it is the MIS system that controls the scheduling, execution, and control of work in progress. The role of the MIS system is described within this standard, but it isn't highly defined. In fact, the JDF standard does not dictate how a JDF system should be built. Many printers, prepress services, and other graphic arts shops will already have MIS systems in place. JDF enabled workflow and MIS systems, custom-tailored to print production requirements, will soon be available on the market. However, many printers already have MIS and workflow systems that have been customized or developed for their own environments. In most cases these legacy systems can be modified to work with the new JDF workflows and JDF enabled equipment. There are a variety of XML support tools available on the market to address the databases underlying all MIS systems.

JMF

The Job Messaging Format (JMF) functions as a standard interface between your equipment and your information systems, or other equipment already on the shop floor. By buying only equipment that supports JMF you will reduce the cost and complexity of integrating new equipment into your production operations, and you will improve the flexibility and adaptability of your shop.

X M L & D a t a b a s e s

To learn more about how XML and database work together, check out the white papers and tutorials available from XML.org at http://www.xml.org/xml/resources_focus_rdbms.shtml.

Table of Contents

Table of Figures

Chapter 1 Introduction

This document defines the technical specification for the Job Definition Format (JDF) and its counterpart, the Job Messaging Format (JMF). We will describe the components of JDF, both internal and external, and explain how to integrate the format components to create a viable workflow. Ancillary aspects are also introduced, such as how to convert PJTF or PPF to JDF, and how JDF relates to IfraTrack. It is intended for use by programmers and systems integrators for operations addressed by the International Cooperation for Integration of Processes in Prepress, Press and Postpress (CIP4). In this first chapter, we present the concept of JDF, how to use this document and some basic document navigational aids.

1.1 Background on JDF

JDF is an extensible, XML-based format built upon the existing technologies of CIP3's Print Production Format (PPF) and Adobe's Portable Job Ticket Format (PJTF). It provides three primary benefits to the printing industry: 1.) the ability to unify the prepress, press, and postpress aspects of any printing job, unlike any previous format; 2.) the means to bridge the communication gap between production services and Management Information Systems (MIS); and 3.) the ability to carry out both of these functions no matter what system architecture is already in place, and no matter what tools are being used to complete the job. In short, JDF is extremely versatile and comprehensive.

JDF is an interchange data format to be used by a system of administrative and implementation-oriented components, which together produce printed products. It provides the means to describe print jobs in terms of the products eventually to be created, as well as in terms of the processes needed to create those products. The format provides a mechanism to explicitly specify the controls needed by each process, which may be specific to the devices that will execute the processes.

JDF works in tandem with a counterpart format known as the Job Messaging Format, or JMF. JMF provides the means for production components of a JDF workflow to communicate with system controllers and administrative components. It relays information about the progress of JDF jobs and gives MIS the active ability to query devices about the status of processes being executed or getting ready to be executed. JMF will provide the complete job tracking functionality that is defined by IfraTrack messaging standard. Depending on the system architecture, JMF may also provide the means to control certain aspects of these processes directly.

JDF and JMF are maintained and developed by CIP4 (http://www.cip4.org). They were originally developed by four companies prominent in the graphic arts industry—Adobe, Agfa, Heidelberg, and MAN Roland, with significant contributions provided by CIP3, the IfraTrack working group, Fraunhofer IGD and the PrintTalk consortium.

1.2 Document References

This specification assumes that the reader has a basic awareness of, or access to, the following documents:

Portable Job Ticket Format
Version 1.1
Date: 2-April-1999
Produced by Adobe Systems Inc.
Available at: http://partners.adobe.com/asn/developer/PDFS/TN/5620.pdf

Print Production Format
Version 3.0
Date: 2-June-1998
Produced by the International Cooperation for Integration of Prepress, Press, and Postpress
Available at: http://www.cip4.org/documents/technical_info/cip3v3_0.pdf

XML Specification
Version 1.0
Date: 10-February-1998
Produced by: World Wide Web Consortium (W3C)
Available at: http://www.w3.org/TR/REC-xml

XML Schema Part 0+1+2: Primer, Structures and Datatypes
Version (W3C Recommendation of 02 May 2001)

Date: 02-May-2001
Produced by: World Wide Web Consortium (W3C) XML Schema working group
Available at: http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/ and
     http://www.w3.org/TR/xmlschema-2/

IfraTrack Specification
Version 2.0
Date: June-1998
IFRA Special Report 6.21.2
Produced by IFRA
Available at: http://www.ifra.com/

Spec ICC.1:1998-09
File Format for Color Profiles
Version 3.5
Date: 1998
Produced by: International Color Consortium
Available at: http://www.color.org/ICC-1_1998-09.PDF

PrintTalk Implementation
Version 1.0
Produced by:PrintTalk Consortium
Available at: http://www.printtalk.org/

1.3 Conventions Used in This Specification

This section contains conventions and notations used within this document.

1.3.1 Text Styles

The following text styles are used to identify the components of a JDF job:

- Elements are written in sans serif. Examples are: Comment, CustomerInfo, and ResourceLinks.
- Attributes are written in italic sans serif. Examples are: *Status*, *ResourceID*, and *ID*.
- Resources are written in bold sans serif. Examples are **ImpositionProof**, **Toner**, and **ExposedMedia.**
- Processes are written in bold-italic sans serif. Examples are ***ColorSpaceConversion***, ***Rendering***, and ***Scanning***.
- Enumerative and boolean values of attributes are written in italics. Examples are: *true*, W*aiting*, *Completed*, and *Stopped*.
- Standard bold text is used for the following purposes:
  - to highlight glossary items. Examples are **device**, **element**, and **job**.
  - to highlight defined items inside a table. An example is the data type **NMTOKEN** in the table in Section 1.4 Data Structures.
  - to highlight definitions of local terms. These are terms that are of local importance for a certain chapter, or some sections inside a chapter. An example is a **spawned job** in Section 4.4 Spawning and Merging.
  - to designate PPF objects in Appendix D, Converting PPF to JDF. Examples are **CIP3ProductName** and **CIP3ProductComponent**.
- For the benefit of those who are reading this document in PDF or online, cross-reference

Extended Backus-Naur Form

The Extended Backus-Naur Form (EBNF) provides a compact notation that is commonly used in the specifications of programming languages. The official EBNF standard, ISO/IEC 14977:1996(E), is not freely available online. To order a paper copy from ISO, contact:

International Organization for Standardization
Case postale 56
1, rue de Varembé
CH-1211 Genève 20 Switzerland
Phone: +41 22 749 01 11
Fax: +41 22 733 34 30
Email: sales@isocs.iso.ch

links are denoted by gray text. Examples are Chapter 6 Processes, and Section 1.2 Conventions Used in This Specification. To follow a link, click the highlighted text. The examples provided are not actual links.

- Also for the benefit of online readers, external hyperlinks are graphically designated. An example is http://URL.com. To follow a link, click the highlighted text. The example provided is not an actual link.

1.3.2 Specification of Cardinality

The cardinality of JDF Data Types is expressed using a simple Extended Backus-Naur Form (EBNF) notation. The symbols in this notation may be combined to indicate both simple and complex patterns, as demonstrated in the following table. A and B represent simple expressions.

Notation	Description
(expression)	Expression is treated as a unit and may be combined as described in this list.
A	Matches A. A must occur exactly one time.
A ?	Matches A or nothing. A is optional, or is required only in the circumstances explained in the description field.
A +	Matches one or more occurrences of A.
A *	Matches zero or more occurrences of A.

1.4 Glossary of Terminology

The following terms are defined as they are used throughout this specification. For more detail on job and workflow components, see Section 2.1 System Components.

Term	Definition
Agent	The component of a JDF-based workflow that writes JDF.
Attribute	An XML-based syntactic construct describing an unstructured characteristic of a JDF **node** or **element**.
Big job	The combined **job** that independent jobs are merged into in the case of independent spawning and merging.
Class	A set of complex data types with common content in an object-oriented sense. A complex data type may consist of **elements** and **attributes**.
Controller	The component of a JDF-based workflow that initiates **devices**, routes JDF, and communicates status information.
Default	Used to indicate the attribute value that a JDF Consumer must use if an Agent omits an Optional attribute (as indicated by a "?" in this spec) from a JDF instance. See Section 1.4.2.1 Conformance Requirements for Support of Attributes and Attribute Values.
Deprecated	Indicates that a JDF element is being phased out of JDF usually in favor of newer JDF element(s). It is recommended that an Agent not include such a JDF element in a JDF instance. Such an indicated JDF element may be removed from a future version of the JDF specification. JDF Consumers should only support such JDF elements for backward compatibility with previous versions of JDF. Deprecated items are flagged with Deprecated in JDF 1.1 in this specification.
Device	The component of a JDF workflow part that interprets JDF and executes the instructions. Devices control **machines** in a proprietary manner.
Document set	A set of instance documents presumed to be related.
Element	An XML-based syntactic construct describing structured data in JDF.

Term	Definition
Finished page	A *finished page* is a page of a final product with no fold inside. The folds of the finished product for packaging, e.g., folding letters into an envelope, have no effect on the finished page definition. A sheet of paper with no fold inside consists of two *finished pages* (front and back side). If there are folds seen in a sheet in the final product, the number of *finished pages* of one sheet is given by $2*(X+1)*(Y+1)$, where X denotes the number of folds in X direction and Y denotes the number of folds in Y direction, each seen in the completely opened sheet. Examples: One sheet in a book has two *finished pages*, one front, one back; a brochure with one fold inside has four *finished pages*.
Instance document	A document that is part of the output of a job. This generally refers to personalized printing jobs. Each of the individual documents produced from the same input template is referred to as an instance document. For example, in a credit card statement run, each statement is an instance document.
JDF consumer	A Device, Controller, Process, Queue or Agent that consumes JDF instances
JMF	Job Messaging Format. A communication format with multi-level capabilities. Structures information between **MIS** and **controllers**.
Job	A hierarchical tree structure comprised of **nodes**. Describes the output that is desired by a customer.
Job part	One or more **nodes** which comprise the smallest level of control of interest to MIS.
Link	A pointer to information that is located elsewhere in a JDF document or that is located in another document.
Machine	The part of a device that does not know JDF and is controlled by a JDF device in a proprietary manner.
MIS	Management Information Systems. The functional part of a JDF workflow that oversees all processes and communication between system components and system control.
Node	The JDF **element** type detailing the resources and process specification required to produce a final or intermediate product or resource.
Partitioned resource	Structured resource that represents multiple physical or logical entities, such as separated plates.
PDL	Page Description Language. A generic term for any language that describes pages, which may be printed. Examples are PDF®, PostScript® or PCL®.
Process	An individual step in the workflow.
Queue	Entity that accepts job entries via a **JMF** messaging system.
Reader page	A *reader page* is a logical page as perceived by a reader, for example one **RunList** entry. One *reader page* may span more than one *finished page,* e.g,. a centerfold. One *finished page* may contain contents defined by multiple *reader pages,* e.g., NUp imposition. *Reader pages* are defined independent of *finished pages*.
Resource	A physical or conceptual entity that is modified or used by a **node**. Examples include paper, images, or process parameters.
Small job	An independent job that is merged into a **big job**.
Support	A JDF Consumer **supports** a JDF syntactic construct (processes, resources, elements, attributes, and attribute values) if the JDF Consumer performs the action defined in this specification for the JDF construct when consuming a JDF instance that includes the JDF syntactic construct. If the Machine that a Device is representing supports a feature which is represented by a JDF construct, then the Device **should** support that JDF syntactic construct.
Tag	A syntactic construct that marks the start or end of an **element**.
Work center	An organizational unit, such as a department or a subcontracting company, that can accomplish a task.

1.4.1 Conformance Terminology

The words "**must**", "**must not**", "**required**", "**should**", "**should not**", "**recommended**", "**may**", and "**optional**" are used in this specification to define a requirement for the indicated **Agent** or the indicated **JDF Consumer** as follows:

Table 1-1 Conformance Terminology

Term	Meaning
Must, Required	Mean that the definition is an absolute requirement of the specification.
Must not	Means that the definition is an absolute prohibition of the specification.
Should, Recommended	Mean that there may exist valid reasons in particular circumstances for an implementer to ignore a particular item, but the implementer must fully understand the implications and carefully weigh the alternatives before choosing a different course.
Should not, Not recommended	Mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the implementer should fully understand the implications and then carefully weigh the alternatives before implementing any behavior described with this label.
May, Optional	Mean that an item is truly optional. Unless specified otherwise, the word "optional" refers to JDF syntax, i.e., what an Agent **may** include in a JDF instance, and does not refer to a JDF Consumer option, i.e., not to what a JDF Consumer **may** support. If a JDF Consumer is using a JDF parser, that parser will supply the default values indicated in this specification, if any, for optional attributes that the Agent has omitted (indicated by "?" in this specification.) See Section 1.3.2 Specification of Cardinality. For features that are optional for a JDF Consumer to support, one vendor may choose to support such an item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit support of that item. Similarly, one vendor of an Agent may choose to supply such an item in a JDF instance, while another vendor may omit the same item in a JDF instance. A JDF Consumer implementation which does not include support of a particular option must be prepared to interoperate with an Agent implementation which does supply the option, though with reduced functionality. In the same vein, a JDF Consumer implementation which does include support for a particular option must be prepared to interoperate with an Agent implementation which does not supply the option in the JDF instance.

Note: There is no corresponding "may not" or "need not" term for something that an implementation may optionally omit or optionally not perform. The term "may not" sounds more like a prohibition. Also, it is better form to put the requirement into a positive statement. For example, instead of saying that an Agent need not include an attribute that this specification indicates with a "?" character, it is better to say that a JDF produce may omit an attribute in a JDF instance that this specification indicates with a "?" character.

1.4.2 Conformance Requirements for JDF Entities

The subsections of this section define the general conformance requirements for the JDF entities: 1.) attributes and attribute values, 2.) resources, 3.) processes, and 4.) combined processes.

1.4.2.1 Conformance Requirements for Support of Attributes and Attribute Values

If a JDF Consumer supports an attribute, it must support all of the values that this specification indicates are required for a JDF Consumer to support (whether or not the attribute is required for the Agent to supply in that context). If this specification is silent on which values are required for support of an attribute, then the JDF Consumer must support at least one value in order to claim support for the attribute.

Attributes that are optional for an Agent to include in a JDF instance are indicated by a "?" character following the attribute name as indicated in Section 1.3.2 Specification of Cardinality. In the description of most optional attributes there is a "Default = ..." statement that indicates the default value that a JDF Consumer must use if the Agent omits the optional attribute from a supplied resource in a JDF instance. Such an indicated default value

must have the same semantic meaning as if an Agent includes the attribute in the JDF instance with the same value. If the indicated default value is the special *SystemSpecified* value or is indicated as "system specified", then the JDF Consumer must provide an actual value that depends on the implementation of the JDF Consumer and which may be configurable by a system administrator. If an optional attribute does not have a default value indicated in its description and the JDF instance does not include the attribute, then the JDF Consumer must supply a system-specified value.

1.4.2.2   Conformance Requirements for Support of Resources

If a JDF Consumer supports a resource, it:

1. must support all of the attributes (see Section 1.4.2.1) defined for that resource that an Agent is required to include in the resource instance (attributes with either no marks or a "+"), and – see section 1.3.2), and
2. must support the JDF:*SettingsPolicy* (see section 3.1.2)*,* JDFResource:*SettingsPolicy* (see section 3.7)*,* JDF: *BestEffortExceptions,* JDF:*MustHonorExceptions,* and JDF:*OperatorInterventionExceptions* (see section 3.1.1) attributes and all of their defined values. These attributes control the policy that a JDF Consumer must follow when it encounters unsupported settings, i.e., subelements, attributes or attribute values in the resource.

1.4.2.3   Conformance Requirements for Support of Processes

All processes are optional for a JDF Consumer to support. However, a Device must support at least one process or a combined process. If a JDF Consumer supports a process, it:

1.   must support all of the input and output resources as described in Section 1.4.2.2 that this specification defines for that process and

2. may make its own assumptions regarding attributes and subelements of an optional input resource (resources with either a "?" or an "*" – see section 1.3.2) that an Agent has omitted from the process in the JDF instance. Therefore, default attribute values defined in this specification are not guaranteed when the Agent omits the resource from the process in the JDF instance (see section 6.1 Process Template).

3. must find the processes that it supports in a JDF instance and must ignore all other processes, independent of the *SettingsPolicy* attribute for those other processes.

1.4.2.4   Conformance Requirements for Support of Combined Processes

All combined processes are optional for a JDF Consumer to support. If a JDF Consumer supports a combined process, it:

1. must support all of the input resources as defined in Section 1.4.2.2 that this specification defines for the *first* process in the combined process node, i.e., the first process listed in the *Types* attribute, and

2. must support all of the output resources as defined in Section 1.4.2.2 that this specification defines for the *last* process in the combined process.

3. may support resources that are used as exchange resources between processes in the process chain of the combined process, i.e., resources that are both produced and consumed within the combined node.

4. must support resources in intermediate process steps that are *not* used as exchange resources between processes in the process chain of the combined process.

5. may make its own assumptions regarding attributes and subelements of an optional input resource that an Agent has omitted from the combined process in the JDF instance. Therefore, default attribute values defined in this specification are not guaranteed when the Agent omits the resource from the combined process in the JDF instance (see section 6.1 Process Template).

6. must search a JDF instance and find the combined process nodes that exactly match what it supports, i.e., that match the value list of the *Types* attribute*,* and must ignore all other process nodes, independent of the *SettingsPolicy* attribute for those other processes.

1.5 Data Structures

The following table describes the data structures as they are used in this specification. For more details on JDF Schema and Datatypes, see Appendix A **Encoding**.

Table 1-2 JDF data types

Data Type	Description
boolean	Binary-valued logic: (true | false).
CMYKColor	Represents a CMYK color specification.
date	Represents a time period that starts at midnight of a specified day and lasts for 24 hours.
dateTime	Represents a specific instant of time. It must be a UTC-time or a local time that includes the time zone.
double	Corresponds to IEEE double-precision, 64-bit floating point type
duration	Represents a duration of time.
DurationRange	*DurationRange* is used to describe a range of time durations. More specifically, it describes a time span that has a relative start and end.
element	Structured data. The specific data type is defined by the element name.
enumeration	Limited set of **NMTOKEN** (see below).
enumerations	Whitespace-separated list of enumeration data types.
gYearMonth	Represents a specific gregorian month in a specific gregorian year.
hexBinary	Represents arbitrary hex encoded binary data.
ID	Unique identifier as defined by [XML Specification 1.0] (see Section 1.2 Document References). Must be unique within the scope of the JDF-document.
IDREF	Reference to an element holding the unique identifier as defined by [XML Specification 1.0].
IDREFS	List of references (IDREFs) separated by white spaces as defined by [XML Specification 1.0].
integer	Represents numerical integer values.
IntegerList	Whitespace-separated list of **integer**s.
IntegerRange	Two integers separated by a "~" character that define a closed interval .
IntegerRangeList	Whitespace-separated list of **integers** and **IntegerRange**s.
LabColor	Represents a Lab color specification.
language	Represents a language and country code (for example, en-US) for a natural language.
matrix	Whitespace-separated list of 6 **number**s representing a coordinate transformation matrix.
NamedColor	Represents a color definition by name. A list of valid NamedColor values is provided in Appendix **A.2.8**.
NameRange	Two **NMTOKEN** separated by a "~" character that define an interval of NMTOKEN.
NameRangeList	Whitespace-separated list of **NMTOKEN** and **NameRange**s.
NMTOKEN	A continuous sequence of special characters as defined by the [XML Specification 1.0].
NMTOKENS	Whitespace-separated list of **NMTOKEN**.
number	double or integer
NumberList	Whitespace separated list of **number**s.
NumberRange	Two numbers separated by a "~" (tilde) character that defines the closed interval of the two.
NumberRangeList	Whitespace-separated list of **NumberRange**s
path	Whitespace-separated list of path operators as defined in PDF.
rectangle	Whitespace-separated list of 4 **numbers** representing a rectangle.
refelement	**element** or a reference to an element. Used to define candidates for inter-resource linking in resources.

Data Type	Description
shape	Whitespace-separated list of 3 **numbers** representing a 3-dimensional shape consisting of a width, height, and length. Unless specified otherwise in the attribute Description, these three numbers are an X-dimension, a Y-dimension, and a Z-dimension, respectively.
ShapeRange	Two **Shape**s separated by a "~" (tilde) character that defines a 3-dimensional box bounded by x1 y1 z1~ x2 y2 z2.
ShapeRangeList	Whitespace-separated list of **shapes** or **ShapeRange**s.
sRGBColor	Represents an sRGB color specification.
string	Character strings without line feed.
telem	Text elements that contain larger chunks of character data and may include line feeds.
text	Text data contained in a telem (text **element**).
TimeRange	Two dateTimes separated by a "~" (tilde) character that defines the closed interval of the two. TimeRange corresponds semantically to the time interval (two time instants separated by a slash) defined in ISO 8601.
TransferFunction	Whitespace separated list of an even number of **numbers** representing a set of XY coordinates of a transfer function.
URI	URI-reference. Represents a Uniform Resource Identifier (URI) Reference as defined in Section 4 of [RFC 2396].
URL	URL-reference. Represents a Uniform Resource Locator (URL) Reference as defined in Section 4 of [RFC 2396].
XYPair	Whitespace-separated list of 2 **numbers**. Unless specified otherwise in the attribute Description, these two numbers are an X-dimension and a Y-dimension, respectively.
XYPairRange	Two **XYPair**s separated by a "~" (tilde) character that defines a rectangle bounded by x1 y1 ~ x2 y2
XYPairRangeList	Whitespace-separated list of **XYPairRange**s.

1.6 Units

JDF specifies most values in default units. That means you can't use alternate units instead of the defined default units. All measurable quantities are stated in double precision. Processors should only specify a *Unit* if no default exists, such as when new resources are defined. Then the units must be based on metric units. Overriding the default units that are defined in this table is non-standard and may lead to undefined behavior. Any exceptions are specified in the appropriate descriptive tables.

The following table lists the units used in JDF. The representation column specifies the XML representation in the *Unit* attribute of resources.

Table 1-3 Units used in JDF

Measurement	Unit	Representation	Remarks
Length	point (1/72 inch)	pt	Used for all except microscopic lengths (see below)
	micron	mu	
Volume	liter	l	-
Weight	gram	g	-
Area	m^2	m2	-
Resolution	dpi or lpi	dpi or lpi	-
Paper weight	g/m^2	g/m2	-
Speed	units/hour	*/h	Replace the "*" in the representation with the appropriate unit

Measurement	Unit	Representation	Remarks
Temperature	C° (Celsius)	C	degree centigrade
Angle	degrees°	degree	-
Countable Objects	1	-	Countable objects, such as sheets, have no unit specification.

Chapter 2 Overview of JDF

Introduction
This chapter explains the basic aspects of JDF. It outlines the terminology that is used and is recognized by the format, and the components of a workflow necessary to execute a printing job using JDF. Also provided is a brief discussion of JDF process structure and the role of messaging in a JDF job.

2.1 System Components
This section defines unique terminology used in this specification for the job and workflow components of JDF. Links to additional information is included for some terms.

2.1.1 Job Components
This terminology describes how JDF is described conceptually and hierarchically.

2.1.1.1 Jobs and Nodes
A job is the entirety of a JDF project. Each job is organized in a tree structure containing all of the information required to complete the intended project. The information is collected logically into what is called a **node**. Each node in the tree structure represents an aspect of the job to be executed.

The nodes in a job are organized in a hierarchical structure that resembles a pyramid. The node at the top of the pyramid describes the overall intention of the job. The intermediate nodes describe increasingly process-oriented aspects of the job, until the nodes at the bottom of the pyramid each describe a single, simple process. Depending on where in the job structure a node resides, it can represent a portion of the product to be created, one or many processing steps, or other job parts. For more information about jobs and nodes, see Chapter 3 Structure of JDF Nodes and Jobs.

2.1.1.2 Elements
An element is an XML syntactic construct. (See also: attributes.) Within this document, the term refers to the structured subparts of a JDF **node**. Technically, JDF nodes are themselves XML elements. However, within this specification, "node" is used to distinguish between the independent JDF aspect and its subparts. Furthermore, elements that are subparts of other elements are often referred to as subelements. There is no structural distinction between nodes, elements and subelements; rather, the different terminology is intended to describe the hierarchical relationships.

JDF elements are represented by two kinds of data types: element and text element. The latter is abbreviated as telem. For more information about elements, see Section 3.1.2 Fundamental JDF Attributes and Elements.

X M L
C r a s h   C o u r s e

Need a crash course in XML? XML101.com provides online tutorials that non-programmers can easily follow. The site includes examples. See http://xml101.com/

2.1.1.3 Attributes
An attribute is an XML syntactic construct. (See also: elements.) Within this document, the term refers to characteristics of **elements,** a subpart of a **node**. For instance, each node has an *ID* attribute that contains a unique identifier. Attributes contain parameters of different data types, such as **string**, **enumeration**, and **dateTime**.

For more information about attributes, see Section 3.1.2 Fundamental JDF Attributes and Elements. Note that an attribute with an empty (zero length) value string is illegal except when the attribute value is defined as an arbitrary string.

2.1.1.4 Relationships
The hierarchical JDF structure implies relationships between **nodes** and **elements** within a JDF tree structure. The terms used in this document to describe these relationships are defined below, and, in some cases, include a brief representation of the encoding that would express them.
- **Parent**: An element that directly contains a child element.
  `<Parent><Child/></Parent>`
- **Child**: An element that resides directly in the parent element.

- **Sibling**: An element that resides in the same parent element as another child element.
  ```
  <Any><Sibling/><Sibling/></Any>
  ```
- **Descendent**: An element that is a child or a child of a child, etc.
- **Ancestor**: An element that is a parent or a parent's parent, etc.
  ```
  <Ancestor>
    <Any>
      <Descendent/>
      <MoreAnys>
        <Descendent/>
      </MoreAnys>
    </Any>
  </Ancestor>
  ```
- **Root**: The single element that contains all other elements as descendents.
- **Leaf**: Node without further children.
- **Branch**: An intermediate node in a hierarchy that contains at least one child node. A branch is never a leaf.

2.1.1.5 Links

There are two kinds of links in JDF: internal links and external links. Internal links are pointers to information that is located elsewhere in a JDF document. The data that is referenced by the link is located in a target **element**. External links are used to reference objects that are outside of the JDF document itself, such as content files or color profiles. These objects are linked using standard URLs (Uniform Resource Locators).

JDF makes extensive use of links in order to reuse information that is relevant in more than one context of the job. The same target may be referenced by multiple links. However, no link references more than one target.

2.1.2 Workflow Component Roles

The four components required to create, modify, route, interpret and execute a JDF job are known as agents, controllers, devices and machines. Overseeing the workflow created by these components is MIS, or Management Information Systems. These five aspects of a JDF workflow are described in the sections that follow.

By defining these terms, this specification does not intend to dictate to manufacturers how a JDF/JMF system should be designed, built, or implemented. The intention is to name the component mechanisms required for the interaction of actual components in a workflow during the course of a JDF job. In practice, it is very likely that individual system components will include a mixture of the capabilities described in the following sections. For example, many controllers are also agents.

2.1.2.1 Machines

A machine is any part of the workflow system designed to execute a **process**. Most often, this term refers to a piece of physical equipment, such as a press or a binder, but it can also refer to the software components used to run a particular machine. Computerized workstations, whether run through automated batch files or whether controlled by a human worker, are also considered machines if they have no JDF interface.

2.1.2.2 Devices

The most basic function of a device is to execute the information specified by an **agent** and routed by a **controller**. Devices must be able to execute JDF **nodes** and initiate **machines** that can perform the physical execution. The communication between machines and devices is not defined in this specification. Devices may, however, support **JMF** messaging in order to interact dynamically with controllers.

2.1.2.3 Agents

Agents in a JDF workflow are responsible for writing JDF. An agent has the ability to create a **job**, to add **nodes** to an existing job, and to modify existing nodes. Agents may be software processes, automated tools, or even text editors. Anything that can be used in composing JDF can be considered an

> **Agents, Controllers & Devices**
>
> "Agents," "Controllers," and "Devices" are special, logical descriptions. You probably won't ever buy one. An agent (writes and reads JDF) may be any software tool that can parse JDF. Controllers communicate instructions that devices act upon. They are functions that may be embedded into your software, production equipment, or MIS systems.

agent.

Actual implementations of **devices** or **controllers** will most often be able to modify JDF. These system components have agent properties in the terms of this specification.

2.1.2.4   Controllers

Agents create and modify JDF information; controllers route it to the appropriate **devices**. The minimum requirement of a controller is that it can initiate **processes** on at least one device, or at least one other slave controller that will then initiate processes on a device. In other words, a controller is not a controller if it has nothing to control. In some cases, a pyramid-like hierarchy of controllers can be built, with controllers at the top of the pyramid controlling a series of lower-level controllers at the bottom. The lowest-level controllers in the pyramid, however, must have device capability. Therefore, controllers must be able to work in collaboration with other controllers. In order to communicate with one another, and to communicate with devices, controllers must support the JDF file-exchange protocol and may support **JMF**. Controllers can also determine process planning and scheduling data, such as process times and planned production amounts.

2.1.2.5   Management Information Systems—MIS

The overseer of the relationships between all of the units in a workflow is known as Management Information Systems, or MIS. MIS is, in effect, a macrocosmic **controller**. It is responsible for dictating and monitoring the execution of all of the diverse aspects of the workflow. To do this, it must remain in contact with the actual production facilities. This can be accomplished either in real time using **JMF** messaging or post-facto using the audit records within JDF.

To allow MIS to communicate effectively with the other workflow components, JDF supplies what is essentially a messenger service, in the form of JMF, to run between MIS and production. This format is equipped with a variety of message types, ranging from simple, unidirectional notification to queries and even commands. System designers have a great deal of flexibility in terms of how they choose to use the messaging architecture, so that they can tailor the processes to the capabilities of the existing workflow mechanism. Figure 2.1 depicts how various communication threads can run between MIS and production.

JDF also provides system components the ability to collect performance data for each **node**, which can then be passed on to a job-tracking system for use by the MIS system. These data may be derived from the messages that the controller receives or from the audit records in the job (for more information on audits, see Section 3.10.1 Audit Elements). Alternatively, the completed job may be passed to the job accounting system, which examines the audit records to determine the costs of all the processes in the job.

> **Automating Data Flows**
>
> A JDF-enabled workflow may require a tremendous amount of information. This could seem daunting to anyone who expects to have to enter information into a system, but it need not be the case. From the style information in a layout file, to automatically generated image file header information, to the color profiles tagged onto images automatically by digital cameras or image editing systems, a great deal of information can be captured and passed along from one JDF-enabled application to another. Furthermore, where, in the specification, there are many options, those options can be set to a default that represents your particular plant or workflow. For instance, JDF provides a variety of staple folds. If your plant only supports a crown fold, that becomes the default in your JDF-enabled system and is never manually specified or keyed.

2.1.2.6   System Interaction

An example of the interaction and hierarchical structure of the components considered in the preceding sections is shown in the following figure. Single arrows indicate uni-directional communication channels and double arrows indicate bi-directional communication.
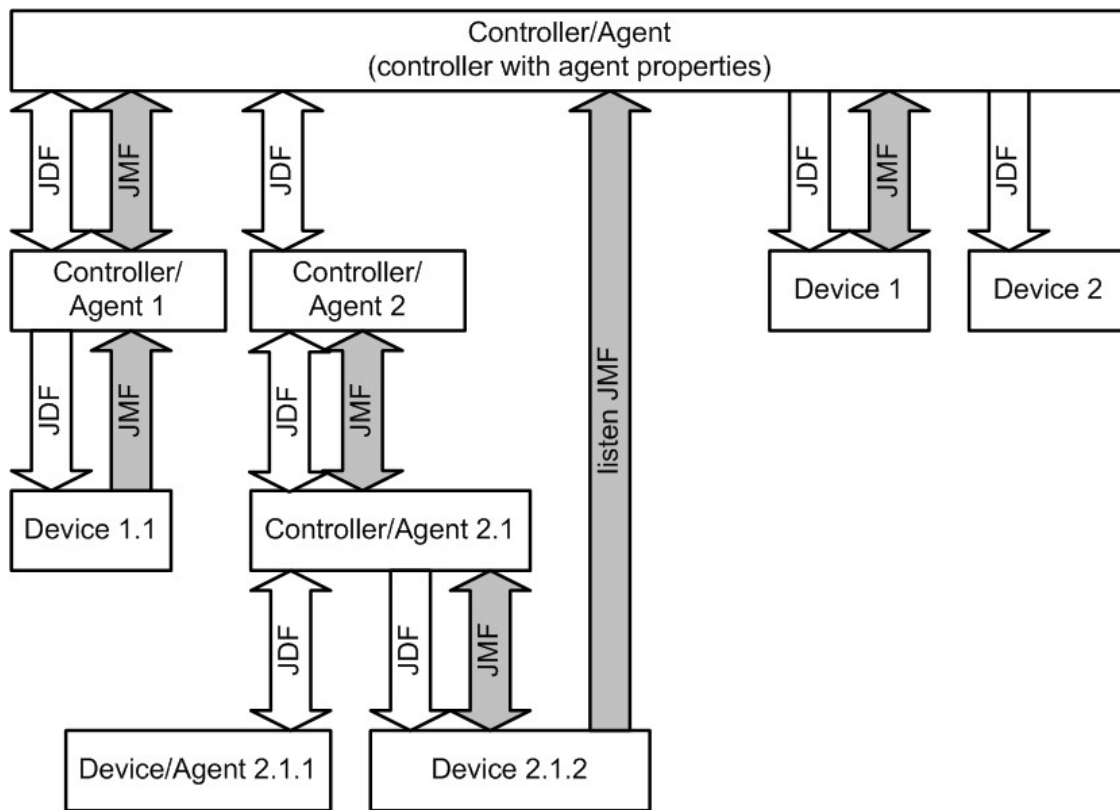
Figure 2.1 Example of JDF and JMF workflow interactions

2.2 JDF Workflow

JDF does not dictate that a workflow be constructed in any prespecified way for it to be usable. On the contrary, its flexibility has allowed JDF to model existing custom solutions for the graphic arts, as well as those yet to be imagined. JDF is equally as effective with a simple system using a single controller-agent and device as it is with a completely automated industrial press workflow with integrated pre- and postpress operations.

Because of workflow system construction in today's industry, the principal subsection procedures of a printing job—prepress, press, and postpress—remain largely disconnected from one another. JDF provides a solution for this lack of unity. With JDF, a print job becomes an interconnected workflow that runs from job submission through trapping, RIP'ing, filmmaking, platemaking, inking, printing, cutting, binding, and sometimes even through shipping. JDF enables an architecture that defines the process necessary to produce each intended result and identifies the elements necessary to complete the processes. All processes are separated into nodes, and the entire job is represented by a tree of these nodes. All of the nodes taken together represent a desired printed product.

Each individual node in JDF is defined in terms of inputs and outputs. The inputs for a node consist of the resources it uses and the parameters that control it. For example, the inputs in a node describing the process parameters for imaging the cover of a brochure might include requirements for trapping, RIP'ing, and imposing the image. The output of such a node might be a raster image.

Unless they represent the absolutely final product, resources that are produced by one node are in turn modified or consumed by subsequent nodes. Therefore, the output of the process described above—the raster image—becomes one of the input resources for a node describing the printing process for the brochure. This input resource would be joined in the node by other input resources such as inks, press sheets, plates, and a set of parameters that indicate how many sheets should be produced. The output would be a set of printed press sheets that in turn would become the input resource for postpress operations such as folding and cutting. And so on until the brochure is completed.

This system of interlinked nodes effectively unites the prepress, press, and postpress processes, and even extends the notion of where a job begins. A JDF job, like any printing job, is defined by the original intent for the end product. The difference between a JDF job and a generic printing job, however, is that JDF allows the entire job, from prepress through postpress, to be defined up front. All of the resources and processes necessary to produce an entire printed product can be identified and organized into nodes before the first prepress process is set in motion. Furthermore, the product intent specification can be extremely broad *or* extremely detailed, or anywhere in between. This means that a job may be so well defined before production begins that the system administrator only has to set the wheels in motion and let the job run its course. It may also mean that the person submitting the job has only a general idea of what the final product will look like and that modifications to the intent will be made along the way, depending on the course of the job.

For example, the person submitting the job specification for the brochure described above may know that she wants 400 copies, that she wants it done on a four-color press with no spot colors, that the cover will be on a particular paper stock and the contents on another, that the binding will be stapled, and that she requires the job in two weeks. Another person might know only that he wants the pages she's designed to be put into some sort of brochure form, although she doesn't know exactly what. Either person's request can be translated into a JDF product intent node that will eventually branch into a tree structure describing each process required to complete the brochure. In the first example, the prepress, press, and postpress processes will be well defined from the start. In the second example, information will be included as it is gathered. The following sections describe the way in which nodes can combine to form a job.

2.2.1 Job Structure

JDF jobs consist of a set of nodes that specify the production steps needed to create the desired end product. The nodes, in addition to being connected through inputs and outputs, are arranged in a hierarchical tree structure. Figure 2.2, below, shows a simple example of a tree of nodes.
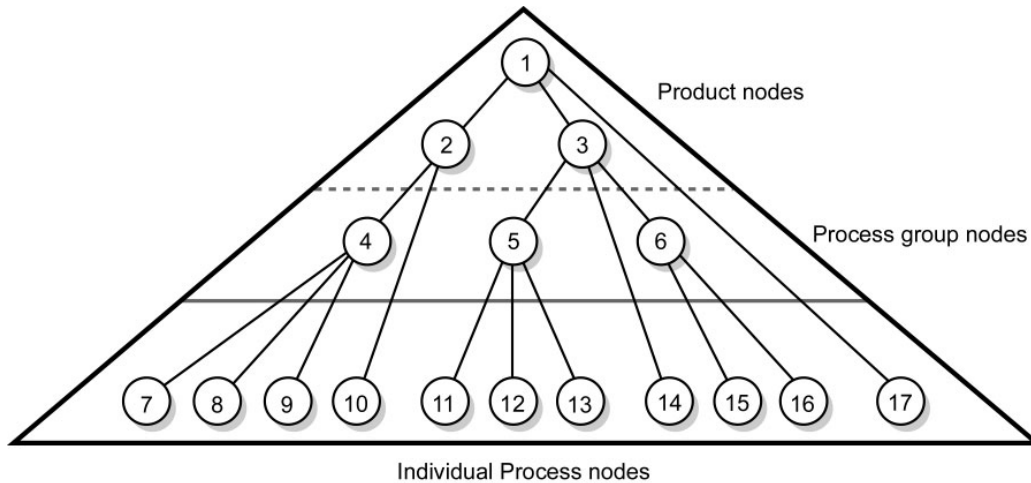
Figure 2.2 JDF tree structure

The following table provides a hypothetical breakdown of the nodes in the tree structure shown above:

Table 2-1 Information contained in JDF nodes, arranged numerically

Node #	Meaning
1	Entire book
2	Cover
3	Contents
4	Production of cover
5	Production of all color pages

Node #	Meaning
6	Production of all black-and-white pages
7	Cover production process 1
8	Cover production process 2
9	Cover production process 3
10	Cover Finishing process
11	RIP'ing for color pages
12	Plate making for color pages
13	Printing for color pages
14	Color page finishing process
15	RIP'ing for black-and-white pages
16	Printing for black-and-white pages on a digital press
17	Binding process for entire book

The uppermost nodes (1, 2, & 3) represent the product intent in general terms. These nodes describe the desired end product and the components of that product, which, in this case, are the cover and the content pages. As the tree branches, the information contained within the nodes gets more specific. Each subnode defines a component of the product that has a unique set of characteristic, such as different media, different physical size, or different color requirements. The nodes that occur in the middle of the tree (4, 5, & 6) represent the groups of processes needed to produce each component of the product. The nodes that occur closest to the bottom of the tree (7 – 17) each represent individual processes.

In this example, there are two subcomponents of the job, the cover and the contents, each with distinct requirements. Therefore, two nodes—nodes 2 and 3—are required to describe the elements of the job in broad terms. Within the content pages there are some black-and-white pages and some color pages. Since fabricating each requires a different set of processes, further branching is necessary. The following table arranges the nodes in groups according to the processes they will be executing:

Table 2-2 Information contained in JDF nodes, arranged by group

Process Group	Node #	Meaning
Entire book	1	Entire book
	17	Assemble book
Cover	2	Cover
	4	Cover assembly processes
	7	Cover production process 1
	8	Cover production process 2
	9	Cover production process 3
	10	Finishing process for cover
Contents	3	Contents
Color Pages	5	Production of all color pages
	11	RIP'ing for color pages
	12	Plate making for color pages
	13	Printing for color pages
	14	Color page finishing
Black-and-white pages	6	Production of all black-and-white pages
	15	RIP'ing for black-and-white pages
	16	Printing for black-and-white pages on a digital press

This hierarchical structure is discussed in more detail in the following section.

2.3 Hierarchical Tree Structure and Networks in JDF

Output resources of JDF nodes are often the input resources for other JDF nodes. Many nodes cannot begin executing until all of their resources are complete and ready. This means that the nodes execute in a well defined sequence. One process follows the next. For example, a process for making plates will produce, as output resources, press plates that are required by a printing process.

In the hierarchical organization of a JDF job, nodes that occur higher in the tree represent high level, more abstract operations, while lower nodes represent more detailed process operations. More specifically, nodes near the top of the tree may represent only intent regarding the components or assemblies that make up the product, while the leaf nodes provide explicit instructions to a device to perform some operation. Figure 2.3 shows an example of a hierarchical structure.

Trees & Nodes

In the real world, if you wanted to scan a photo, you would probably go to the prepress department to find a scanner. JDF uses this same common-sense approach to organization. Processes (nodes) are organized into a hierarchy (tree). Consider your own operations. If you were to group your departments, equipment, and processes into an "org chart," what would it look like?
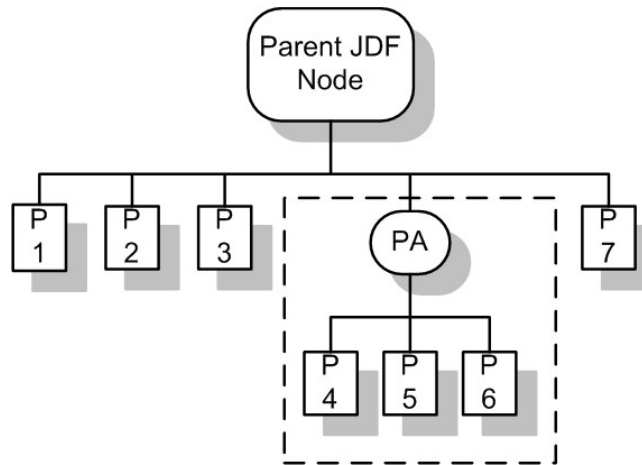
Figure 2.3 Example of a hierarchical tree structure of JDF nodes

In addition to the hierarchical structure of the node tree, sibling nodes are linked in a process chain by their respective resources. In other words, an output resource of one node ends up representing the input resource of the following node (as represented in Figure 2.4). This interrelationship is known as resource linking.

With resource linking, complex networks of processes can be formed. Figure 2.4 displays an alternate representation of the process described in Figure 2.3. Whereas Figure 2.3 represents a hierarchical structure, Figure 2.4 shows an example of the linking mechanism of the same job. Note that there are many possible process networks that map to the same node hierarchy.

P 2

R7

P 1

in.

R1

out.

P 7

P 3

R3

P 4

R4

P 5

R5

P 6

Key:
"P" = Process
"R" = Resource

PA
(Node consisting of process P4, P5, & P6)
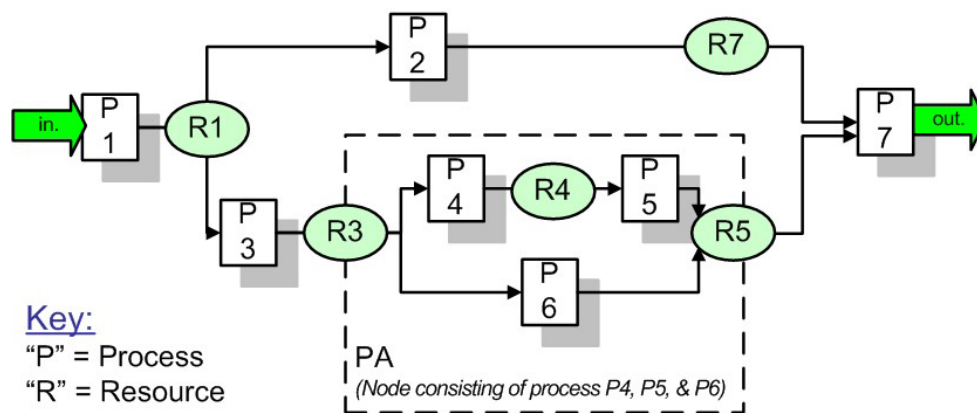
Figure 2.4 Example of a process chain linked by input and output resources

In JDF, the linking of processes is not explicitly specified. In other words, nodes are not arranged in an abstract chronology, dictating, for example, that the trapping node must come before the RIP'ing node. Rather, the links are implicitly defined in the exchange of inputs and outputs. Resource dependencies form a network of processes, and the sequence of process execution—that is, the routing of processes—can be derived from these dependencies. One resource dependency might have the possibility of multiple process routing scenarios. It is up to MIS to define the proper solution to meet local constraints.

The agent or set of agents employed by MIS to write the JDF job must be familiar with these local constraints. They must take into account factors such as the control abilities of the applications that complete the prepress processes, the transport distance between the prepress facility and the press itself, the load capabilities of the press, and the time requirements for the job. All of the factors taken together build a process network representing the workflow of production. To aid agents in defining the workflow, JDF provides the following four different and fundamental types of process routing mechanisms, which may be combined in any way:

1. **Serial processing** that is subsequent production and consumption of resources as a whole, represented by a simple process chain.
2. **Overlapping processing** that is simultaneous production and consumption of resources by pipes.
3. **Parallel processing** that involves the splitting and sharing of resources.
4. **Iterative processing** that is a circular or back and forward processing for developing resources by repeated activity.

These mechanisms are discussed in greater detail in Section 4.3 Execution Model.

2.4 Role of Messaging in JDF

JDF provides a container to define a job. Messaging language in JMF, defined in Chapter 5, provides a method to generate snapshots of job status and to interactively manipulate elements of a workflow system.

JMF is specifically designed for communication between the production system controller and the work centers or devices with which it interacts. It provides a series of queries and commands to check the status of processes and, in some cases, to dictate the next course of action. For example, the KnownDevices and KnownJDFServices queries allow the controller to determine what processes can be executed by a particular device or workcenter. These processes are likely to be determined at system initialization time. The QueueEntry messages provide a means for the controller to submit a job ticket to individual work centers or devices. And the Status, Resource and Occupation messages allow the device or work center to communicate quasi real-time[1] processing status to a controller. Depending on the system configuration, the message handler may choose to record status changes in the history logs. The status message allows the controller to request status updates from the controller.

JDF also provides mechanisms to define recipients for individual messages on a node-by-node basis. This enables controllers to define the aspects and the parts of jobs that they want to track. For more information about messaging, see Chapter 5 **JDF Messaging with the Job** Messaging Format.

---

[1] Real-time is the time-scale typically associated with macro-cosmic production control systems. JMF is not intended for real-time, lower level machine control.

2.5 Coordinate Systems in JDF

This chapter explains how coordinate systems are defined and used in JDF. It also shows how the matrices are used to specify a certain transformation and how these matrices can be used to transform coordinates from one coordinate system to another coordinate system. In addition it clarifies the meaning of terms like *Top* or *Left*.

2.5.1 Introduction

During the production of a printed product it often happens that one object is placed onto another object. During imposition, for example, single pages and marks (like cut, fold, or register marks) are placed on a sheet surface. Later, at image setting, a bitmap containing one separation of a sheet surface is imposed on a piece of film. In a following step, the film is copied to a printing plate, which then is mounted on a press. In postpress, the printed sheets are gathered on a pile. The objects involved in all these operations have a certain orientation and size when they are put together. In addition one has to know *where* to place one object on the other.

The position of an object, e.g., a cut mark, on a plane can be specified by a two-dimensional coordinate. Every digital or physical resource has its own coordinate system. The origin of each coordinate system is located in the lower left corner, i.e., the X coordinate increases from left to the right and the Y coordinate increases from bottom to top.
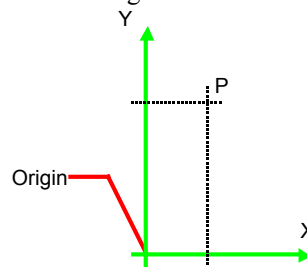
Figure 2.5 Standard coordinate system

Each page contained in a PDL file has its own coordinate system. In the same way a piece of film or a sheet of paper has a coordinate system. Within JDF each of these coordinate systems is called *resource coordinate system*.

If a process has more than one input resources with a coordinate system, it is necessary to define the relation between these input coordinate systems. Therefore, an idealized *process coordinate system* is defined for each process. The coordinate systems of the input resources are mapped to the process coordinate system. Each of those mappings is defined by a transformation matrix, which specifies how a coordinate (or position) of the input coordinate system is transformed into a coordinate of the target coordinate system. See Section 2.5.6 Homogeneous Coordinates for mathematical background information. In the same way the mapping from the process coordinate system to the coordinate systems of the output resources is defined. The process coordinate system is also used to defined the meaning of terms like *Top* or *Left*, which are used as values for parameters in some processes.
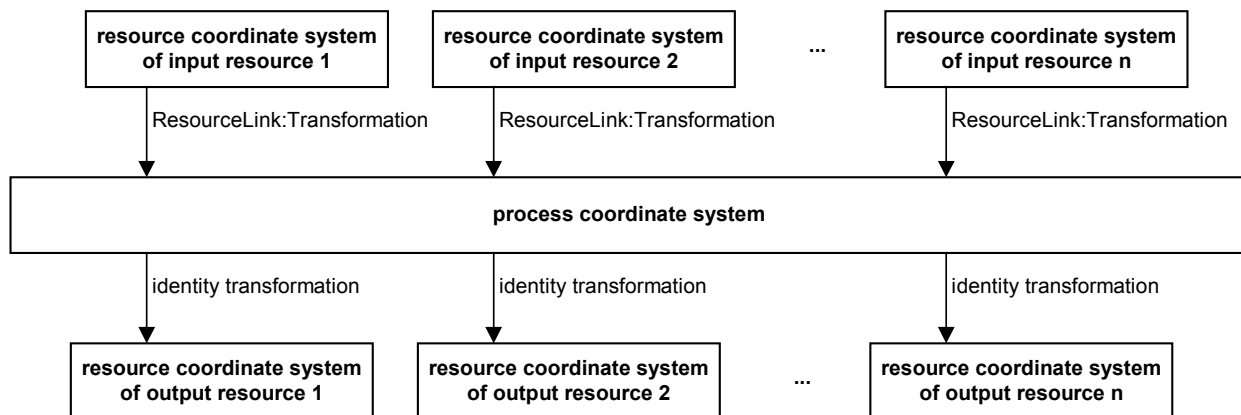
Figure 2.6. Relation between resource and process coordinate systems

It is important that no implicit rotations are assumed if the dimensions of the input resources of a process do not match each other. Instead every transformation (e.g., a rotation) must be specified explicitly by using the *Orientation* or *Transformation* attribute of the corresponding ResourceLink. The same applies also to other areas in JDF, e.g., the **LayoutPreparation** process.

2.5.2 How and Where Coordinates and Transformations Are Used/Defined in JDF

The following data types are used for the specification of coordinates and transformation:

- XYPair   "612 792"

- Number   "20.7"

- Rectangle   "0 0 595 843" *(Order of elements is "lower-left x, lower-left y, upper-right x, upper-right y" or "left, bottom, right, top".)*

- Matrix   "1 0 0 1 30.0 235.3" (The ordering of elements is defined in 2.5.6 Homogeneous Coordinates)

- Named orientations   "*Rotate180*" or ""*Flip90*"

Coordinates and transformations are used throughout JDF, to include:

Intent Resources, such as:

- LayoutIntent   specifies size of finished product
- MediaIntent   specifies size of media
- InsertingIntent   specifies rotation and offset

Process Resources, such as:

- Component   specifies coordinate system
- CutBlock   specifies cut block coordinate system
- FoldingParams   specifies folding operations

2.5.3 Coordinate Systems of Resources and Processes

Each physical input **Resource**, e.g., **Component** of a process has, by default, its own coordinate system, which is called the source or resource coordinate system. The coordinate system also implies a specific orientation of that **Resource**. On the other hand there is a coordinate system that is used to define various process-specific parameters. This coordinate system is called target or process coordinate system.

It is often necessary to change the orientation of an input **Resource** before executing the operation. This can be done by specifying a transformation matrix. It is stored in the *Orientation* or *Transformation* attribute of the ResourceLink. This provides the ability to specify different matrices for the individual resources of a process.

The following table shows some matrices that can be used to change the orientation of a physical **Resource**. Most of the transformations require the X- (**w**) and the Y-dimension (**h**) of the **Component** as specified in the Dimension element. If these are unknown, it is still possible to define a general orientation in the *Orientation* attribute of the ResourceLink. The naming of the attribute reflects the state of the Resource and not necessarily the order of applied transformations. Thus *Rotate90* and *Flip90* specify that the original Y axis as represented by the spine is on top. In the case of *Flip90*, the **Component** is additionally flipped front to back.

Table 2-3 Matrices and names used to describe the orientation of a Component

Orientation Name	Source Coordinate System	Transformation Matrix According Action	Target Coordinate System
Rotate0		1 0 0 1 0 0 No Action	
Rotate180		-1 0 0 -1 *w* *h* 180° Rotation	
Rotate90		0 1 -1 0 *h* 0 90° Counterclockwise Rotation	
Rotate270		0 -1 1 0 0 *w* 90° Clockwise Rotation	
Flip180		-1 0 0 1 *w* 0 Horizontal Flip	
Flip0		1 0 0 -1 0 *h* Vertical Flip	
Flip270		0 1 1 0 0 0 90° Counterclockwise Rotation + Horizontal Flip	
Flip90		0 -1 -1 0 *h* *w* 90° Clockwise Rotation + Horizontal Flip	

The descriptions of **Component**-specific attributes use some terms whose meaning depends on the culture in which they are used. For example, different cultures mean different things when they refer to the "front" side of a magazine.

Other terms, such as binding, are defined by the production process and therefore do not depend on the culture. Whenever possible, this specification endeavors to use culture-independent terms. In cases where this is not possible, Western style (left-to-right and top-to-bottom writing) is assumed. Please note that these terms may have a different meaning in other cultures (such as those writing from right to left).

Book-like product viewed from first page (front side)   Calendar-like product viewed from first page (front side)
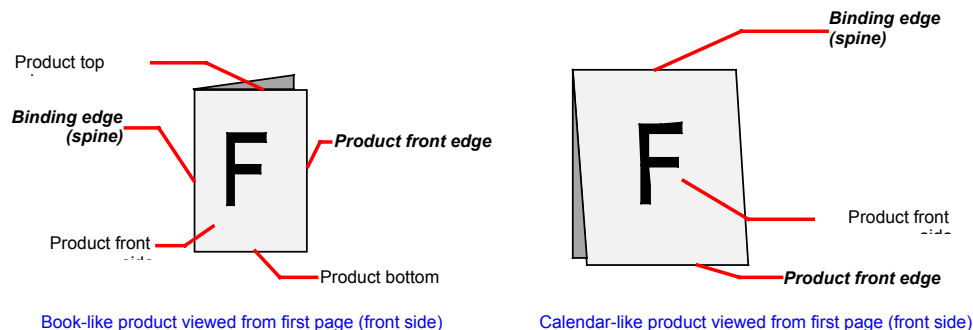
Figure 2.7 Terms and definitions for components

2.5.3.1 Coordinate Systems in Combined processes

Combined processes have no resource links defined between the individual processes that comprise the combined process. Therefore, the coordinate transformations for these processes must be specified within the resources themselves. For example, StitchingParams defines a *ReferenceEdge* attribute. If the document is actually a landscape or a reverse-landscape document, the creator of the JDF supplies the appropriate transformed *ReferenceEdge* value as follows: To position a staple in the upper left hand corner of a landscape document when held for reading, the *ReferenceEdge* value must be *Bottom* and *StitchType* = "*Corner*" (since landscape is defined as a +90 degree rotation of the image with respect to the media from portrait, i.e., counter-clockwise). On the other hand, to position a staple in the upper left hand corner of a reverse-landscape document when held for reading, the *ReferenceEdge* value must be *Top* and *StitchType* = "*Corner*" (since reverse-landscape is defined as a -90 degree rotation of the image with respect to the media from portrait, i.e., clockwise). The same applies to the *HoleReferenceEdge* attribute of **HoleMakingParams**.

2.5.4 Product Example: Simple Brochure

To illustrate the use of coordinate systems in JDF, a simple saddle stitched brochure with eight pages is used as an example. The brochure is printed on two sheets with front and back. The two sheets are then folded, collected on a saddle, and saddle stitched. Finally the brochure is cut with a three-side trimmer. The following table lists the JDF processes used for the production of the simple brochure.

Input Resources	Process	Output Resources
Layout **RunList** (Document) **RunList** (Marks)	*Imposition*	**RunList**
RunList	*Interpreting*	**InterpretedPDLData**
InterpretedPDLData **Media** **RenderingParams**	*Rendering*	**RunList** (rasterized ByteMaps)
RunList (rasterized ByteMaps)	*Screening*	**RunList** (Bitmaps)
ImageSetterParams **Media** (Film) **RunList** (Bitmaps)	*ImageSetting* (to Film)	**ExposedMedia** (Film)
ExposedMedia (Film)	*ContactCopying*	**ExposedMedia** (Plate)

Input Resources	Process	Output Resources
ExposedMedia (Plate) **ConventionalPrintingParams**	*ConventionalPrinting*	**Component** (Good)
FoldingParams **Component**	*Folding*	**Component**
CollectingParams **Component**	*Collecting*	**Component**
SaddleStitchingParams **Component**	*SaddleStitching*	**Component**
TrimmingParams **Component**	*Trimming*	**Component**

At imposition, the layout describes a signature with two sheets, each having a front and a back surface. On each surface, two content objects, i.e., pages, are placed.
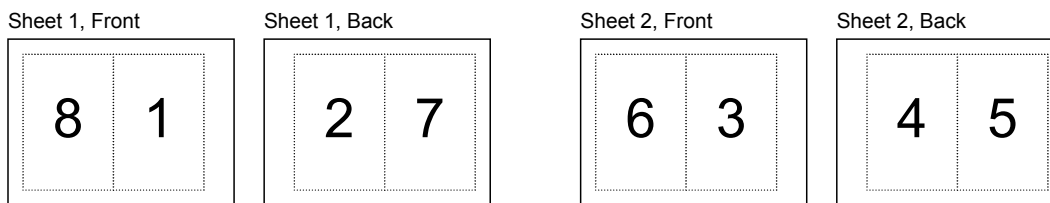
Figure 2.8 Layout of simple saddle stitched brochure (product example)

Each surface has its own coordinate system, in which a surface contents box is defined. This coordinate system is also referred to as the **Layout** coordinate system because the **Surface**, **Sheet,** and **Signature** elements are defined within the hierarchy of the **Layout** resource. The content objects are placed by specifying the CTM attribute relative to the surface contents box. If the position of an object within a page is given in the page coordinate system, this coordinate can be transformed into a position within the surface coordinate system:

$$P_{\text{Surface}} = P_{\text{Page}} \times \text{CTM}_{\text{Page}} + \left[\text{SurfaceContentsBox}_{\text{Xlowerleft}} \quad \text{SurfaceContentsBox}_{\text{Ylowerleft}} \quad 0 \right]$$

Please note, that the width and height of the surface are not known at this point.
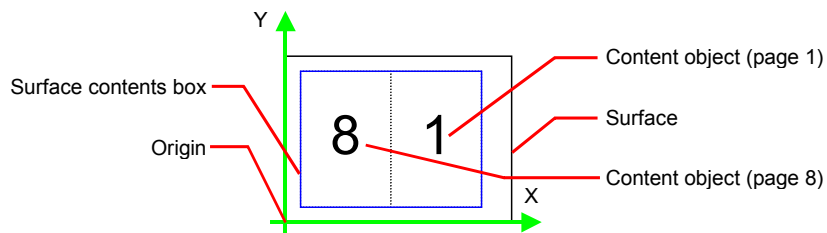
Figure 2.9 Surface coordinate system

The sheet coordinate system is identical with the coordinate system of the front surface. This means that no transformation is needed to convert a coordinate from one system to the other. Instead, the coordinates are valid (and equal) in both coordinate systems. The relation between the coordinate system of the front and the back surfaces depends on the value of the *Sheet:LockOrigins* attribute. The sheet coordinate system is also identical with the signature coordinate system, which in turn is identical with the coordinate system of the imposition process.

The output resource of the imposition process is a run list. Each element of the run list has its own coordinate system, which is identical with the corresponding signature coordinate system. The interpretation, rendering and screening processes do not affect the coordinate systems. This means that the coordinate systems of all these processes are identical.

At the image setting process, the digital data is set onto film. The process coordinate system is defined by the media input resource. The width and height of the media are defined in the *Media:Dimension* attribute. The position of the signatures (as defined by the run list input resource) on the film is defined by the *ImageSetterParams:CenterAcross* attribute.

The coordinate system of the conventional printing process is called *press coordinate system*. It is defined by the press: the X-axis is parallel to the press cylinder, and the Y-axis is going along the paper travel. Y = 0 is at begin of print, X = 0 is at the left edge of the maximum print area. The relation between the layout coordinate system and the press coordinate system is defined by the *CTM* attributes of the corresponding TransferCurveSet elements located in the TransferCurvePool.
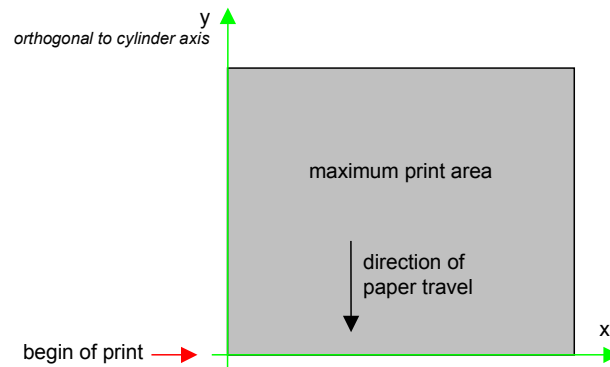
Figure 2.10. Press coordinate system used for sheet-fed printing
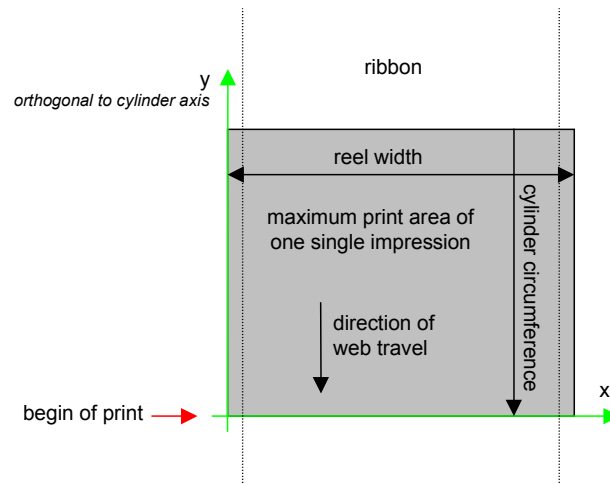
Figure 2.11 Press coordinate system used for web printing

The output of the printing process, e.g., a pile of printed sheets, is described as a **Component** resource in JDF. The coordinate system of the printed sheets is defined by the transformation given in the *TransferCurveSet:CTM* attribute (where *Name* = *Paper*).

Each of the two sheets is folded in a separate folding process. In this example, the orientation of the sheets is not changed before folding. This can be specified by setting the *Orientation* attribute of the input resource to *Rotate0* or by setting the Transformation *attribute* to "1 0 0 1 0 0". The folding process changes the coordinate system. In this example the origin of the coordinate system is moved from the lower left corner of the flat sheet (input) to the lower left corner of the folded sheet (output), i.e., it is moved to the right by half of the sheet width.
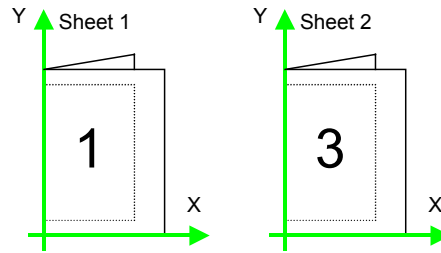
Figure 2.12 Coordinate systems after Folding (product example)

The two folded sheets are now collected. In this example, the orientation of the folded sheets is not changed before collecting. This can be specified by setting the **Orientation** attribute of the input resource to *Rotate0* or by setting the **Transformation** attribute to "1 0 0 1 0 0". The collecting process does not change the coordinate system.
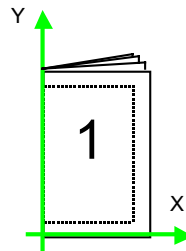
Figure 2.13 Coordinate systems after Collecting (product example)

The two collected and folded sheets are now trimmed to the final size of the simple brochure. In this example, the orientation of the collected and folded sheets is not changed before trimming. This can be specified by setting the **Orientation** attribute of the input resource to *Rotate0* or by setting the Transformation **attribute** to "1 0 0 1 0 0". The trimming process changes the coordinate system: the origin is moved to the lower left corner of the trimmed product.

In looking at the whole production process, a series of coordinate systems is being involved. The relation between the separate coordinate systems is specified by transformation matrices. This allows transformation of a coordinate from one coordinate system to another coordinate system. As an example, note the position of the title on page 1 of the product example in Figure 2.13. By applying the first transformation, this position can be converted into a position of the surface (or layout) coordinate system. This position can then be converted into the paper coordinate system by applying (in this order) the *Film*, *Plate*, *Press*, and *Paper* transformations stored in the TransferCurvePool.

From now on, every process is using components as input and output resources. The resource link of each input and output component contains a **Transformation** attribute or an **Orientation** attribute. The **Transformation** attribute is used if the width and the height of the component are known. Otherwise the **Orientation** attribute must be used to specify a change of the orientation, e.g., a rotation.

Since the folding process changes the coordinate system depending on the fold type, the transformations specified in the resource links are not sufficient to transform a position given in the paper coordinate system to a position in the coordinate system of the folded sheets, i.e. the resource coordinate system of the output component of the folding process. An additional transformation depending on the fold type has to be applied. The corresponding transformation matrix is not explicitly stored in the JDF file.

The collecting process does not change the coordinate system. Therefore, only the transformations specified in the resource links of the input and output resources, i.e. components, have to be applied.

The trimming process again changes the coordinate system depending on the trimming parameters. Therefore, a transformation depending on the trimming parameters has to be applied in addition to the transformations specified in the resource links. The matrix for the additional transformation (depending on the trimming parameters) is not explicitly stored in the JDF file.

After having applied all transformations mentioned above, the resulting coordinate specifies the position of the title in the coordinate system of the final product.

```
┌──────────────────────────────────────────────────┐
│ page coordinate system                            │
│ = resource coordinate system of input component   │
└──────────────────────────────────────────────────┘
                    │
         Surface:SurfaceContentsBox and CTM_Page
                    ▼
┌──────────────────────────────────────────────────┐
│ surface coordinate system = layout coordinate system │
│ = process coordinate system of Imposition, Interpreting, Rendering, Screening │
└──────────────────────────────────────────────────┘
                    │
         TransferCurveSet:CTM (Name = Film)
                    ▼
┌──────────────────────────────────────────────────┐
│ film coordinate system                            │
│ = process coordinate system of ImageSetting       │
└──────────────────────────────────────────────────┘
                    │
         TransferCurveSet:CTM (Name = Plate)
                    ▼
┌──────────────────────────────────────────────────┐
│ plate coordinate system                           │
│ = process coordinate system of ContactCopying     │
└──────────────────────────────────────────────────┘
                    │
         TransferCurveSet:CTM (Name = Press)
                    ▼
┌──────────────────────────────────────────────────┐
│ press coordinate system                           │
│ = process coordinate system of ConventionalPrinting │
└──────────────────────────────────────────────────┘
                    │
         TransferCurveSet:CTM (Name = Paper)
                    ▼
┌──────────────────────────────────────────────────┐
│ paper coordinate system                           │
│ = resource coordinate system of output component of ConventionalPrinting │
│ = resource coordinate system of input component of Folding │
└──────────────────────────────────────────────────┘
                    │
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
┌──────────────────────────────────────────────────┐
│ process coordinate system of Folding              │
└──────────────────────────────────────────────────┘
                    │
         Transformation according type of fold and
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
┌──────────────────────────────────────────────────┐
│ resource coordinate system of output component of Folding │
│ = resource coordinate system of input component of Collecting │
└──────────────────────────────────────────────────┘
                    │
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
┌──────────────────────────────────────────────────┐
│ process coordinate system of Collecting           │
└──────────────────────────────────────────────────┘
                    │
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
┌──────────────────────────────────────────────────┐
│ resource coordinate system of output component of Collecting │
│ = resource coordinate system of input component of Trimming │
└──────────────────────────────────────────────────┘
                    │
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
┌──────────────────────────────────────────────────┐
│ process coordinate system of Trimming             │
└──────────────────────────────────────────────────┘
                    │
         Transformation according trimming parameters and
         ResourceLink:Transformation (or ResourceLink:Orientation)
                    ▼
       resource coordinate system of output component of Collecting
       = coordinate system of final product
```

Figure 2.6 Examples of Transformations and Coordinate Systems in JDF.

2.5.5 General Rules

The following rules summarize the use of coordinate systems in JDF:

- Every individual piece of material (film, plate, paper) has a *resource coordinate system.*

- Every process has a *process coordinate system.*

- Terms like *top*, *left*, etc., are used with respect to the *process coordinate system* in which they are used and are independent of orientation, i.e., *landscape* or *portrait*, and the human reading direction.

- The coordinate system of each input component is mapped to the process coordinate system.

- The coordinate system may change during processing, e.g., in **Folding**.

- The description of a product in JDF is independent of particular machines used to produce this product. When creating setup information for an individual machine, it might be necessary to compensate for certain machine characteristics. At printing, for example, it might be necessary to rotate a landscape job, because the printing width of the press is not large enough to run the job without rotation.

2.5.6 Homogeneous Coordinates

A convenient way to calculate coordinate transformations in a two-dimensional space is by using so-called homogeneous coordinates. With this concept, a two-dimensional coordinate P=(x,y) is expressed in vector form as [x y 1]. The third element "1" is added to allow the vector being multiplied with a transformation matrix describing scaling, rotation, and translation in one shot. Although this only requires a 2*3 matrix (as it is used in PostScript for example), in practice 3*3 matrices are much more common, because they can be concatenated very easily. Thus, the third column is set to "0 0 1".

$$\text{Trf} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix} \qquad \text{would in JDF be written as "a b c d e f"}$$

Some often used transformation matrices are

$$\text{Trf} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{identity transformation}$$

$$\text{Trf} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix} \qquad \text{translation by dx, dy}$$

$$\text{Trf} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{rotation by } \varphi \text{ degrees counter-clockwise}$$

Transforming a point
In this example, the position P given in the coordinate system A is transformed to a position of coordinate system B. The relation between the two coordinate systems is given by the transformation matrix Trf.
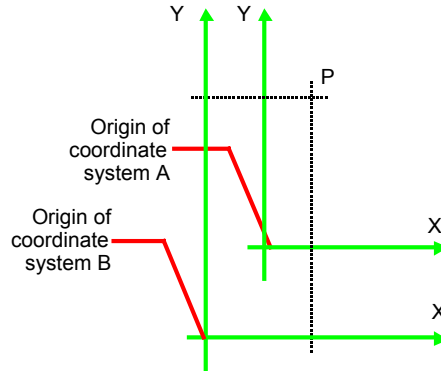
Figure 2.7 Transforming a point (example)

$$P_A = \begin{bmatrix} 30 & 100 & 1 \end{bmatrix}$$
$P_A = (30, 100)$

$$P_B = P_A \times Trf$$

$$P_B = \begin{bmatrix} 30 & 100 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 40 & 60 & 1 \end{bmatrix}$$
in JDF, *Trf* is written as "1 0 0 1 40 60"

$$P_B = \begin{bmatrix} 70 & 160 & 1 \end{bmatrix}$$
$P_B = (70, 160)$

Chapter 3 Structure of JDF Nodes and Jobs

Introduction

This chapter describes the structure of JDF nodes and how they interrelate to form a job. As described in Section 2.1.1 Job Components, a node is a construct, encoded as an XML element, that describes a particular part of a JDF job. Each node represents an aspect of the job: 1.) in terms of a process necessary to produce the end result, such as imposing, printing, or binding; 2.) in terms of a product that contributes to the end result, such as a brochure; or 3.) in terms of some combination of the previous two. In short, a node describes a product or a process.

In addition to describing the structure of an individual JDF node, this chapter examines in what way those nodes interact to form a coherent job structure. The interrelation of nodes can be divided into two categories: hierarchical and lateral. Hierarchical interrelation is the nested structure of parent nodes that contain child nodes. The visual correlative of this structure resembles a family tree, with a single node describing the entire job at the top, and a number of nodes at the bottom that each describe only one specific process. JDF-supported, leaf-level processes are described in Chapter 6 **Processes**.

Lateral interrelation, on the other hand, is the interrelation that occurs between nodes as a result of resource linking. Resource linking is the result of the transformation of inputs into outputs, which in turn may become inputs of other nodes. It also occurs when nodes share the same resource. The combination of hierarchical nesting of nodes and lateral linking allows complex process networks to be constructed. In a very simple case, however, a JDF file may contain only one node.

The hierarchical structure of a JDF job achieves a functional grouping of processes. For example, a job may be split into a prepress node, a press node, and a finishing node that contain the respective process nodes. Each and every node in turn contains attributes that represent various characteristics of that node. Nodes also contain subelements of certain types, such as resources, process information, customer information, audits, logging information, and other JDF nodes. Some elements, such as those that deal with customer information, generally occur only in the root structure, while other elements, such as resources, may occur anywhere in the tree. Where the elements can reside depends on their type and their usage scope.

This chapter describes the elements, subelements, and attributes commonly found in JDF nodes, and provides the characteristics necessary to understand where each belongs and how it is used. Many of these characteristics are presented in tables, and each of these tables includes the following three columns:

- **Name**—Identifies the element being discussed.
- **Data Type**—Refers to the data type, all of which are described in Section 0. Only the data types **element** or **telem** (which is short for text element) are applied to elements. All other types are attributes.
- **Description**—Provides detail about the element or attribute being discussed.

The JDF workflow model is based on a resource/consumer model. JDF nodes are the consumers that are linked by input resources and output resources. The ordering of siblings within a node, however, has no effect on the execution of a node. All chronological and logical dependencies are specified using ResourceLinks, which are defined in Section 3.8 Resource Links.

Figure 3.1 is a schematic structure of the JDF node type. In this figure, generic attributes and elements (see Section 3.1.1 Generic Contents of JDF Elements) are inserted only in the JDF root node. The element types that are displayed in this figure are described in the subsequent sections. Abstract data types are surrounded by a dashed line. Types derived from the abstract data type Resource are shown schematically in Figure 3.4.

Figure 3.1 Structure of the JDF Node
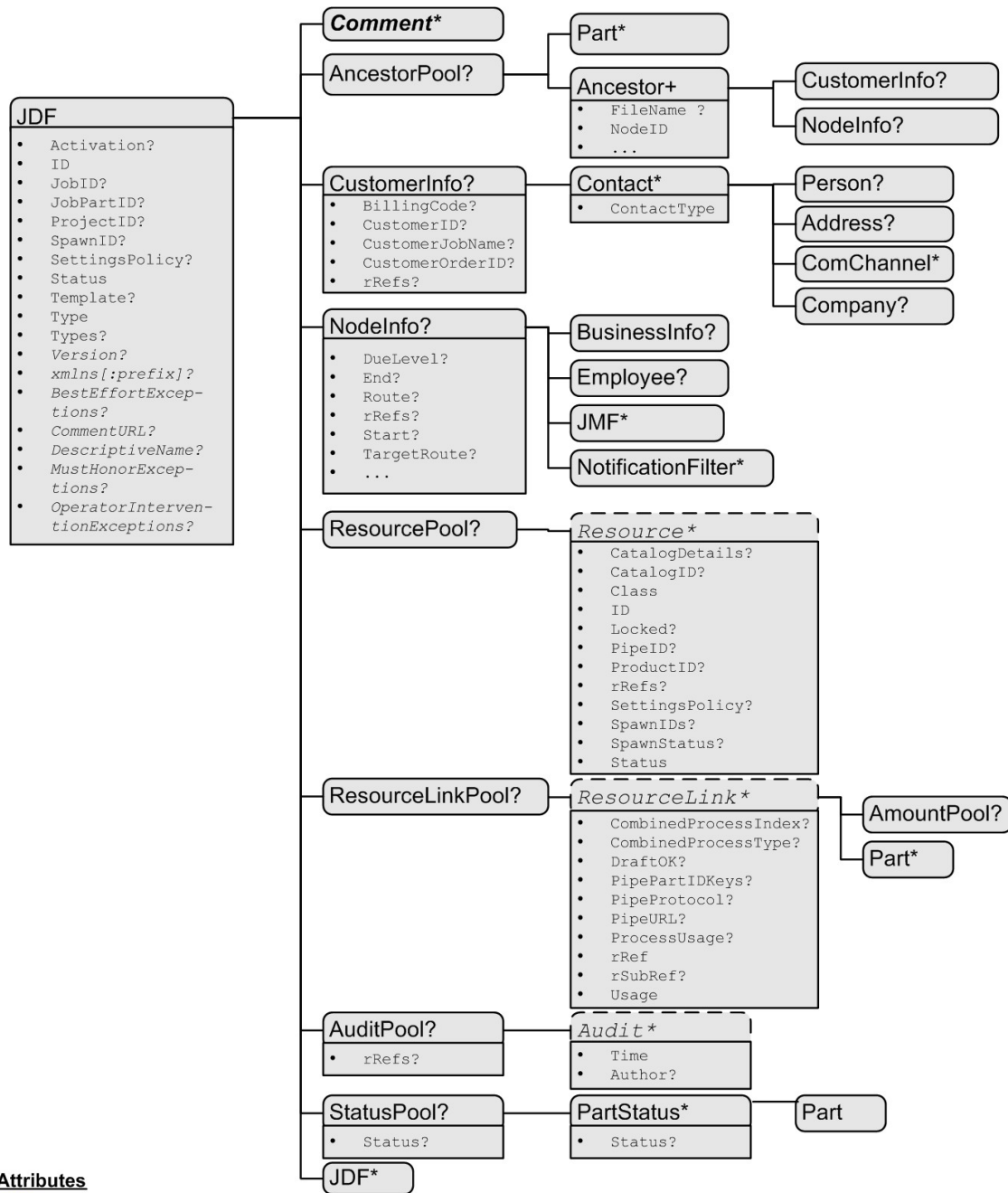
Attributes

JDF:

Type	= Product | ProcessGroup | Combined | *any process name*
Status	= Waiting | TestRunInProgress | Ready | FailedTestTun | Setup | InProgress | Cleanup | Spawned | Stopped | Completed | Aborted | Pool
Activation	= Inactive | Informative | Held | TestRun | TestrunAndGo | **Active**

Resource:

Status	= Incomplete | Unavailable | InUse | Draft | Complete | Available
SpawnStatus	= NotSpwaned | SpawnedRO | SpawnedRW
Locked	= **false** | true ; (volatile or persistent)

ResourceLink:

Usage	= Input | Output

3.1 JDF Nodes

JDF nodes are encoded as XML elements. Nodes, in turn, contain various attributes and further subelements, including nested JDF nodes.

Many of the tables in this section contain a fourth column that provides further details about the valid range of the attribute/element content, how the content is inherited by descendents (children, grandchildren, etc.), and where the attribute/element may reside in the JDF tree. The heading for this column is "Scope," which is short for "Scope and Position." The following abbreviations are defined:

D) Descendent: The content is valid locally within its node and in all descendent nodes, unless a descendent contains an identical attribute that overrides the content.

L) Local: The content is only valid locally, within the node where the content is defined.

R) Root: The attribute may only be specified in the root node. An exception from the localization only in the root node occurs if the spawning and merging mechanism for independent job tickets is applied as described in Section 4.4 Spawning and Merging.

All attributes and elements listed in subsequent chapters should be considered local, unless otherwise noted.

3.1.1 Generic Contents of JDF Elements

JDF contains a set of generic structures that may occur in any element of a JDF or JMF document. These are provided as containers for human-readable comments and descriptions and are described below.

Table 3-1 Generic Contents of elements

Name	Data Type	Description
BestEffortExceptions ? New in JDF 1.1	NMTOKENS	The names of the attributes in this element that are to have the best effort policy applied when JDF:*SettingsPolicy* or JDFResource:*SettingsPolicy* is not *BestEffort*. A JDF Consumer must support this attribute and must support any value of this attribute, so that an Agent can specify any exceptions to the *SettingsPolicy* in a JDF instance. The job will be processed by substituting or ignoring the attributes or attribute values that are not supported. *BestEffortExceptions* is ignored if the current value of *SettingsPolicy = BestEffort*.
CommentURL ?	URL	URL to an external, human-readable description of the element.
DescriptiveName ?	string	Human-readable descriptive name, e.g., a resource, process, or product.
MustHonorExceptions ? New in JDF 1.1	NMTOKENS	The names of the attributes in this element that are to have the must honor policy applied when JDF*:SettingsPolicy* or JDFResource:*SettingsPolicy* is not *MustHonor*. A JDF Consumer must support this attribute and must support any value of this attribute, so that an Agent can specify any exceptions to the *SettingsPolicy* in a JDF instance. The job will be rejected if any of these attributes or attribute values are not supported. *MustHonorExceptions* is ignored if the current value of *SettingsPolicy = MustHonor*.

Name	Data Type	Description
OperatorInterven-tionExceptions ? New in JDF 1.1	NMTOKENS	The names of the attributes in this element that are to have the operator intervention policy applied when JDF:*SettingsPolicy* or JDFResource:*SettingsPolicy* is not *OperatorIntervention*. A JDF Consumer must support this attribute and must support any value of this attribute, so that an Agent can specify any exceptions to the *SettingsPolicy* in a JDF instance. The job will be paused and the operator will be queried if any of these attributes or attribute values are not supported. If a device has no operator intervention capabilities, *OperatorIntervention* is treated as *MustHonor*. *OperatorInterventionExceptions* is ignored if the current value of *SettingsPolicy = OperatorIntervention*.
Comment *	telem	Any human-readable text.

The comment fields may contain a language attribute to support internationalization.

Table 3-2 Contents of the Comment element

Name	Data Type	Description
Attribute ? New in JDF 1.1	NMTOKEN	Name of the attribute in this element that the comment refers to. The name should include the prefix, if the attribute is in a non-JDF namespace.
Box ?	rectangle	The rectangle that is associated with the comment. The coordinate system of the rectangle is the same as the coordinate system defined in the Path attribute.
Language ?	language	Possible values are defined in IETF RFC 1766. If none is specified, the system specified value is assumed.
Name ?	NMTOKEN	A name that defines the usage of a comment. For example, it may determine whether two comments should fill two distinct fields of a user interface. Predefined values include: *Description* – Human readable description, which is required if the Comment element is required in a given context, as is the case in the Notification element (see Table 3-30 Contents of the Notification element). *Orientation* – Description of the orientation of a physical resource. Default = Description, which is required if the Comment element may become required, as is the case in the Notification element (see Table 3-30 Contents of the Notification element.
Path ?	path	Description of the area that the comment is associated with in the coordinate system of the element where the path resides. For example, if the comment is inserted in an ExposeMedia resource that describes a plate, the path refers to the plate coordinate system.
	text	Body of the comment.

The following figure shows the structure of the generic content defined above.
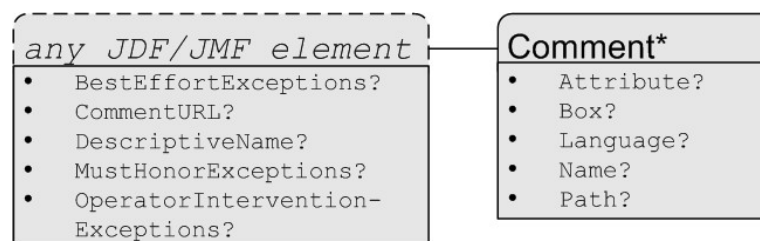
Figure 3.2 Structure of JDF Generic Contents

3.1.2 Fundamental JDF Attributes and Elements

The following table presents the attributes and elements likely to be found in any given JDF node. Three of the attributes in Table 3.3, below, are required, and must appear in every JDF node. Although the rest are designated as optional, they are optional in the sense that they are required only under certain circumstances, not that they may be left out if desired. The circumstances under which they are required are described in the Description column.

The most important of the attributes is the *Type* attribute, which defines the node type. The value of the *Type* attribute defines the product or process the JDF node represents. As is detailed in Section 3.2 Common Node Types, all nodes fall into one of the following four general categories: process, process group, combined processes and product intent. Each node is identified as belonging to one of these categories by the value of its *Type* attribute, as described in the table below. For example, if *Type* = *Product*, the node is a product intent node. Each of these categories is described in greater detail in the sections that follow.

Table 3-3 Contents of a JDF node

Name	Data Type	Scope	Description
Activation ? Modified in JDF 1.1	enumeration	special see text (D)	Describes the activation status of the node. Allows for a range of activity, including deactivation and testrunning. Possible values, in order of involvement from least to most active, are:
			Inactive – The node and all its descendents must not be executed or tested. This value is set if only certain parts of a JDF job should be executed or tested or if the node contains information required by other processes (as is the case with independent spawning and merging, described in Section 4.4.5).
			Informative – The JDF ticket is for information only. If a job is *Informative*, it must not be processed. Jobs with *Activation= Informative* will generally be sent to an operator console for preview but are still completely under the control of an external controller.
			Held – Execution has been held. If a job is *Held*, it must not be processed until its Activation is changed to Active.
			TestRun – The node requests a test run check by an controller or a device. This does not imply that the node should be automatically executed when the check is completed. Descendents of a node that is being test run are not to be considered *Active*.
			TestRunAndGo – Similar to *TestRun*, but requests a subsequent automatic start if the testrun has been completed successfully.
			Active – Default value. The node maybe executed as soon as all inputs are Available or Complete and all outputs are not incomplete.
			A child node inherits the value of the *Activation* attribute from its parent. The value of *Activation* corresponds to the least active value of *Activation* of any ancestor, including itself. Therefore, if any ancestor has an *Activation* of *Inactive*, the node itself is *Inactive*. If no ancestor is *Inactive* but any ancestor is *TestRun*, the node is *TestRun* unless the node itself is *Inactive*. If no ancestor has a value of *Inactive* or *TestRun* and any ancestor has a value of *TestRunAndGo*, the node has a value of *TestRunAndGo* unless that node is *Inactive* or *TestRun*, and so on.
			The following table illustrates the actions to be applied to a node depending on the value of *Activation*.
			Activation Test Node Execute Node

Name	Data Type	Scope	Description		
			Inactive	*false*	*false*
			Informative	*false*	*false*
			Held	*false*	*false*
			Active	*false*	*true*
			TestRun	*true*	*false*
			TestRunAndGo	*true*	*true*
ID	ID	L	Unique identifier of a JDF node. This ID is used to refer to the JDF node.		
JobID ?	string	D	Job identification used by the application that created the JDF job. Typically, a job is identified by the internal order number of the MIS system that created the job.		
JobPartID ?	string	D	Identification of a part of a job, used by the application that created the job. Typically, this is internal to the MIS system that created the job and coincides with a process or set of processes.		
ProjectID ? New in JDF 1.1	string	D	Identification of the project context that this JDF belongs to. Used by the application that created the JDF job.		
SpawnID ? New in JDF 1.1	NMTOKEN	D	Identification of a spawned part of a job. Typically this is used to map Audits and messages to a spawned processing step in the workflow.		
SettingsPolicy ? New in JDF 1.1	enumeration	D	The policy for this Node indicates what happens when unsupported settings, i.e., subelements, attributes or attribute values, are present in the resources. A JDF Consumer must support this attribute and all of the defined values so that an Agent can depend on the JDF Consumer following the policy requested by the Agent in a JDF instance. Possible values are: *BestEffort* – Substitute or ignore unsupported attributes, attribute values, default attribute values, or elements and continue processing the job. *MustHonor* – Reject the job when (1) any unsupported attributes, attribute values, or elements are present or (2) any omitted attributes have an unsupported default value defined in this specification. *OperatorIntervention* – Pause job and query the operator when (1) any unsupported attributes, attribute values, or elements are present or (2) any omitted attributes have an unsupported default value defined in this specification. If a device has no operator intervention capabilities, *OperatorIntervention* is treated as *MustHonor*. Default = *BestEffort*		
Status Modified in JDF 1.1	enumeration	L	Identifies the status of the node. Possible values are: *Waiting* – The node may be executed, but it has not completed a test run. *TestRunInProgress* – The node is currently executing a test run. *Ready* – As indicated by the successful completion of a test run, all ResourceLinks are correct, required resources are available, and the parameters of resources are valid. The node is ready to start. *FailedTestRun* – An error occurred during the test run. Error in-		

Name	Data Type	Scope	Description
			formation is logged in the **Notification** element, which is an optional subelement of the **AuditPool** element described in Section 3.10.
			Setup –The process represented by this node is currently being set up.
			InProgress – The node is currently executing.
			Cleanup – The process represented by this node is currently being cleaned up.
			Spawned – The node is spawned in the form of a separate spawned JDF.
			The status Spawned can only be assigned to the original instance of the spawned job. For details, see Section 4.4.
			Stopped – Execution has been stopped. If a job is *Stopped*, running may be resumed later. This status may indicate a break, a pause, maintenance, or a breakdown—in short, any pause that does not lead the job to be aborted.
			Completed – Indicates that the node has been executed correctly, and is finished.
			Aborted – Indicates that the process executing the node has been aborted, which means that execution will not be resumed again.
			Pool – Indicates that the node processes partitioned resources and that the **Status** varies depending on the partition keys. Details are provided in the **StatusPool** element of the node.
			Derivation of the **Status** of a parent node from the **Status** of child nodes is non-trivial and implementation-dependent.
Template ? New in JDF 1.1	boolean	R	Identifies a template that is used to generate JDFs but should not be exchanged as a job description. Default = "*false*".
Type	NMTOKEN	L	Identifies the type of the node. Any JDF process name is a valid type. The processes that have been predefined are listed in Chapter 6, although the flexibility of JDF allows anyone to create processes. In addition to these, there are three values which are described in greater detail in the sections that follow: *Combined* *ProcessGroup* *Product:* Identifies a Product Intent node.
Types ?	NMTOKENS	L	List of the **Type** attributes of the nodes that are combined to create this node. This attribute is required if **Type** = *Combined*, and is ignored if **Type** equals any other value. For details on using *Combined* nodes, see Section 3.2.3.
Version ? Modified in JDF 1.1	string	RD	Text that identifies the version of the JDF node. The current version of this specification is "1.1". The Version attribute is required in the JDF root node, but not in child nodes.
xmlns ? New in JDF 1.1	URI	RD	JDF supports use of XML namespaces. The namespace must be declared in the root JDF element. For details on using namespaces in XML, see http://www.w3.org/TR/REC-xml-names/. For version 1.1 of JDF *xmlns,* see http://www.CIP4.org/JDFSchema_1_1

Name	Data Type	Scope	Description
AncestorPool ?	element	R	If this element is present, the current JDF node has been spawned, and this element contains a list of all ancestors prior to spawning. See Section 3.3.
AuditPool ?	element	L	List of elements that contains all relevant audit information. Audits are intended to serve the requirements of MIS for evaluation and invoicing. See Section 3.10.
CustomerInfo ?	element	D	Container element for customer-specific information. See Section 3.4.
JDF *	element	L	Child JDF nodes. The nesting of JDF nodes defines the JDF tree. In contrast to the elements above, JDF child nodes are not contained in a list element.
NodeInfo ?	element	L	Container element for process-specific information such as scheduling and messaging setup. Scheduling affects the planned times when a node should be executed. Actual times are saved in the AuditPool. See Section 3.5 for more details.
ResourceLinkPool ?	element	L	List element for ResourceLink elements, which describe the input and output resources of the node. See Section 3.8 for more details.
ResourcePool ?	element	L[1]	List element for resources. See Section 3.6 for more details.
StatusPool ?	element	L	Lists the details of a nodes partition dependent *Status* if the *Status* of the node is "*Pool*".

3.2 Common Node Types

As was noted in the preceding section, the *Type* of a node can fall into four categories. The first is comprised of the specific processes of the kind delineated in **Chapter** 6, known simply as process nodes. The other categories are made up of three enumerative values of the *Type* attribute: *ProcessGroup*, *Combined*, and *Product*, which is also known as product intent. These three node types are described in this section.

The figure below, which was also presented as an illustration in Chapter 2, represents a theoretical job hierarchy comprised of *Product* nodes, *ProcessGroup* nodes, and nodes that represent individual processes. The diagram is divided into three levels to help illustrate the difference between the three kinds of nodes, but these levels do not dictate the hierarchical nesting mechanism of a job. Note, however, that an individual process node may be the child of a product intent node without first being the child of a process group node. Likewise, a process group node may have child nodes that are also process groups.

---

[1] Resources are unique and cannot be overwritten by descendents. Rather, they can only be used by descendents. An exception to this is described in Section 4.4.5 Case 5: Spawning and Merging of Independent Jobs. In this case, resources may also be used by a parent node.
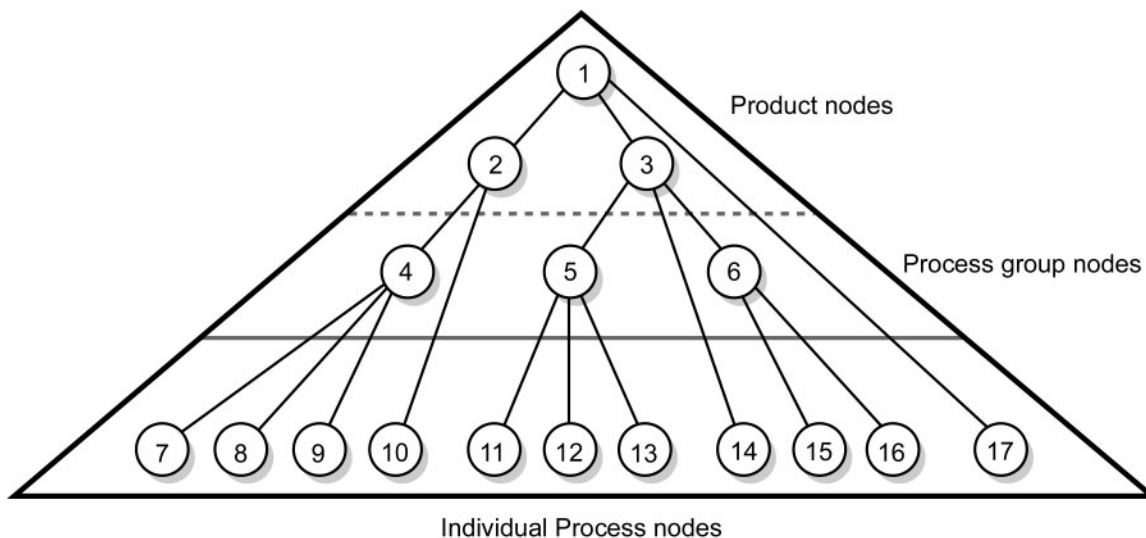
Figure 3.3 Job hierarchy with process, process group, and product intent nodes

3.2.1 Product Intent Nodes

Except in certain specific circumstances, the agent assigned to begin writing a JDF job will very likely not know every process detail needed to produce the desired results. For example, an agent that is a job-estimating or job-submission tool may not know what devices can execute various steps, or even which steps will be required.

If this is the case, the initiating agent creates a set of top-level nodes to specify the product intent, without providing any of the processing details. Subsequent agents then add nodes below these top-level nodes to provide the processing details needed to fulfill the intent specified.

These top-level nodes have a *Type* attribute value of *Product* to indicate that they do not specify any processing. All processing needed to produce the products described in these nodes must be specified in *Process* nodes, which exist lower in the job hierarchy.

Product nodes include intent resources that describe the end results the customer is requesting. The intent resources that have already been defined for JDF are easily recognizable, as they contain the word "intent" in their titles. Examples include **FoldingIntent** and **ColorIntent**. All intent resources share a set of common subelements, which are described in Section 7.1.1 **Intent Resource Span Subelements**. These resources do not attempt to define the processing needed to achieve the desired results; instead they provide a forum to define a range of acceptable possibilities for executing a job.

Each Product Intent node should contain at most one ResourceLink for one type of intent resource. If multiple product parts with different intents are required, each part has its own Product Intent node. **DeliveryIntent** resources are a notable exception. Specifying multiple **DeliveryIntent** resources effectively requests multiple options of a quote. For more information about product intent, see Section 4.1.1 Product Intent Constructs.

3.2.2 Process Group Nodes

Intermediate nodes in the JDF job hierarchy—i.e., nodes 4, 5, and 6 in Figure 3.3—describe groups of processes. The *Type* attribute value of these kinds of nodes is *ProcessGroup*. These nodes are used to describe multiple steps in a process chain that have common resources or scheduling data.

Since the agent writing the job has the option of grouping processes in any way that seems logical, custom workflows can be modeled flexibly. Process group nodes may contain further process group nodes, individual process nodes, or a mixture of both node types. Sequencing of process group nodes should be defined by linking resources of the appropriate leaves or, if the nature of the interchange resources is unknown, by linking **PlaceHolder** resources.

The higher the level of the process group nodes within the hierarchy, the larger the number of processes the group contains. A high level process group node might include, for example, prepress, finishing, or printing processes. Lower level process groups, on the other hand, define a set of individual steps that are executed as a group of

steps in the individual workflow hierarchy. For example, all steps performed by one designated individual may be grouped in a lower level process group.

The following example shows the ResourceLink structure for a *ProcessGroup* in-line finishing node. Note the presence of intermediate component links that link the individual processes. The corresponding **Component**s have been omitted for brevity.

```
<JDF Type = "ProcessGroup" ID = "J1">
  <JDF Type = "DigitalPrinting" ID = "J2">
    <ResourceLinkPool>
<!-- digital printing parameters -->
      <DigitalPrintingParamsLink Usage="Input" rRef="L1"/>
<!-- input sheets -->
      <MediaLink Usage="Input" rRef="L2"/>
<!-- printed output components -->
      <ComponentLink Usage="Output" rRef="L3"/>
    </ResourceLinkPool>
  </JDF>
  <JDF Type = "Gathering" ID = "J3">
    <ResourceLinkPool>
<!-- gathering parameters -->
      <GatheringParamsLink Usage="Input" rRef="L4"/>
<!-- printed output components -->
      < ComponentLink Usage="Input" rRef="L3"/>
<!-- gathered output components -->
      < ComponentLink Usage="Output" rRef="L5"/>
    </ResourceLinkPool>
  </JDF>
  <JDF Type = "Stitching" ID = "J4">
    <ResourceLinkPool>
<!-- Stitching parameters -->
<StitchingParamsLink Usage="Input" rRef="L6"/>
<!-- gathered output components -->
      <ComponentLink Usage="Input" rRef="L5"/>
<!-- stitched output components -->
      <ComponentLink Usage="Output" rRef="L7"/>
    </ResourceLinkPool>
  </JDF>
</JDF>
```

3.2.3 Combined Process Nodes

The processes described in Chapter 6 **Processes** define individual workflow steps that are assumed to be executed by a single-purpose device. Many devices, however, are able to combine the functionality of multiple single-purpose devices and execute more than one process. For example, a digital printer may be able to execute the *Interpreting*, *Rendering*, and *DigitalPrinting* processes. To accommodate such devices, JDF allows processes to be grouped within a node whose *Type* = *Combined*. Such a node must also contain a *Types* attribute, which in turn contains an ordered list of the *Type* values of each of processes that the node specifies. The ordering of the process names in the *Types* attribute is significant and specifies the ordering in which the processes are assumed to be executed.

Furthermore, ResourceLink elements in *Combined* nodes should specify a *CombinedProcessIndex* attribute in order to define the subprocess to which the resource belongs. *Combined* nodes are leaf nodes and must not contain further nested *JDF* nodes.

A device with multiple processing capabilities is able to recognize the *Combined* node as a single unit of work that it can execute. Therefore, all resources for each of the subtasks that define the *Combined* node and that are explicitly defined as ResourceLinks must be available before the node can be executed. In addition, all input and output resources that are consumed and produced externally by the process must be specified in the ResourceLinkPool element of the node. This includes all required Parameter resources as well as the initial input

resources and final output resources. Intermediate resources that are internally produced and consumed, on the other hand, need not be specified.

In a combined process node, the information defined by the various resources linked as input to the various subprocesses are logically available to all processes of the combined node. In situations where the parameter resource of more then one subprocess specifies the mapping of sheet surface content to media, the subprocess that specifies such a mapping that is defined earliest in the *Types* attribute list must be used, and any other mappings specified by any down-stream subprocess Resource must be ignored.

3.2.3.1 Combined Process Nodes with Multiple Processes of the Same Type

A *Combined* node may contain multiple instances of the same process type, e.g. *Types* = "*Cutting Folding Cutting*". In this case, the ordering and mapping of links processes is significant – the parameters of the first *Cutting* process are most likely to be different from those of the second *Cutting* process. Mapping is accomplished using the *CombinedProcessIndex* attribute in the respective ResourceLink.

```
<JDF Type = "Combined" Types = "Cutting Folding Cutting" ID = "J1">
<!—Resources  (incomplete…) -->
  <ResourcePool>
<!-- parameters of the first Cutting Process-->
    <CuttingParams ID="L1"/>
<!-- Folding parameters -->
    <FoldingParams ID="L2"/>
<!-- parameters of the third Cutting Process-->
    <CuttingParams ID="L3"/>
<!-- raw input components -->
    <Component ID="L4"/>
<!-- completed output components -->
    <Component ID="L5"/>
  </ResourcePool>

<!-- Links -->
  <ResourceLinkPool>
<!-- parameters of the first Cutting Process-->
    <CuttingParamsLink Usage="Input" CombinedProcessIndex="0" rRef="L1"/>
<!-- Folding parameters -->
    <FoldingParamsLink Usage="Input" CombinedProcessIndex="1" rRef="L2"/>
<!-- parameters of the first Cutting Process-->
    <CuttingParamsLink Usage="Input" CombinedProcessIndex="2" rRef="L3"/>
<!-- raw input components -->
    <ComponentLink Usage="Input" rRef="L4"/>
<!-- completed output components -->
    <ComponentLink Usage="Output" rRef="L5"/>
  </ResourceLinkPool>
</JDF>
```

3.2.3.2 Examples of Combined Process Nodes

The following example of the ResourceLinkPool of a JDF node describes digital printing with in-line finishing and includes the same processes as the previous ProcessGroup example. The node requires the parameter resources and consumable resources of all three processes as inputs, and produces a completed booklet as output. The intermediate printed sheets and gathered piles are not declared, since they exist only internally within the device and cannot be accessed or manipulated by an external controller.

```
<JDF Type = "Combined" Types = "DigitalPrinting Gathering Stitching" ID =
"J1">
  <ResourceLinkPool>
<!-- digital printing parameters -->
<DigitalPrintingParamsLink Usage="Input" CombinedProcessIndex="0" rRef="L1"/>
<!-- gathering parameters -->
    <GatheringParamsLink Usage="Input" CombinedProcessIndex="1" rRef="L4"/>
```

```
<!-- Stitching parameters -->
    <StitchingParamsLink Usage="Input" CombinedProcessIndex="2" rRef="L6"/>
<!-- input sheets -->
    <MediaLink Usage="Input" CombinedProcessIndex="0" rRef="L2"/>
<!-- stitched output components -->
    <ComponentLink Usage="Output" CombinedProcessIndex="2" rRef="L7"/>
  </ResourceLinkPool>
</JDF>
```

3.2.4 Process Nodes

Process nodes represent the very lowest level in a job hierarchy. They must not contain further nested JDF nodes, as every process node is a leaf node. These nodes define the smallest work unit that may be scheduled and executed individually within the JDF workflow model. In Figure 3.6 below, nodes 7-17 represent process nodes. The various individual process node types are specified in Chapter 6 **Processes**.

3.3 AncestorPool

When a job is spawned, an AncestorPool is created in the spawned job to iden-
tify its parents and grandparents. This allows storing of information about job
context in a spawned node as well as allowing the job to be correctly merged with
its parent after it is completed. The AncestorPool element is only required in the
root of a spawned job. Spawning and merging is described in Section 4.4
Spawning and Merging. The AncestorPool element contains an ordered list of
one or more Ancestor elements, which reflect the family tree of a spawned job.
Each Ancestor element identifies exactly one ancestor node. The ancestor nodes
reside in the original job where the job with the AncestorPool has been spawned
off. The position of the Ancestor element in the ordered list defines the position
in the family tree. The first element in the list is the original root element, the last
element in the list is the parent, the last but one the grandparent, and so on. The
following table lists the contents of an AncestorPool element.

Ancestor Pool

An ancestor pool contains the job's context when the job is spawned. This includes scheduling information and optionally customer information.
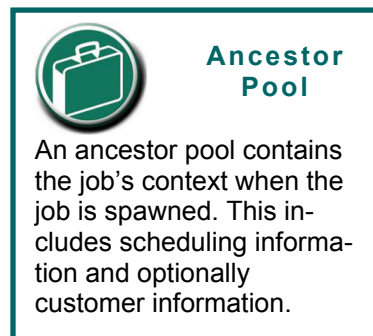
Table 3-4 Contents of the AncestorPool element

Name	Data Type	Description
Ancestor +	element	Ordered list of one or more Ancestor elements, which reflect the family tree of a spawned job.
Part * New in JDF 1.1	element	List of parts that this node was spawned with. Used in case of parallel Spawning of a node.

An Ancestor element may contain read only copies of all the attributes of the node that it represents with the excep-
tion of the *ID* attribute, which must be copied to the *NodeID* attribute of that Ancestor element. Ancestor ele-
ments cannot, however, contain further subelements except for read only copies of CustomerInfo and NodeInfo.
The attributes of Ancestor elements are described in

Table 3-5 Attributes of the Ancestor element

Name	Data Type	Description
Activation ?	enumeration	Copy of the *Activation* attribute from the ancestor node. For details, see Table 3-3.
FileName ?	URL	The URL of the JDF file where the ancestor node resided prior to spawning.
JobID ?	string	Copy of the *JobID* attribute from the ancestor node. For details, see Table 3-3.
JobPartID ?	string	Copy of the *JobPartID* attribute from the original ancestor node. For details,

Name	Data Type	Description
		see Table 3-3.
NodeID	NMTOKEN [2]	Copy of the *ID* attribute of the ancestor node.
ProjectID ?	string	Identification of the project context that this JDF belongs to. Used by the application that created the JDF job.
SpawnID ? New in JDF 1.1	NMTOKEN	Copy of the *SpawnID* attribute of the ancestor node.
Status ?	enumeration	Copy of the *Status* attribute from the original ancestor node. For details, see Table 3-3.
Type ?	NMTOKEN	Copy of the *Type* attribute from the original ancestor node. For details, see Table 3-3.
Types ?	NMTOKENS	Copy of the *Types* attribute from the original ancestor node. For details, see Table 3-3.
Version ?	string	Copy of the *Version* attribute from the original ancestor node. For details, see Table 3-3.
CustomerInfo ? New in JDF 1.1	element	Reference copy of the CustomerInfo element from the original node. For details, see Table 3-3.
NodeInfo ? New in JDF 1.1	element	Reference copy of the NodeInfo element from the original node. For details, see Table 3-3.

3.4 Customer Information

The CustomerInfo element contains information about the customer who orders the job. Usually, this element is specified in the uppermost node of a job (that is, the root node), although it is also valid in lower nodes in situations such as model subcontracting.   Table 3-6 Contents of the CustomerInfo element describes the contents of this element.

Creating Better
Job Tracking & Reporting

Customer information within JDF can provide a bridge between your CRM systems and production. How could JDF be used to automate the process of reporting to customers on the status of their jobs?

Table 3-6 Contents of the CustomerInfo element

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.
rRefs ?	IDREFS	Array of *ID*s of any elements that are specified as ResourceRef elements. In this version it will be the IDREF of a ContactRef[3].

---

[2] The data type is NMTOKEN and not IDREF because the ID does not reside in the spawned job. The corresponding ID element resides in the original job.

Name	Data Type	Description
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 must be one **Contact** which has ContactTypes including "Customer".

3.5 Node Information

The NodeInfo element contains information about planned scheduling and message routing. It allows MIS to plan, schedule and invoice jobs or job parts. Table 3-7 Contents of the NodeInfo element describes the contents of the NodeInfo element.

Table 3-7 Contents of the NodeInfo element

Name	Data Type	Description
CleanupDuration ?	duration	Estimated duration of the clean-up phase of the process.
DueLevel ?	enumeration	Description of the severity of a missed deadline. Possible values are: *Unknown* – Default value. Consequences of missing the deadline are not known. *Trivial* – Missing the deadline has minor or no consequences. *Penalty* – Missing the deadline incurs a penalty. *JobCancelled* – The job is cancelled if the deadline is missed.
End ?	dateTime	Date and time at which the process is scheduled to end.
FirstEnd ?	dateTime	Earliest date and time at which the process may end.
FirstStart ?	dateTime	Earliest date and time at which the process may begin.
IPPVersion ? [New in JDF 1.1]	XYPair	A pair of numbers indicating the version of the IPP protocol to use when communicating to IPP devices. The X value is the major version number. Default = system specified
JobPriority ? [New in JDF 1.1]	integer	The scheduling priority for the job where 100 is the highest and 1 is the lowest. Amongst the jobs that can be printed, all higher priority jobs should be printed before any lower priority ones. If one of the deadline oriented attributes, e.g., *FirstStart* or *LastEnd* and *JobPriority* are specified, the deadline oriented attributes must be honored before considering *JobPriority*. Default = 50.
LastEnd ?	dateTime	Latest date and time at which the process may end. This is the deadline to which *DueLevel* refers.
LastStart ?	dateTime	Latest date and time at which the process may begin.
NaturalLang ? [New in JDF 1.1]	language	Language selected for communicating attributes. If not specified, the operating system language is assumed.
MergeTarget ? [Deprecated in JDF 1.1]	boolean	If *MergeTarget* = *true* and this node has been spawned, it must be merged with its direct ancestor by the controller that executes this node. The path of the ancestor is specified in the last Ancestor element located in the AncestorPool of this node. It is an error to specify both *MergeTarget* and *TargetRoute* in one node. Default = *false*, which means that some other controller will take care of merging.

---

[3] rRefs also enables spawning and merging if CustomerInfo is extended with private ResourceRef elements.

Name	Data Type	Description
		Note: *MergeTarget* has been deprecated in JDF 1.1 because avoiding concurrent access to the ancestor node is ill defined and cannot be implemented in an open system without proprietary locking mechanisms.
Route ?	URL	The URL of the controller or device that should execute this node. If URL is not specified, the routing controller must determine a potential controller or device independently. For details, see Process Routing
rRefs ?	IDREFS	Array of *ID*s of any elements that are specified as ResourceRef elements. In this version it may be the IDREF of a JMFRef or EmployeeRef[4].
SetupDuration ?	duration	Estimated duration of the setup phase of the process.
Start ?	dateTime	Date and time of the planned process start.
TargetRoute ?	URL	The URL where the JDF should be sent after completion. If *TargetRoute* is not specified, it defaults to the input *Route* attribute of the subsequent node in the process chain. If this is also not known, the JDF should be sent to the processor default output URL.
TotalDuration ?	duration	Estimated total duration of the process, including setup and cleanup.
BusinessInfo?	element	Container for business related information. It is expected that JDF will be utilized in conjunction with other eCommerce standards, and this container is provided to store the eCommerce information within JDF in case a workflow with JDF as the root level document is desired. When JDF is used as part of an eCommerce solution such as PrintTalk, the information given in the envelope document overrides the information in BusinessInfo.
Employee ?	refelement	The internal administrator or supervisor that is responsible for the product or process defined in this node.
JMF *	element	Represents JMF query messages that set up a persistent channel, as described in Section 5.2.2.3 **Persistent Channels**. These message elements define the receiver that is designated to track jobs via JMF messages. These message elements should be honored by any JMF-capable controller or device that executes this node. When these messages are honored, a persistent communication channel is established that allows devices to transmit, for example, the status of the job as JMF Signals.
NotificationFilter *	element	Defines the set of Notification elements that should be logged in the AuditPool. This provides a logging method for devices that do no not support JMF messaging. For details of the NotificationFilter element, see 5.5.1.1 **Events**.

3.6 StatusPool

The StatusPool describes the *Status* of a JDF node that processes partitioned resources. StatusPool elements are only valid if the node's *Status*="*Pool*", otherwise the node's *Status* is valid for all parts, regardless of the contents of StatusPool. It may contain PartStatus elements that define the node's status with respect to specific partitions. It is an error to define PartStatus elements that reference identical or overlapping parts within one StatusPool. Partitioned resources are described in Section 3.9.2 Description of Partitionable Resources.

---

[4] rRefs also enables spawning and merging if NodeInfo is extended with private ResourceRef elements.

Table 3-8 Contents of the StatusPool element

Name	Data Type	Description
Status ?	enumeration	Identifies the status of the node. The *Status* of individual partitions may be overwritten by PartStatus elements. Possible values are all valid *Status* attributes of a JDF node except "*Pool*" are valid as defined in Table 3-3 Contents of a JDF node, *Status*.
PartStatus *	element	Element that defines the node's status for a set of parts.

The following table describes the PartStatus element.

Table 3-9 Contents of the PartStatus element

Name	Data Type	Description
Status ?	enumeration	Identifies the status of an individual part of the node. Overwrites the Status attribute defined in StatusPool. Possible values are identical to those defined in: Status
Part [5] Modified in JDF 1.1	element	Specifies the selected part that the PartStatus is valid for. This must be a leaf partition of the resource.

3.7 Resources

Resources represent the "things" that are produced or consumed by processes. They may be physical items such as inks, plates, or glue; electronic items such as files or images; or conceptual items such as parameters and device settings. Processes describe what resources they input or output through ResourceLinks, discussed in Section 3.8 Resource Links. By examining the input and outputs of a set of processes, it is possible to determine process dependencies, and therefore job routing.

All resources are contained in the ResourcePool element of a node. The ResourcePool element is described in the following table.

Table 3-10 Contents of the ResourcePool element

Name	Data Type	Description
Resource *	element	List of Resource elements. The Resource elements are abstract and serve as placeholders for any resource type.

Like the *Type* attribute in abstract JDF nodes, the *Class* attribute in Resource elements helps to identify how particular resources should be used. This attribute contains seven values, and all resources fall under one of these seven classifications. For example, all resources whose *Class* = *Consumable* are physical resources that will be consumed over the course of the process. These values are listed in Table 3-11, below, and are described in greater detail in the sections that follow.

Table 3-11 Contents of the abstract Resource element

Name	Data Type	Description
CatalogID ?	string	Identification of the resource e.g. in a catalog environment. Defaults to the ProductID.
CatalogDetails ?	string	Additional details of a resource in a catalog environment.
Class	enumeration	Defines the abstract resource type. For details, see the sections that follow. Possible values are: *Consumable* *Handling*

---

[5] The cardinality of Part in PartStatus has been changed from * to none, e.g. exactly one element in version 1.1 of the JDF specification.

Name	Data Type	Description
		Implementation
		Intent
		Parameter
		PlaceHolder
		Quantity
ID	ID	Unique identifier of a resource.
Locked ?	boolean	If *true*, the resource is referenced by an *Audit* and cannot be modified without invalidating the *Audit*. Default = *false*
PipeID ?	string	If this attribute exists, the resource is a pipe. The PipeID is used by JMF pipe-control messages to identify the pipe. For more information, see Section 4.3.2 Overlapping Processing Using Pipes.
ProductID ?	string	An ID of the resource as defined in the MIS system.
rRefs ?	IDREFS	Array of *ID*s of internally referenced resources.
SettingsPolicy ? New in JDF 1.1	enumeration	The policy for this Resource indicates what happens when unsupported settings, i.e., subelements, attributes, or attribute values, are present. A JDF Consumer must support this attribute and all of the defined values so that an Agent can depend on the JDF Consumer following the policy requested by the Agent in a JDF instance. Possible values are: *BestEffort* = Substitute or ignore unsupported attributes, attribute values, default attribute values, or elements and continue processing the job. *MustHonor* = Reject the job when (1) any unsupported attributes, attribute values, or elements are present or (2) any omitted attributes have an unsupported default value defined in this specification. *OperatorIntervention* = Pause the job and query the operator when (1) any unsupported attributes, attribute values, or elements are present or (2) any omitted attributes have an unsupported default value defined in this specification. If a device has no operator intervention capabilities, *OperatorIntervention* is treated as *MustHonor*. If not specified, the value defined for the node that this resource resides in is used.
SpawnIDs ? New in JDF 1.1	NMTOKENS	List of SpawnIDs. This is used as a reference count how often the resource has been spawned.
SpawnStatus ?	enumeration	The spawn status of a node indicates whether or not a node has been spawned, and under what circumstances. The list is assumed to be ordered, so that the *SpawnStatus* of a resource that has *rRefs* entries is defined as the maximum *SpawnStatus* of all recursively linked resources. Possible values are: *NotSpawned* – Default value. Indicates that the resource has not been copied to another process. *SpawnedRO* – Indicates that the resource has been copied to another process where it cannot be modified. RO stands for read-only. *SpawnedRW* – Indicates that the resource has been copied to another process where it can be modified. RW stands for read/write.
Status Modified in JDF 1.1	enumeration	The status of a node indicates under what circumstances it may be processed or modified. The list is assumed to be ordered, so that the *Status* of a resource that has *rRefs* entries is defined as the minimum *Status* of all recursively linked resources. Possible values are:

Name	Data Type	Description
		Incomplete – Indicates that the resource does not exist, and the metadata is not yet valid.
		Unavailable – Indicates that the resource is not ready to be used or that the resource in the real world represented by the physical resource in JDF is not available for processing. The metadata is valid.
		InUse – Indicates that the resource exists, but is in use by another process. Also used for active pipes (see Sections 3.7.3 and 4.3.2).
		Draft – Indicates that the resource exists in a state that is sufficient for setting up the next process but not for production.
		Complete – Indicates that the resource is completely specified and the parameters are valid for usage. A physical resource with **Status** = *Complete* is not yet available for production, although it is sufficiently specified for a process that references it through a ResourceRef from a parameter resource to commence execution.
		Available – Indicates that the whole resource is available for usage.
UpdateID ? New in JDF 1.1	NMTOKEN	Unique ID that identifies the Resource or Resource partition. Note that only one Resource, Resource partition or ResourceUpdate with a given value of *UpdateID* may occur per JDF document, even though the scope of the ResourceUpdate is local to the resource that it is defined in.

Figure 3.4 shows the structure of the abstract resource classes defined above. Arrows define inheritance relations and the thin orthogonal lines describe containing relations.
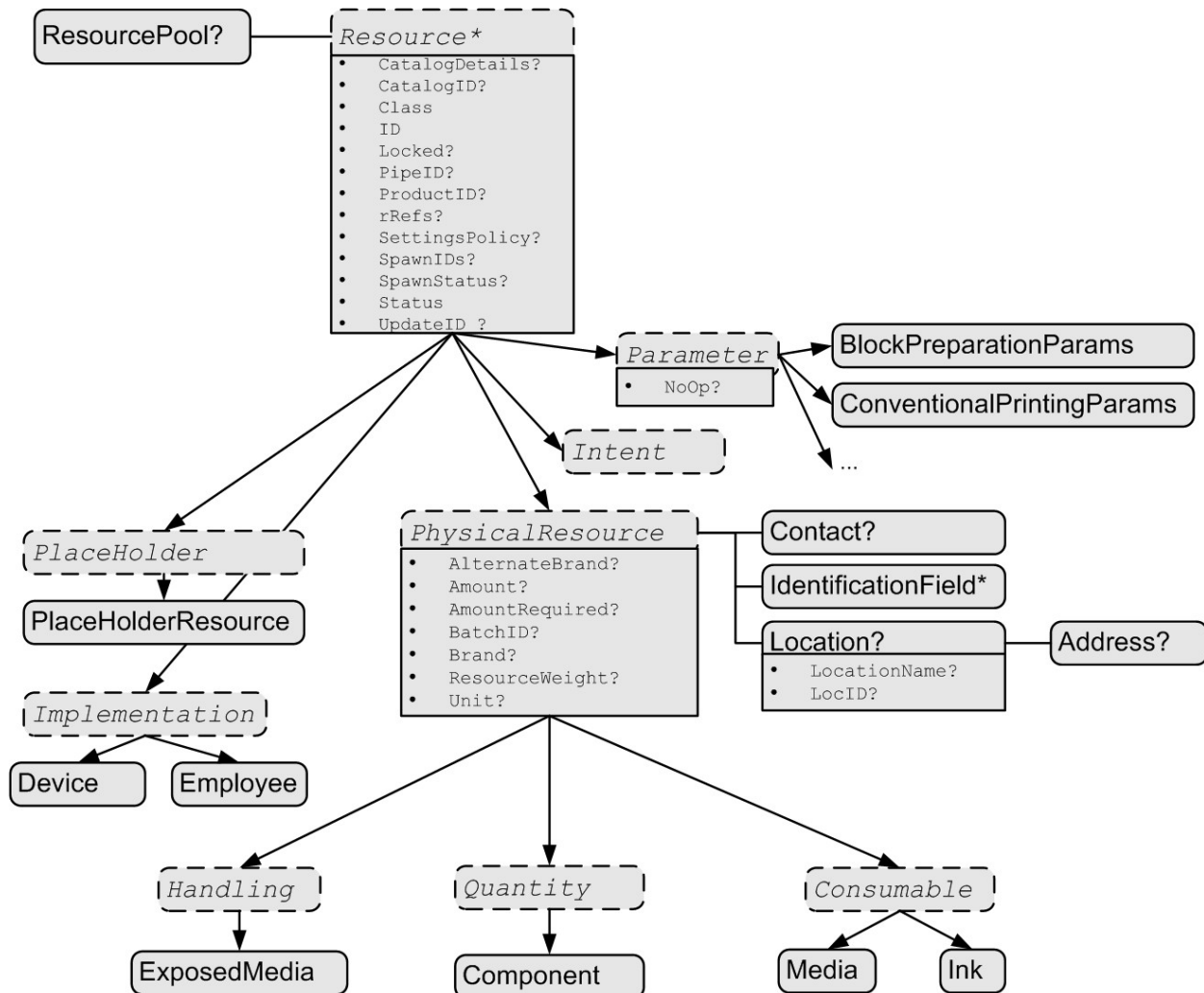
Figure 3.4 Structure of the abstract resource types

3.7.1 Resource Classes

The following sections describe the functions of each of the seven values of the **Class** attribute. All resources fall into one of these classes. In Chapter 7 **Resources**, the class of each resource is indicated in the Resource Properties subheading.

3.7.1.1 Parameter Resources

Parameter resources define the details of processes, as well as any non-physical computer data such as files used by a process. They are usually associated with a specific process. For example, a required input resource of the *ColorSpaceConversion* process is the ColorSpaceConversionParams resource. All predefined parameter resources contain the moniker "Params" in their titles. Other examples of *Parameter* resources include **AdhesiveBindingParams** and **ConventionalPrintingParams**.

> **Parameter & Intent Resources**
>
> Parameter and Intent Resources are *information* about the print job. Intent resources may originate in the customer's RFQ and may include information such as trim size, paper, the number of colors, and so on. Later on in the process of estimating and scheduling the job, these intents may become parameters for production process.

Table 3-12 Additional contents of the abstract parameter Resource elements

Name	Data Type	Description
NoOp ? New in JDF 1.1	boolean	Indicates whether a resource or resource partition should be treated as if it did not exist, e.g., to switch off a complete process step for the process that requires the given parameter resource or partition as input. Default = *false*, i.e., the Resource is operational and should be honored.

3.7.1.2 Intent Resources

Intent resources define the details of products to be produced without defining the process to produce them. In addition, they provide structures to define sets of allowable options and to match these selections with prices. The details of all intent resources are described in Section 7.1 **Intent Resources**. The abstract *Intent* resource element contains no attributes or elements besides those contained in the abstract Resource element.

3.7.1.3 Implementation Resources

Implementation resources define the devices and operators that execute a given node. Only two implementation resource types are defined: Employee (see Section 7.2.49) and Device, each of which is described in greater detail in the Chapter 7.

Implementation resources can only be used as input resources and may be linked to any process. The abstract *Implementation* resource element contains no attributes or elements besides those contained in the abstract Resource element. An example demonstrating how to use implementation resources is provided in Section 3.8.2 Links to Implementation Resources.

Note that it is not recommended to specify the capabilities of a **Device** that is linked to a process to specify that it should execute the given process.

3.7.1.4 Physical Resources (Consumable, Quantity, Handling)

Any resource whose *Class* is *Consumable*, *Quantity*, or *Handling* is considered a physical resource. They are defined as follows:

- *Consumable* resources are resources that are consumed during a process. Examples include **Ink** and Media. They are the unmodified inputs in a process chain.

- *Quantity* resources are resources that have been created by a process from either a *Consumable* resource or an earlier *Quantity* resource. For example, printed sheets are cut and a pile of cut blocks is created. **Component** resources are an example of *Quantity* resources.

- A *Handling* resource is used during a process, but is not destroyed by that process. ExposedMedia and Tool are examples of such a resource, although it does describe various kinds of items such as film and plates. A *Handling* resource may be created from a *Consumable* resource.

Automating Inventory Management

JDF's handling of physical resources provides a bridge between your JDF enabled systems and inventory management, ordering and replenishing systems. This opens the door to just-in-time inventory management driven by real-time scheduling and consumption data.

Table 3-13 Additional contents of the abstract physical Resource elements defines the additional attributes and elements that may be defined for physical resources. The processes that consume physical resources—any kind of physical resource—have the option of using these attributes and elements to determine in what way the resources should be consumed. Table 3-13 Additional contents of the abstract physical Resource elements then describes the contents of the Location subelement of physical resource elements.

Table 3-13 Additional contents of the abstract physical Resource elements

Name	Data Type	Description
AlternateBrand ?	string	Information, such as the manufacturer or type, about a resource compatible to that specified by the *Brand* attribute, which is described below.

Name	Data Type	Description
Amount ?	number	Actual amount of the resource that is available.
		Note that the amount of consumption and production of a node is specified in the corresponding resource links.
AmountRequired ?	number	Total amount of the resource that is referenced by all nodes that will consume this resource. This corresponds to the sum of all *Amount* values of input resource links that reference this resource.
BatchID ?	string	ID of a specific batch of the physical resource
Brand ?	string	Information, such as the manufacturer or type, about the resource being used.
ResourceWeight ? New in JDF 1.1	double	Weight of a single component of the resource in grams.
Unit ?	NMTOKEN	Unit of measurement for the values of *Amount* and *AmountRequired*. Note that it is strongly discouraged to specify units other than those that are defined in Units
Weight ? Illegal in 1.1	double	Weight of a single component of the resource in grams. This parameter collides with Media::Weight and is therefore illegal and has been replaced with *ResourceWeight* in version 1.1 and beyond.
Contact ?	refelement	If this element is specified, it describes the owner of the resource.
IdentificationField * New in JDF 1.1	refelement	If this element is specified, a bar code or label is associated with this physical resource.
Location ?	refelement	Description of details of the resource location.
		Note, in order to describe multiple locations, resources may be partitioned by the *Location*-key as described in Section 3.9.2 Description of Partitionable Resources.

Structure of Location Subelement

Table 3-14 Contents of the Location element

Name	Data Type	Description
LocationName ? New in JDF 1.1	string	Name of the location, e.g., for example in MIS. This part key allows to describe distributed resources.
LocID ?	string	Location identifier, e.g., within a warehouse system.
Address ?	refelement	Address of the storage facility. For more information, see Section **7.2.2**.

3.7.1.5 PlaceHolder Resources

PlaceHolder resources, unlike physical resources, do not describe any logical or physical entity. Rather, they define process linking and help to define process ordering when the exact nature of interchange resources is still unknown. In essence, they serve as placeholders that stand in for defined resources. Using *PlaceHolder* resources, a processing skeleton can be constructed that gives a basic shape to a job. The appropriate resources can be substituted for *PlaceHolder* resources when they become known.

This kind of resource should only be used to link nodes of *Type* = *ProcessGroup*, since process leaf nodes have well defined resources that should be used in preference. The only resource whose *Class* = *PlaceHolder* is called **PlaceHolderResource**.

Like *Parameter* and *Implementation* resources, *PlaceHolder* resources contain no attributes besides those contained in the abstract Resource element.

3.7.1.6   Selector Resources

Removed in JDF 1.1

Resources of class *Selector* have been removed in JDF version 1.1 and higher. Note that they are not only depre-cated but actually removed from the format including the schema and must not be supported by a JDF 1.1 conform-ing agent

3.7.2   Position of Resources within JDF Nodes

Resources may exist in any JDF node, but JDF nodes may only reference local or global resources. In other words, JDF nodes may only reference resources in the two kinds of locations: in the node's own **ResourcePool** element or in JDF nodes that are hierarchically closer to the JDF root. An exception to this rule, however, occurs if two in-dependent jobs are merged for a process step and are to be separated afterwards, as is the case when two independ-ent jobs are printed on the same web-fed press. For further details on independent job merging, see Section 4.4.5 Case 5: Spawning and Merging of Independent Jobs.

It is good practice to put resources into the highest-level node that references the resource. For example, the **RenderingParams** resource should be located in the *Rendering* node, unless it is used by multiple *Rendering* processes, in which case it should be located in the ***ProcessGroup*** node that contains the ***Rendering*** process nodes. Resources that link more than one node should be placed in the parent node of the siblings that are linked by the resource.

A process that needs additional detailed process information specifying the creation of a resource must infer this information by explicitly linking to the appropriate parameter resource.

3.7.3   Pipe Resources

A Pipe describes the resource dependency in which a process begins to consume a resource while it is being pro-duced by another process. For example, stacking components while they are being printed, or consuming a data stream while it is being written by an upstream process.

Using dynamic pipe control, a downstream process may control the total quantity produced by an upstream process, and/or the quantity buffered by an inter-process transport device (i.e. Conveyor belt.) Additional description of pipes and process communication via pipes is provided in Section 4.3.2 Overlapping Processing Using Pipes.

Resources may contain a string attribute called *PipeID* that declares the resource to be a pipe, and identifies it in a dynamic-pipe messaging environment. A pipe that is also controlled by JMF pipe messages is called **dynamic pipe**. For more information about dynamic pipes, see Section 4.3.2.2 Dynamic Pipes.

3.7.4   ResourceUpdate Elements

New in JDF 1.1

ResourceUpdate elements are an abstract element class that optionally contains any of the attributes and elements valid for the **Resource** that they reside in. Required attributes and elements of resources are optional in the respec-tive ResourceUpdate. In addition, a ResourceUpdate defined within a **Resource** must contain a unique *Up-dateID* of type NMTOKEN. Only devices that process the resource as input can reference the *UpdateID* of a ResourceUpdate. Such references to ResourceUpdate elements must update the current state of the device.

When a ResourceUpdate is referenced from a device, e.g., from a PPML TicketRef element, said device will update ONLY those elements that are explicitly specified within the ResourceUpdate. No attributes are inherited from the **Resource** that contains the ResourceUpdate.

ResourceUpdate elements are useful for process input resources only and must not be applied to product in-tent resources.

Table 3-15 Contents of the abstract ResourceUpdate Element

Name	Data Type	Description
UpdateID New in JDF 1.1	NMTOKEN	Unique ID that identifies the ResourceUpdate. Note that only one Resource, Resource partition or ResourceUp-date with a given value of *UpdateID* may occur per JDF document, even though the scope of the ResourceUpdate is local to the resource that it is defined in.

Example:
The following example shows ResourceUpdate elements in highlight.

```
<JDF xmlns="http://www.CIP4.org/JDFSchema_1_1" ID="MyCombinedProcessNode" Status="Ready"
Type="Combined"
Types="Interpreting Rendering DigitalPrinting" Version="1.1">

<ResourceLinkPool>
  <InterpretingParamsLink rRef="PDFIParams" Usage="Input" CombinedProcessIndex="0"/>
  <RenderingParamsLink rRef="RParams" Usage="Input" CombinedProcessIndex="1"/>
  <DigitalPrintingParamsLink rRef="DPParams" Usage="Input" CombinedProcessIndex="2"/>
. . .
</ResourceLinkPool>

<ResourcePool>
  <Media ID="White" … />
  <InterpretingParams ID="PDFIParams" Class="Parameter" Status="Available" PrintQuality="High"
Polarity="Positive" EmitPDFTransfers="false" UpdateID="SetPrintQualityDefault"/>
    <InterpretingParamsUpdate UpdateID="SetNegativePolarity" Polarity="Negative"/>
    <InterpretingParamsUpdate UpdateID="SetPositivePolarity" Polarity="Positive"/>
    <InterpretingParamsUpdate UpdateID="SetPrintQualityDraft" PrintQuality="Draft"/>
    <InterpretingParamsUpdate UpdateID="SetPrintQualityNormal" PrintQuality="Normal"/>
    <InterpretingParamsUpdate UpdateID="SetPrintQualityHigh" PrintQuality="High"/>
  </PDFInterpretingParams>
  <RenderingParams ID="RParams" Class="Parameter" Status="Available">
    <AutomatedOverprintParams OverPrintBlackText="true" OverPrintBlackLineArt="true"/>
  </RenderingParams>
  <DigitalPrintingParams ID="DPParams" Class="Parameter" Status="Available" PrintingType="Sheet">
    <MediaRef rRef="White" MediaLocation="WhiteTray" UpdateID="SetMediaDefault"/>
    <DigitalPrintingParamsUpdate UpdateID="SetMediaYellow"/>
      <Media ID="Yellow" MediaLocation="YellowTray" />
    </DigitalPrintingParamsUpdate>
  </DigitalPrintingParams>
. . .
</ResourcePool>

</JDF>
```

3.8 Resource Links

ResourceLinks describe what resources a node uses, and how it uses them. They also allow node dependencies to be calculated. The following diagram summarizes resource linking within a JDF node.   In this example there are two resources, A and B, which are placed in the node's ResourcePool. To reference the resources, the node has two re-source links, ALink and BLink, in the ResourceLinkPool. The resource links are named by appending "Link" to the type of resource referenced. Resource B also contains a reference to resource A, called ARef.   References to re-sources from within resources are named by appending "Ref" to the type of resource referenced.
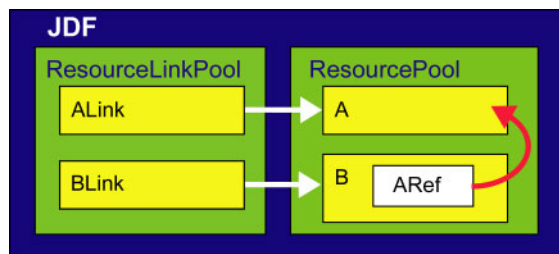
Figure 3.5 Resource Links and ResourceRefs

The previous section described resources used by the node in which it resides. This section describes how resources may serve as links between nodes. As was described in Section 2.2 JDF Workflow, any resource that is the output of one process will very likely serve as an input of a subsequent resource. Furthermore, some resources are shared between ancestor nodes and their child nodes.

Each JDF node contains a ResourceLinkPool element that in turn contains all of the ResourceLink elements that link the node to the resources it uses. They also define whether the resources are inputs or outputs. These inputs and outputs provide conceptual links between the execution elements of JDF nodes. Outputs of one node may in turn become inputs in another node, and a given node must not be executed before all required input resources are available.[6] Figure 3.6 shows two processes that are linked by a resource. The resource represents the output of Node 1, which in turn becomes an input for Node 2.
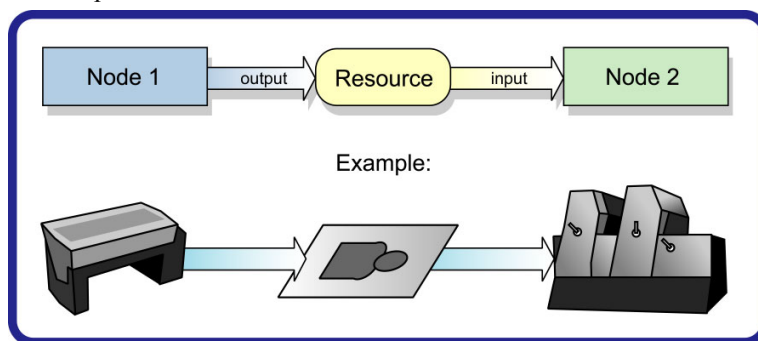
Figure 3.6 Nodes linked by a resource

ResourceLink elements may also contain optional attributes to select a part of a resource, such as a single separation. A detailed description of resource partitioning is given in Section 3.9.2 Description of Partitionable Resources.

ProcessGroup and *Product* nodes may be defined without the knowledge of the individual process nodes that define a specific workflow. In this case, these intermediate nodes will contain ResourceLink elements that link the appropriate resources. For example, a prepress node may be defined that produces a set of plates. When the processes for creating the plates are defined in detail, the agent that writes the nodes may remove the ResourceLink elements from the intermediate node. Removing the ResourceLink specifies that the intermediate node may execute; that is, it may be sent to the appropriate controller or department, even though the specific resources are not yet available. If the ResourceLinks are not removed, the intermediate node must not execute until the input resources that are linked are available.

Resource links may be used for process control. For example, if a proof input resource is required for a print process, a print run may only commence when the proof is signed. The JDF format specification also includes a complete specification of how resources are managed when JDF tickets are spawned and merged.

In some cases, determining whether information should be stored in an input or an output resource may be difficult, as the distinction can be ambiguous. For example, is the definition of the color of a separation in the RIP process a property of the output separation or a parameter that describes the RIP process? In order to reduce this

---

[6] The availability of a resource that is consumed as a whole is given by the Resource attribute *Status* = *Available*. In the case of pipe resources, the availability depends on the individual parameter defining the dynamics of a pipe (for details see Section 4.3.2 Overlapping Processing Using Pipes).

ambiguity, the following rules have been applied for the definition of input and output resources of processes as described in Chapter 6 **Processes** and Chapter 7 **Resources**:

- Product intent and process parameters are generally input resources, except when one process defines the parameters of a subsequent process.
- *Consumable* resources are always input resources.
- *Quantity* and *Handling* resources are used both as input and output resources. Their usage is defined by the "natural" process usage. For example, a printing plate is described as an ExposedMedia resource that is the output of a *ImageSetting* process and the input of a *ConventionalPrinting* process.
- Printed material is exchanged from node to node using the **Component** resource. Product intent nodes also create **Component** output resources.
- Every detailed process description must be defined as an input parameter of the first process where it is referenced. This means that a device must not imply process parameters from its output resources. For example, paper grammage MAY be defined in the **Component** output resource of the printing process but MUST be defined as an input parameter of the **Media** of the printing process.
- Any resource parameter that is used must be referenced explicitly. Resource parameters cannot be inferred by following the chain of nodes backwards. This would make spawning of nodes non-local.
- The last process in a chain of processes defines the output resource of its parent process.
- In case of parallel processing, the sum of the outputs of all parallel subnodes defines the output of the parent node.
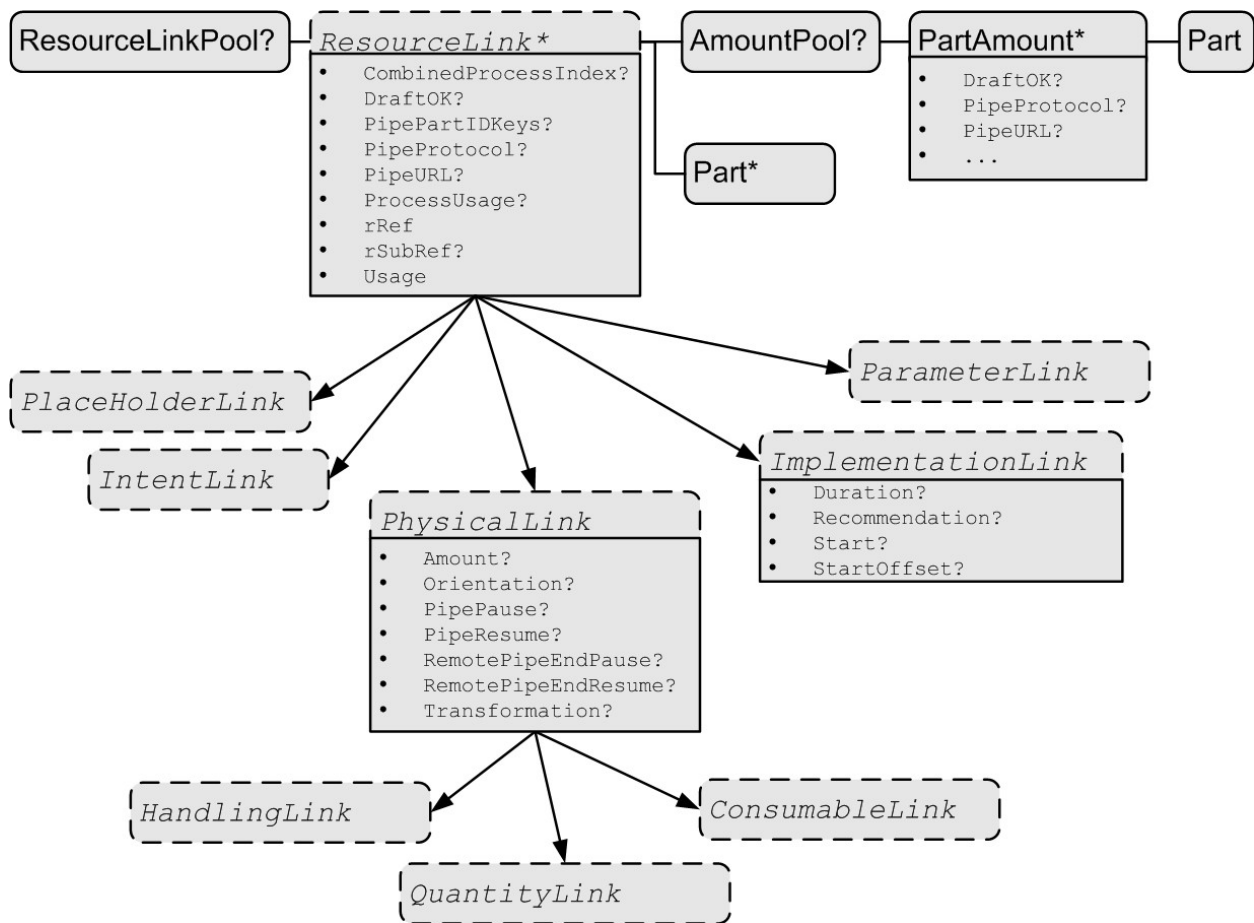
Figure 3.7 Structure of the abstract ResourceLink types

Like Resource elements, ResourceLink elements are an abstract data type. The class tree of abstract ResourceLink elements is further subdivided into classes defined by the *Class* attribute of the resource that it references.

Individual instances of **ResourceLink** elements are named by appending the suffix "Link" to the name of the referenced resource. For example the link to a **Component** resource is entitled ComponentLink and the link to a ScanParams resource is entitled ScanParamsLink. The following eight abstract resource link classes exist:

- ParameterLink
- ImplementationLink
- ConsumableLink
- QuantityLink

- HandlingLink
- PlaceHolderLink
- IntentLink

Each listed class name is described in greater detail in the sections that follow. The following figure shows the abstract resource link types derived from the abstract ResourceLink type.

The following table lists the contents of a ResourceLinkPool element.

Table 3-16 Contents of the ResourceLinkPool element

Name	Data Type	Description
ResourceLink *	element	List of ResourceLink elements. The ResourceLink elements are abstract and are a placeholder for any resource link element.

The following table lists the possible contents of all ResourceLink elements.

Table 3-17 Contents of the abstract ResourceLink element

Name	Data Type	Description
CombinedProcessIndex ? New in JDF 1.1	IntegerList	*Combined* nodes contain input resources from multiple process nodes. The *CombinedProcessIndex* attribute specifies the indices of individual processes in the *Types* attribute to which a ResourceLink in a *Combined* node belongs. Multiple entries in *CombinedProcessIndex* specify that the *ResourceLink* is used by the respective multiple processes in the *Combined* node.
CombinedProcessType ? Deprecated in JDF 1.1	NMTOKEN	*Combined* nodes contain input resources from multiple process nodes. The *CombinedProcessType* attribute specifies the name individual process to which a ResourceLink in a *Combined* node belongs. Must match one of the entries in the *Types* attribute of the node. Replaced by *CombinedProcessIndex* in JDF 1.1.
DraftOK ?	boolean	If true, the process may commence with a draft resource. Default = *false*
PipePartIDKeys ?	enumerations	Defines the granularity of a dynamic pipe for a partitioned resource. For instance, a resource may be partitioned by sheet, surface and separation (resource attribute *PartIDKeys* = *SheetName Side Separation*), but pipe requests should only be issued once per surface (resource link attribute *PipePartIDKeys* = *SheetName Side*). The contents of *PipePartIDKeys* must be a subset of the *PartIDKeys* attribute of the resource that is linked by this *ResourceLink*. If *PipePartIDKeys* is not specified, it defaults to *PartIDKeys*, i.e. maximum granularity. For details on partitioned resources, see Section 3.9.2.
PipeProtocol ? New in JDF 1.1	NMTOKEN	Defines the protocol use for pipe handling. *JMF* is the only non-proprietary piping protocol that is supported. Proprietary pipe protocols may be specified in addition to those defined below but will not necessarily be interoperable. Allowed values include: *JMF* – JMF based PipePush / PipePull messages. *None* – No pipe support. If *PipeURL* is specified and *PipeProtocol* is not specified, *JMF* is assumed.

Name	Data Type	Description
PipeURL ?	URL	Pipe request URL. Dynamic pipe requests from this end of a pipe should be made <u>to</u> this URL.[1] Note that this URL is only used for initiating pipe requests. Responses to a pipe request are issued to the URL that is defined in the PipePush or PipePull message. For details on using *PipeURL*, see Section 4.3.2.
ProcessUsage ?	string	Identifies the resource usage in the process if multiple resources of the same type are required. For example, this attribute appears when two components—one Cover and one BookBlock—are used in **Adhesive-Binding**. The allowed values of *ProcessUsage* are defined in the appropriate process descriptions in Chapter 6 **Processes**.
rRef	IDREF	Link to the target resource.
rSubRef ?	IDREF	Link to a subelement within the resource.
Usage	enumeration	Resource usage within this JDF node. Possible values are: *Input* – The resource is an input. *Output* – The resource is an output.
AmountPool ? New in JDF 1.1	element	Definition of partial amounts and pipe parameters for this ResourceLink. The allowed contents of the AmountPool are described for the various types of resource links in the sections below.
Part *	element	The Part elements identify the parts of a partitioned resource that are referenced by the ResourceLink. The structure of the Part element is defined in Table 3-25 Contents of the Part element. For details on partitioned resources, see Section 3.9.2.

The following table lists the generic contents of an AmountPool element. Further parameters of the AmountPool are described in the sections below.

Table 3-18 Contents of the AmountPool element

Name	Data Type	Description
PartAmount * New in JDF 1.1	element	Element that defines the amounts and pipe parameters for a partitioned resource. The contents of a PartAmount depends on the type of the ResourceLink.

The following table lists the generic contents of a PartAmount element. Further parameters of the PartAmount are described in the respective sections below (Table 3-20 Contents of the abstract ImplementationLink or PartAmount element and Table 3-21 Additional contents of the abstract physical ResourceLink and PartAmount or AmountPool element). Note that PartAmount inherits values from its parent ResourceLink.

---

[1] Note that in most cases this is the URL of the controller of the *other end* of the pipe. This may seem counterintuitive, but it allows parallel spawning and merging of processes that represent a dynamic pipe without having to include the node that describes the other end in the spawned file.

Table 3-19 General contents of the PartAmount element

Name	Data Type	Description
DraftOK ? New in JDF 1.1	boolean	If *true*, the process may commence with a draft resource partition.
PipeURL ? New in JDF 1.1	URL	Pipe request URL for this partition. Dynamic pipe requests from this end of a pipe should be made to this URL.[2] Note that this URL is only used for initiating pipe requests. Responses to a pipe request are is-sued to the URL that is defined in the PipePush or PipePull message. For details on using *PipeURL*, see Section 4.3.2.
Part New in JDF 1.1	element	Specifies the selected part that the PartAmount is valid for. This must be a leaf partition of the resource.

3.8.1 Links to Parameter Resources

Parameter resources are linked by an instance of a ParameterLink element. These elements contain no further at-tributes or elements besides those found in the abstract ResourceLink element.

3.8.2 Links to Implementation Resources

Implementation resources are linked by an instance of an ImplementationLink element. Using the resource attrib-utes, the link may specify whether the implementation is a recommendation that may be ignored or a request that must be fulfilled. For example, the job may contain a request that the job be run by a specific, experienced operator. If the value or the *Recommendation* is *true* and that operator is ill, he may be replaced by a less experienced operator. If, on the other hand, a product could be created on a device that theoretically can do the job but does not produce sufficient quality, and if it is certain that customer will reject inferior quality, *Recommendation* should be set to *false*.

Since implementation ResourceLinks define the usage of a specific device during the course of a job, situa-tions can arise where that resource is not required during the whole processing time. For instance, a forklift that only has to transport the completed components is not required to be available during the entire process run, only during the times when it is needed. This means that, contrary to the general rule that all resources must be *Available* for node execution to commence, a node may commence when implementation resources are still *InUse* by other processes if *Start* or *StartOffset* are specified. ImplementationLink elements always have a *Usage* of *Input*.

Table 3-20 Contents of the abstract ImplementationLink or PartAmount element

Name	Data Type	Description
Duration ?	duration	Estimated duration during which the resource will be used.
Recommendation ?	boolean	If *true* and the request cannot be fulfilled, the change may be logged as a Modified Audit and the job may continue. If *false*, an error occurs if the request is not fulfilled. Default = *false*
Start ?	dateTime	Time and date when the usage of the implementation resource starts.
StartOffset ?	duration	Offset time when the resource is required after processing has begun. If both *Start* and *StartOffset* are specified, *Start* has precedence.

The following example shows how the operator Smith is linked to a ConventionalPrinting process as the only valid operator:

```
<ResourcePool>
```

---

[2] Note that in most cases this is the URL of the controller of the *other end* of the pipe. This may seem counterintui-tive, but it allows parallel spawning and merging of processes that represent a dynamic pipe without having to in-clude the node that describes the other end in the spawned file.

```
   <Employee PersonalID="007" ID="L1" Class="Implementation">
     <Person FamilyName="Smith" JobTitle="Press Operator">
   </Employee>
</ResourcePool>
…
<ResourceLinkPool>
  <EmployeeLink Recommendation="false" Usage="Input" rRef="L1"/>
</ResourceLinkPool>
```

3.8.3 Links to Physical Resources

The physical resources that fall into the *Consumable*, *Quantity*, and *Handling* classes are linked, predictably, by the appropriate instances of ConsumableLink, QuantityLink, or HandlingLink resource link elements. Just as physical resources inherit the contents of the abstract resource element, physical resource links inherit the contents of the abstract resource link element. They may, however, contain additional contents. These optional attributes are described in Table 3-21, below. The attributes in Table 3-21 may occur either directly in the physical ResourceLink or in AmountPool and PartAmount elements of a resource link.

It is important to note that the order of occurrence of links to physical resources may be significant – most specifically with QuantityLinks. For example, a **Gathering** process might have among its inputs, links to three component resources. The order of these links indicates the order in which the components should occur in the new, gathered output component.

Table 3-21 Additional contents of the abstract physical ResourceLink and PartAmount or AmountPool element

Name	Data Type	Description
Amount ?	number	For a link with a Usage of 'Input', specifies the amount of the resource that is required by the process, in units as defined in the resource.
		For a link with a Usage of 'Output', specifies the amount of the resource that is to be produced by the process, in units as defined in the resource.
		Allows resources to be only partially consumed or produced (see Section 3.9.1 Resource Amount).
Orientation ? New in JDF 1.1	enumeration	Named orientation describing the transformation of the orientation of a physical resource relative to the ideal process coordinate using this resource as input or output. Allowed values are:
		Rotate0:
		Rotate90:
		Rotate180:
		Rotate270:
		Flip0:
		Flip90:
		Flip180:
		Flip270:
		For details, of the semantics of the enumeration, see Table 2-3. This is needed to convert the coordinate system of the resource to the coordinate system of the process. Agents should supply one of *Orientation* or *Transformation* for resources where they are relevant, e.g., **Component**. When neither *Orientation* or *Transformation* are present, the orientation of the resource is system specified.
		If *Orientation* is specified for an output resource, the node that processes the physical resource should manipulate the resource in such a way as to reflect the transformation. The coordinate system of the resource itself is NOT modified. Only one of *Orientation* or *Transformation* may be specified in one ResourceLink.

Name	Data Type	Description
PipePause ?	number	Parameter for controlling the pausing of a process if the resource amount in the pipe buffer passes the specified value. For details on using *PipePause*, see Section 4.3.2.
PipeResume ?	number	Parameter for controlling the resumption of a process if the resource amount in the pipe buffer passes the specified value. For details on using *PipeResume*, see Section 4.3.2.
RemotePipeEnd-Pause ?	number	Parameter for controlling the pausing of a process at the other end of the pipe if the resource amount in the pipe buffer passes the specified value. For details on using *RemotePipeEndPause*, see Section 4.3.2.
RemotePipeEn-dResume ?	number	Parameter for controlling the resumption of a process at the other end of the pipe if the resource amount in the pipe buffer passes the specified value. For details on using *RemotePipeEndResume*, see Section 4.3.2.
Transformation ? New in JDF 1.1	matrix	Matrix describing the transformation of the orientation of a physical resource relative to the ideal process coordinate using this resource as input or output. This is needed to convert the coordinate system of the resource to the coordinate system of the process. Agents should supply one of *Orientation* or *Transformation* for resources where they are relevant, e.g., **Component**. When neither *Orientation* or *Transformation* are present, the orientation of the resource is system specified. If *Transformation* is specified for an output resource, the node that processes the physical resource should manipulate the resource in such a way as to reflect the transformation. The coordinate system of the resource itself is NOT modified.

The following example shows an InkLink with an AmountPool.

```
<ResourcePool>
  <Ink ID="Link0015" Brand="NoName" Class="Consumable" Locked="false"
Status="Available" PartIDKeys="Separation">
    <Ink ColorName="Cyan" Separation="Cyan"/>
    <Ink ColorName="Magenta" Separation="Magenta"/>
    <Ink ColorName="Yellow" Separation="Yellow"/>
    <Ink ColorName="Black" Separation="Black"/>
    <Ink ColorName="Heidelberg Spot Blau" Separation="Heidelberg Spot Blau"/>
  </Ink>
</ResourcePool>
<ResourceLinkPool>
  <InkLink rRef="Link0015" Usage="Input">
    <AmountPool>
      <PartAmount Amount="1000">
        <Part Separation="Cyan"/>
      </PartAmount>
      <PartAmount Amount="1200">
        <Part Separation="Magenta"/>
      </PartAmount>
      <PartAmount Amount="700">
        <Part Separation="Yellow"/>
      </PartAmount>
      <PartAmount Amount="3000">
        <Part Separation="Black"/>
      </PartAmount>
      <PartAmount Amount="300">
        <Part Separation="Heidelberg Spot Blau"/>
      </PartAmount>
    </AmountPool>
```

```
   </InkLink>
</ResourceLinkPool>
```

3.8.4 Links to PlaceHolder Resources

PlaceHolder resources are linked by a PlaceHolderLink element. *PlaceHolder* links, used together with the **PlaceHolderResource** resource, can be employed to predefine a skeleton of a processing network consisting of process group nodes without knowing the exact nature of the interchange resources. For instance, although the deadlines for the job may be known, it may not be known whether a press run will be defined for a digital press or a conventional press.

3.8.5 Links to Intent Resources

Intent resources are linked by an instance of a IntentLink element. They have no additional parameters.

3.8.6 Inter-Resource Linking Using ResourceRef

In some cases, it is necessary to reference resource elements directly from other resources in order to reuse information. These links are abstract ResourceRef elements. The ResourceRef's name is generated by appending the string "Ref" to the element name. Candidate elements for inter-resource linking have a data type of refelement in the content description tables of this chapter and **Chapter** 7. The following table defines the attributes of the abstract ResourceRef element (see also Figure 3.4 and ResourceElement in Table 3-11). The ResourceElement is defined in Table 3-22 Contents of the abstract ResourceElement

Table 3-22 Contents of the abstract ResourceElement

Name	Data Type	Description
ID ?	ID	Unique identifier of a resource element.

Table 3-23 Contents of the abstract ResourceRef element

Name	Data Type	Description
rRef	IDREF	Reference to the resource.
rSubRef ?	IDREF	Reference to a subelement of the resource.
Part ? New in JDF 1.1	element	Definition of the partition that this ResourceRef references. This must be a leaf partition of the resource.

In order to enable spawning and merging without having to scan every single resource, inter-resource links must be specified in the *rRefs* attribute of the resource. In the case of a link to a resource subset, the *rRefs* attribute contains a reference to the atomic resource. Even if a resource is linked more than once, one occurrence of that resource in the *rRefs* array is sufficient.

The Part element in a ResourceRef defines the part of the target that this ResourceRef references. If both the resource that contains ResourceRef element and the target resource are partitioned, the ResourceRef does NOT implicitly reference the part of the target with the same partitioning attributes, but rather the parts of the target resource that are explicitly specified by the Part element within the ResourceRef.

```
<MediaRef  rRef="MediaID">
  <Part Location="desk"/>
</MediaRef>
```

ResourceRef elements may also occur in the NodeInfo and CustomerInfo element of a JDF node.

Elements within a resource, i.e. not direct children of the ResourcePool, may also contain an *ID* attribute (see Table 3-22 Contents of the abstract ResourceElement). These elements are denoted as ResourceElement. These elements may be explicitly referenced by a ResourceRef. The ResourceRef element has an optional *rSubRef* attribute that contains an IDREF to the ID of the ResourceElement within the resource.

In some cases, it is desirable to define a ResourceElement that is not explicitly linked by a Node directly within a ResourcePool as a Resource. These Resources are referenced only by other resources which contain ResourceRef elements pointing to these. The ResourceElements instantiated as a Resource must contain the required attributes of abstract resources and have a *Class*="*Parameter*". The following example demonstrates inter-resource linking.

```
<ResourcePool>
  <Layout rRefs="res1 res2"><!—This is a Resource-->
  …
    <!—These are ResourceRefs-->
    <SurfaceRef rRef="res1" rSubRef="surf1"/>
    <SurfaceRef rRef="res2" rSubRef="surf2"/>
    <SurfaceRef rRef="res1" rSubRef="surf1"/>
<!-- another link to the same resource -->
  </Layout>
  <Sheet ID="res1"><!—This is a Resource-->
    <Surface ID="surf1" … /> <!—This is a ResourceElement-->
  </Sheet>
  <Sheet ID="res2"> <!—This is a Resource-->
    <Surface ID="surf2" … /> <!—This is a ResourceElement-->
  </Sheet>
</ResourcePool>
```

3.8.6.1 Status of Resources That Contain rRef References

The *Status* of a resource that contains an rRef attribute is defined by the lowest *Status* of all recursively referenced resources. The ordering is defined as:

Incomplete	*Draft*
Unavailable	*Complete*
InUse	*Available*

Thus, if any referenced resource has a *Status* of *Incomplete*, the complete resource has a calculated *Status* of *Incomplete*, even though its own *Status* attribute may be *Unavailable, Draft, Available* etc.

3.8.6.2 Alignment of ResourceLink and ResourceRef

New in JDF 1.1A

ResourceRef elements must not contain any of the attributes and elements that may be specified in the ResourceLink as defined in chapter 3.8 Resource Links. The value of these properties is implied from the value of the properties for the appropriate part in the AmountPool of the ResourceLink of the node. The following example illustrates the alignment of a MediaLink and MediaRef in a *DigitalPrinting* node.

```
<JDF ID="n20020626134204" Type="DigitalPrinting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Waiting" Version="1.1">
  <ResourcePool>
    <!—Media is partitioned so that it can be referenced from the AmountPool -->
    <Media ID="r0006" Class="Consumable" Status="Available" PartIDKeys="RunIndex">
      <Media RunIndex="0 -1"/>
      <Media RunIndex="1~-2"/>
    </Media>
    <DigitalPrintingParams ID="r0007" Class="Parameter" rRefs="r0006" Status="Available" PartID-
Keys="RunIndex">
      <DigitalPrintingParams RunIndex="0 -1">
      <!-- PartAmount with <Part RunIndex="0 -1"/> contains the partition details for this Me-
diaRef -->
        <MediaRef rRef="r0006">
          <Part RunIndex="0 -1"/>
        </MediaRef>
      </DigitalPrintingParams>
      <DigitalPrintingParams RunIndex="1~-2">
      <!-- PartAmount with <Part RunIndex="1~-2/> contains the partition details for this Me-
diaRef -->
        <MediaRef rRef="r0006">
          <Part RunIndex="1~-2"/>
```

```
          </MediaRef>
       </DigitalPrintingParams>
    </DigitalPrintingParams>
  </ResourcePool>
  <ResourceLinkPool>
    <MediaLink rRef="r0006" Usage="Input">
      <!-- the AmountPool contains the ResourceLink partition details -->
      <AmountPool>
        <PartAmount Usage="Input" Orientation="Flip180">
          <Part RunIndex="0 -1"/>
        </PartAmount>
        <PartAmount Usage="Input" Orientation="Rotate0">
          <Part RunIndex="1~-2"/>
        </PartAmount>
      </AmountPool>
    </MediaLink>
    <DigitalPrintingParamsLink rRef="r0007" Usage="Input"/>
  </ResourceLinkPool>
</JDF>
```

3.9 Subsets of Resources

In many cases, a set of similar resources—such as separation films, plates, or **RunList** resources—is produced by one process and consumed by another. When this occurs, it is convenient to define one resource element that describes the complete set and allows individual subsets to be referenced. This mechanism also removes process ambiguity if multiple input resource links and multiple output resource links exist that must be unambiguously correlated.

In other cases, there can be a need to change some attribute of a parameter resource for some subset of the processing to be done by a device (for instance, when printing a document using *DigitalPrinting*, it would be a common application to change the dimensions of the media to be selected based on the actual media box changes in a PDF file).

Resource elements and ResourceLink elements have optional attributes that enable an agent to specify an explicit part of a structured resource. There are two ways to reference a subset of a resource.   The first is by quantity, by specifying an Amount in a ResourceLink that is less than the Resource's Amount. The second is to select certain parts of a partitioned resource by supplying a filtering Part element in the ResourceLink.

3.9.1 Resource Amount

Yet another flexible feature of resources is that they may be only partially consumed. For example, in a scenario in which various versions of a product share identical parts—such as versioned books that all have the same cover— each version will only use as many copies of the cover as it needs to fulfill its job requirement, even though all of the covers can be printed in one step for all versions. This feature is specified in the *Amount* attribute of the resource links and allows multiple JDF nodes to share resources. It allows both the sharing of output resources (as when a binding process consumes identical sheets from multiple press lines) and the sharing of input resources (as when the covers for multiple jobs are identical and are all printed in one press run).

The *Amount* attribute of a physical resource element contains the actual amount of a given resource. It is adjusted by the production or consumption amount of every process that is executed, and refers to that amount in the corresponding physical resource link element. Thus the value of the *Amount* attribute of a resource that is consumed as an input should be reduced by the amount that is consumed. It is up to the agent that writes a JDF job to ensure that the *Amount* attributes of resources and the resource links that reference them are consistent. The units used in the *Amount* attribute of a physical resource link element is defined by the unit of the resource element to which the link refers. The definition of *Amount* for partitioned resources is explained in detail in Section 3.9.2 Description of Partitionable Resources.

Note that for resources which are the output of processes, the Amount attribute on the ResourceLink determines the quantity of the resource to be produced. For example, for a DigitalPrinting process that included a RunList as its input with 16 pages to be printed and a ComponentLink to its output, the Amount attribute would indicate the number of copies of those 16 pages that the process would produce.

3.9.2 Description of Partitionable Resources

Printing workflows contain a number of processes that are repeated over a potentially large number of individual files, sheets, surfaces or separations. In order to define a partitioned resource in a concise manner without having to create a large number of individual nodes and resources, a set of resources may be partitioned by factoring them by one or more attributes. The common elements and defaults are placed in the parent element, while partition-specific attributes and overrides are placed in the child elements. This saves space. Also, by providing a single parent ID for the resources, it allows easy access to the entire resource, or iteration over each part.

To reference part of a resource, a ResourceLink references the parent resource, and supplies a Part element that contains an actual value for a partition. The result is all the child elements with matching partition values, including common values and defaults from the parent resource. If *PartUsage* = "*Implicit*", the parent attributes are returned even if there is no matching partition.

A partitionable resource contains nested elements, each with the same name as the resource. The part-independent resource elements and attributes are located in the root of the resource, while the partition-dependent elements are located in the nested elements. Thus one individual part is defined by the convolution of the partition-independent elements and attributes, with the elements and attributes contained in the appropriate nested elements. The attributes of nested part elements may be overwritten by the equivalent attributes in descendent parts. If a leaf contains elements that may multiply, and additional elements with the same name exist in nodes that are closer to the root, only the elements in the leaf are valid for the respective part. For example, the following SeparationSpec is two color duo-tone (only Black and SpotGreen) in the part with *PageNumber*=1:

```
<LayoutElement PartIDKeys="PageNumber">
  <SeparationSpec Name="Cyan"/>
  <SeparationSpec Name="Magenta"/>
  <SeparationSpec Name="Yellow"/>
  <SeparationSpec Name="Black"/>
  <FileSpec (…)/>
  <LayoutElement PageNumber="0" (…)/>
  <LayoutElement PageNumber="1" (…)>
    <SeparationSpec Name="Black"/>
    <SeparationSpec Name="SpotGreen"/>
  </LayoutElement>
</LayoutElement>
```

The *Amount* attribute of a partitioned resource is treated formally exactly in the same manner as any other attribute. This implies that the amount specified refers to the amount defined by one leaf and not to the amount defined by the sum of leaves in a branch. The *Amount* attribute defined in the example below is, therefore, two, even though 24 physical plates are described.

The following example defines two sets of 12 plates for two sheets with three surfaces. Each has a common brand attribute called "Gooey". Each individual separation has its own *ProductID*. Furthermore, the *Status* attribute varies from part to part. For example, if a yellow plate breaks, only it will need to be remade and therefore set to *Unavailable*; the others, meanwhile, may remain *Available*.

```
<ExposedMedia Class="Handling" Brand="Gooey" ID="L1" Status="Available" PartID-
Keys="SheetName Side Separation" Amount="2">
  <Media MediaType="Plate" Dimension="500 600"/>
  <ExposedMedia SheetName="S1">
    <ExposedMedia Side="Front">
      <ExposedMedia Separation="Cyan" ProductID="S1FCPlateJ42"/>
      <ExposedMedia Separation="Magenta" ProductID="S1FMPlateJ42"/>
      <ExposedMedia Separation="Yellow" ProductID="S1FYPlateJ42"
Status="Unavailable"/>
      <ExposedMedia Separation="Black" ProductID="S1FKPlateJ42"/>
    </ExposedMedia>
    <ExposedMedia Side="Back">
      <ExposedMedia Separation="Cyan" ProductID="S1BCPlateJ42"/>
      <ExposedMedia Separation="Magenta" ProductID="S1BMPlateJ42"/>
      <ExposedMedia Separation="Yellow" ProductID="S1BYPlateJ42"/>
      <ExposedMedia Separation="Black" ProductID="S1BKPlateJ42"/>
```

```
      </ExposedMedia>
   </ExposedMedia>
   <ExposedMedia SheetName="S2" Side="Front">
      <ExposedMedia Separation="Cyan" ProductID="S2FCPlateJ42"/>
      <ExposedMedia Separation="Magenta" ProductID="S2FMPlateJ42"/>
      <ExposedMedia Separation="Yellow" ProductID="S2FYPlateJ42"/>
      <ExposedMedia Separation="Black" ProductID="S2FKPlateJ42"/>
   </ExposedMedia>
</ExposedMedia>
```

Note: Only resources may be partitioned. If a resource contains subelements, the subelements must NOT be partitioned. Subelements must be always specified completely in that part where they occur. The content of subelements is not convoluted with the content of subelements in parts closer to the root.

Five examples are provided below. The first and the fourth example are valid, the second third, and fifth are invalid. In the first example, the **ExposedMedia** resource is partitioned:

```
<ExposedMedia ID="L1" Status="Available" PartIDKeys="Separation" … >
   <Media MediaType="Film" Brand="foo"/>
   <ExposedMedia Separation="Cyan"/>
   <ExposedMedia Separation="Magenta">
      <Media MediaType="Film" Brand="bar"/>
   </ExposedMedia >
</ExposedMedia >
```

In this invalid example #2, the Media in the leaves is not complete because it does not contain the MediaType attribute. MediaType cannot not be derived from the Media part in the root element:

```
<ExposedMedia ID="L1" Status="Available" PartIDKeys="Separation" … >
   <Media MediaType="Film"/>
   <ExposedMedia Separation="Cyan">
      <Media Brand="foo"/>
   </ExposedMedia >
   <ExposedMedia Separation="Magenta">
      <Media Brand="bar"/>
   </ExposedMedia >
</ExposedMedia >
```

In this invalid example #3, Media is a subelement that must NOT be partitioned:

```
<ExposedMedia ID="L1" Status="Available" PartIDKeys="Separation" … >
   <Media MediaType="Film">
      <Media Brand="foo" Separation="Cyan">
      <Media Brand="bar" Separation="Magenta" />
   </Media >
</ExposedMedia >
```

Partitioning may be combined with inter-resource links, i.e. RefElements. In this case the partitioning attributes of the referenced resource must be a subset of the partitioning attributes of the resource leaf that contains the ResourceRef. In the following valid example #4, each MediaRef is equivalent to an in-lined leaf with the explicit Part elements to define the partition, i.e. it is equivalent to the valid example #1.

```
<Media ID="MediaID" MediaType="Film" PartIDKeys="Separation">
   <Media Separation="Cyan" Brand="foo"/>
   <Media Separation="Magenta" Brand="bar"/>
</Media>
<ExposedMedia ID="L1" Status="Available" PartIDKeys="Separation" … >
   <ExposedMedia Separation="Cyan">
      <!—equivalent to <Media MediaType="Film" Brand="foo"/> -->
      <MediaRef rRef="MediaID">
         <Part Separation="Cyan"/>
      </MediaRef>
   </ExposedMedia>
   <ExposedMedia Separation="Magenta">
      <!—equivalent to <Media MediaType="Film" Brand="bar"/> -->
      <MediaRef rRef="MediaID"/>
         <Part Separation=" Magenta"/>
```

```
    </MediaRef>
  </ExposedMedia >
</ExposedMedia >
```

In this invalid example #5, **MediaRef** does not reference the leaves of Media, but rather the root of **Media**. It is equivalent to the invalid example #3.

```
<Media ID="MediaID" MediaType="Film" PartIDKeys="Separation">
  <Media Separation="Cyan" Brand="foo"/>
  <Media Separation="Magenta" Brand="bar"/>
</Media>
<ExposedMedia ID="L1" Status="Available" PartIDKeys="Separation" … >
  <MediaRef rRef="MediaID">
</ExposedMedia >
```

In addition to the usual resource attributes and elements, the partitionable **Resource** element has partition-specific attributes and elements in its root. *PartIDKeys* is required in the root of a partitioned resource. Further attributes are listed in the following table:

Table 3-24 Contents of the partitionable Resource element

Name	Data Type	Description
PartIDKeys ?	enumerations	List of attribute names that are used to separate the individual parts. Possible values are:
		BlockName *PageNumber* *SetIndex*
		DocCopies *PartVersion* *SheetIndex*
		DocIndex *PreviewType* *SheetName*
		DocRunIndex *RibbonName* *Side*
		DocSheetIndex *Run* *SignatureName*
		FountainNumber *RunIndex* *TileID*
		LayerIDs *RunTags* *WebName*
		Location *RunPage*
		Option *Separation*
		For details, see Table 3-25.
PartUsage ? New in JDF 1.1	enumeration	Description of the interpretation of partitions. One of:
		Explicit – Require explicit partition matches. All referenced partitions referenced in Part must exist, otherwise it is an error. The default attributes are returned, overridden by the partition's values, if found. This is the default behavior.
		Implicit – Allow sparse overrides of default values. The referenced partition is not required to exist. The default attributes are returned, overridden by the partition's values, if found.
Resource *	element	Nested resource elements that contain the appropriate part ID(s). These elements must be of the same type as the root **Resource** element. They represent the individual parts or groups of parts.

Partitionable resources are uniquely identified by the attribute values listed in *PartIDKeys* attributes. The choice of which attributes to use depends on how the agent organizes the job.

The following table lists the content of a **Part** element, which contains a set of attributes that have a well described meaning. Each of the attributes, except *Sorting*, may be used in the nested resource elements of partitionable resources as the part ID key (see example above).

Part elements match a given partition when all of the attributes of a **Part** element match the attributes of the referenced Resource. This corresponds to Boolean AND operation. If multiple **Part** elements are defined, the result is a Boolean OR of the multiple parts.

Table 3-25 Contents of the Part element

Name	Data Type	Description
BlockName ? New in JDF 1.1	NMTOKEN	Identifies a *CutBlock* from a **Cutting** process. The value of this attribute must match the value of the *BlockName* attribute of a Cut-Block.
DocCopies ?	IntegerRangeList	Identifies a set of document copies to which the partition applies. *DocCopies* is a logical reference that may be independent of the RunList structure and must NOT be used as a partition key for spawning and merging of **RunList** resources.
DocIndex ?	IntegerRangeList	The *DocIndex* attribute selects a set of logical instance documents from a **RunList** resource. *DocIndex* is a logical reference that may be independent of the **RunList** structure and must NOT be used as a partition key for spawning and merging of **RunList** resources.
DocRunIndex ?	IntegerRangeList	The *DocRunIndex* attribute selects a set of logical pages from instance documents of a **RunList** resource. For example *DocRunIndex = "0 –1"* specifies the first and last page of every copy of every selected instance document (assuming that additional partitioning using DocCopies and/or DocIndex is not also specified). *DocRunIndex* is a logical reference that may be independent of the *RunList* structure and must NOT be used as a partition key for spawning and merging of **RunList** resources. The index always refers to entries of the entire **RunList** and must not be modified if only a part of the **RunList** is spawned. Not to be used in conjunction with RunTags.
DocSheetIndex ?	IntegerRangeList	The *DocSheetIndex* attribute selects a set of logical sheets from individual instance documents. For example *DocSheetIndex = "0 – 1"* specifies the first and last sheet of every selected copy of every instance document (assuming that additional partitioning using DocCopies and/or DocIndex is not also specified). *DocSheetIndex* is a logical reference that may be independent of the RunList structure and must NOT be used as a partition key for spawning and merging of **RunList** resources. The index always refers to entries of the entire **RunList** and must not be modified if only a part of the **RunList** is spawned.
FountainNumber ?	integer	Zero based position index of the fountain. Used to partition fountains along the axis of a roller, may be used for web printing.
LayerIDs ? New in JDF 1.1	IntegerRangeList	The *LayerIDs* attribute selects a set layers that are defined by LayerID. If not specified, all layers are processed.
Location ?	string	Name of the location, e.g., for example in MIS. This part key allows to describe distributed resources.
Option ?	string	Option of an RFQ. Used mainly in Intent resources.
PageNumber ?	IntegerRangeList	Page number in a **Component** or document, e.g., **FileSpec** that is not described as a **RunList**.
PartVersion ?	string	Version identifier, such as the language version of a catalog.
PreviewType ? New in JDF 1.1	enumeration	Type of the preview. Possible values are: *Separation*: separated preview in medium resolution. *SeparatedThumbNail*: Very low resolution separated preview. *ThumbNail*: Very low resolution rgb preview. *Viewable:* rgb preview in medium resolution.
RibbonName ?	string	A string that uniquely identifies each ribbon. Multiple ribbons are created out of one web after dividing in case of web printing.

Name	Data Type	Description
Run ? **Modified in JDF 1.1**	string	The *Run* attribute selects a set of partitioned RunList elements from a **RunList** resource.
RunIndex ?	IntegerRangeList	The *RunIndex* attribute selects a set of logical pages from a **RunList** resource in a manner that is independent from the internal structure of the **RunList**. It contains an array of mixed ranges and individual indices separated by whitespace. Each range consists of two indices connected with a tilde (~) and no whitespace. For example, *RunIndex* = "2~5 8 10 22~-1". Negative numbers reference pages from the back of a file in base-1 counting. In other words, -1 is the last page, -2 the second to last, etc. Thus RunIndex = "0~-1" refers to a complete range of pages, from first to last. *RunIndex* is a logical reference that is independent of the RunList structure and must NOT be used as a partition key for spawning and merging. The index always refers to entries of the entire **RunList** and must not be modified if only a part of the **RunList** is spawned. Not to be used in conjunction with *RunTags*.
RunTags ? **New in JDF 1.1**	NMTOKENS	List of names in a named **RunList.** Used to partition resources that are linked from processes that also have a **RunList** as input when the sequence of the **RunList** is undefined. The partition is selected if the *RunTag* of the **RunList** matches any of the entries in *RunTags*. Not to be used in conjunction with *RunIndex* or *DocRunIndex*.
RunPage ? **New in JDF 1.1**	integer	Zero based page number. Used when a document / file based **RunList** is broken down into a page based **RunList**.
Separation ?	string	Identifies the separation name. Possible values include: *Composite* – Non-separated resource. *Separated* – The resource is separated, but the separation definition is handled internally by the resource, such as a PDF file that contains *SeparationInfo* dictionaries. *Cyan* – Process color. *Magenta* – Process color. *Yellow* – Process color. *Black* – Process color. *Red* – Additional process color. *Green* – Additional process color. *Blue* – Additional process color. *Orange* – Additional process color. *Spot* – Generic spot color. Used when the exact nature of the spot color is unknown. *Varnish* – Varnish. Other values may be any separation name defined in the *Name* attribute of a Color element in the **ColorPool**.
SetIndex ? **New in JDF 1.1**	IntegerRangeList	The *SetIndex* attribute selects a set of logical instance document sets from a **RunList** resource. *SetIndex* is a logical reference that may be independent of the **RunList** structure and must NOT be used as a partition key for spawning and merging of **RunList** resources. The index always refers to entries of the entire **RunList** and must not be modified if only a part of the **RunList** is spawned.

Name	Data Type	Description
SheetIndex ?	IntegerRangeList	The *SheetIndex* attribute selects a set of logical sheets from a **RunList** resource. In 1-up simplex printing, it is identical to *RunIndex*. *SheetIndex* is a logical reference that is independent of the RunList structure and must NOT be used as a partition key for spawning and merging.
SheetName ?	string	A string that uniquely identifies each sheet. The value of this attribute must match the value of the *Name* attribute of a sheet.
Side ?	enumeration	Denotes the side of the sheet. Possible values are: *Front* *Back* If Side is specified, the **Part** element refers to one surface of the sheet. If it is not specified, it refers to both sides. In case of web printing, *Front* is a synonym for the upper side and *Back* for the down side of the web.
SignatureName ?	string	A string that uniquely identifies the signature within the partitionable resource.
Sorting ?	IntegerRangeList	Mapping from the implied partitionable resource order to a process order. The indices refer to the elements of the complete partitionable resource, not to the index in the selection of parts defined by the **Part** element.[1] Defaults to "0~-1", i.e. the part order is the same as the sorting order. *Sorting* must NOT be used as a partition key.
SortAmount ?	boolean	If a sorted resource has an *Amount* attribute and *SortAmount* = *true*, each resource must be processed completely. If *SortAmount* = *false* (the default), each **Part** element must be processed the number of times specified in the *Amount* attribute before starting the next **Part**. *SortAmount* must NOT be used as a partition key.
TileID ?	XYPair	XYPair of integer values that identifies the tile. Tiles are identified by their X and Y indexes. Values are zero-based and expressed in the PS coordinate system. So "0 0" is the lower left tile and "1 0" is the tile next to it on the right. **Tile** resources are described in detail in the Section 7.2.214 **Tile**. May also be used to identify multiple plates per cylinder. Then the x-index corresponds to a zero based position index along the axis of a roller and the y-value to a zero based position index along the circumference of a roller.
WebName ?	string	A string that uniquely identifies each web.

If multiple Part ID keys are used in a partitioned resource, for example *PartIDKeys*="*SheetName Side Separation Location*", then all part ID keys must be defined for each leaf in the partitioned resource. In other words, if you walk from a leaf of a partitioned resource up to the root, each of the part ID keys defined in *PartIDKeys* must occur exactly one time. For example, it is not allowed that only the part ID keys *SheetName* and *Separation* be defined for some leaves in a partitioned resource with *PartIDKeys*="*SheetName Side Separation*" defined in the root.

---

[1] Note that *Sorting* is semantically different from the other attributes in this table, as it implies an ordering of parts, whereas the other attributes define a selection of parts.

3.9.2.1 Options in Intent Resources

JDF defines *Option* as a part key in order to specify multiple options e.g. for multiple quotes in a non-redundant manner. A ResourceLink that links to a resource with an *Option* partition but has no Part element to choose the *Option*, defaults to the root resource.

3.9.2.2 Locations of Physical Resources

Unlike other kinds of resources, physical resources may be stored at multiple, distributed locations. This is specified by including one or more *Location* parts in the resource element and accessing the location by specifying a Part element with a *Location* attribute in the respective ResourceLink.

Location:*LocationName* may also be used to specify a *Location* within a device, e.g., a paper tray. When specifying paper trays, the following locations are predefined (New in JDF 1.1):

Table 3-26 Locations within Printers

Name	Description
Top	The bin that, when facing the device, can best be identified as 'top'.
Middle	The bin that, when facing the device, can best be identified as 'middle'.
Bottom	The bin that, when facing the device, can best be identified as 'bottom'.
Side	The bin that, when facing the device, can best be identified as 'side'.
Left	The bin that, when facing the device, can best be identified as 'left.
Right	The bin that, when facing the device, can best be identified as 'right.
Center	The bin that, when facing the device, can best be identified as 'center.
Rear	The bin that, when facing the device, can best be identified as 'rear'.
FaceUp	The bin that can best be identified as 'face up' with respect to the device.
FaceDown	The bin that can best be identified as 'face down' with respect to the device.
FitMedia	Requests the device to select a bin based on the size of the media.
LargeCapacity	The bin that can best be identified as the 'large capacity' bin (in terms of the number of sheets) with respect to the device.
Mailbox-N	The job will be output to the bin that is best identified as 'Mailbox-1', 'Mailbox-2'…etc.
Stacker-N	The job will be output to the bin that is best identified as 'Stacker-1', 'Stacker-2' …etc.
Tray-N	The job will be output to the tray that is best identified as 'Tray-1', 'Tray-2' … etc.
SystemSpecified	The job will be output to the tray that is defined by the system.

The following example describes a set of plates that are distributed over two locations:

```
<ExposedMedia ID="L1" PartIDKeys="Location" … >
  <ExposedMedia Amount="42" Location="Desk Drawer 1">
    <Location LocationName="Desk Drawer 1" LocID="PP_01234">
      <Address … />
    </Location>
  </ExposedMedia>
  <ExposedMedia Amount="100" Location="Desk Drawer 2">
    <Location LocationName="Desk Drawer 2" LocID="PP_01235">
      <Address … />
    </Location>
</ExposedMedia>
…
<ExposedMediaLink ResourceID="L1" Amount="50" Usage="Input">
  <Part Location="Desk Drawer 2"/>
</ExposedMediaLink>
```

The following example describes two different Media in the top and bottom tray of a LayoutPreparation process. The Media is selected for the cover and inside pages respectively.

```
<Media ID="TopMedia" … >
    <Location LocationName="Top"/>
</Media>
<Media ID="BottomMedia" … >
    <Location LocationName="Bottom"/>
</Media>

…
<LayoutPreparationParams Sides="TwoSidedFlipY" PartIDKeys="RunIndex"(…)>
  <!-- Partition that defines the first and last page of the document -->
  <LayoutPreparationParams RunIndex="0 1 -2 -1">
    <MediaRef rRef="TopMedia"/>
  </LayoutPreparationParams>
  <!-- Partition that defines the inside pages of the document -->
  <LayoutPreparationParams RunIndex="2~-3">
    <MediaRef rRef="BottomMedia"/>
  </LayoutPreparationParams>
</LayoutPreparationParams>
```

3.9.3 Linking to Subsets of Resources

An agent can link to a subset of a resource by including a Part element in a ResourceLink element in order to define a specific subset of a resource. For details of the Part element, please refer to Table 3-25 Contents of the Part element .

Partitionable hierarchies define an implied ordering of the individual parts. In the example in Section 3.9.2 Description of Partitionable Resources, the first element has a *ProductID* = *S1FCPlateJ42* and the last has a *ProductId* = *S2FKPlateJ42*. If process ordering of a partitionable resource is important, the Part element of the ResourceLink must specify a *Sorting* attribute. If *Sorting* is not specified, process ordering is arbitrary. If *Sorting* is specified multiple times, the resolution of the sorting must be unambiguous.

The *Sorting* attribute maps the implied part ordering to a specified process ordering in a 0-based list. The first entry in *Sorting* defines the first entry to be processed. The following example, using a ResourceLink element, describes how the plates described in the previous example could be ordered by separation for the first sheet followed by the complete second sheet, in reverse order (back to front). Each set of two plates, as specified in the *Amount* attribute of the resource, would be processed together.

```
<ExposedMediaLink rRef="L1">
  <Part Sorting="0 4 1 5 2 6 3 7 -1~8" SortAmount="false"/>
</ExposedMediaLink>
```

A partitionable resource may also be split into individual resources by an agent. In this case, one resource must be created for each individual part or set of parts. For example, a resource that describes a set of films that are also separated may be split into a set of resources that each describe all separations of a sheet.

3.9.3.1 Handling Amount in a ResourceLink to a Partitioned Resource

The *Amount* specified in a *ResourceLink* to a physical resource specifies the sum of individual resource partitions. Individual amounts are specified in the PartAmount elements of the AmountPool. The following example shows the *ResourceLink* that refers to the previous example for a total of five plates.

```
<ExposedMediaLink rRef="L1" Amount="4">
  <Part SheetName="S1" Separation="Cyan"/>
  <Part SheetName="S1" Separation="Magenta"/>
  <AmountPool Amount="1">
    <PartAmount>
      <Part SheetName="S1" Side="Front" Separation="Cyan"/>
```

```
      </PartAmount>
      <PartAmount>
         <Part SheetName="S1" Side="Back" Separation="Cyan"/>
      </PartAmount>
      <PartAmount>
         <Part SheetName="S1" Side="Front" Separation="Magenta"/>
      </PartAmount>
      <PartAmount Amount="2">
         <Part SheetName="S1" Side="Back" Separation="Magenta"/>
      </PartAmount>
   </AmountPool>
</ExposedMediaLink>
```

3.9.3.2 Referencing Partitioned Resources from Nodes That Allow Multiple ResourceLinks.

Some processes, e.g., *Collecting, Gathering* allow multiple input resources of the same type. These multiple input resources may be represented by multiple individual resources or by partitioned resources or by a mixture of both. If ordering is significant, the order of the leaves in a partitioned resource defines said ordering. The following examples of gathering three input sheets are equivalent:

Explicit reference of ordered partitioned resources:
```
<JDF ID="Link0037" Type="Gathering" Status="Waiting">
   <ResourcePool>
      <GatheringParams ID="Gather01" Class="Parameter" Locked="false"
Status="Available"/>
      <Component ID="Sheets01" Class="Quantity" Status="Available" PartID-
Keys="SheetName" ComponentType="Sheet" DescriptiveName="printed insert
sheets">
         <Component SheetName="Sheet1"/>
         <Component SheetName="Sheet2"/>
         <Component SheetName="Sheet3"/>
      </Component>
   </ResourcePool>
   <ResourceLinkPool>
      <GatheringParamsLink rRef="Gather01" Usage="Input"/>
      <!--three ComponentLink explicitly reference individual parts -->
      <ComponentLink rRef="Sheets01" Usage="Input">
       <Part SheetName="Sheet1"/>
      </ComponentLink>
      <ComponentLink rRef="Sheets01" Usage="Input">
       <Part SheetName="Sheet2"/>
      </ComponentLink>
      <ComponentLink rRef="Sheets01" Usage="Input">
        <Part SheetName="Sheet3"/>
      </ComponentLink>
   </ResourceLinkPool>
</JDF>
```

Implicit reference of ordered partitioned resources:
```
<JDF ID="Link0037" Type="Gathering" Status="Waiting">
   <ResourcePool>
      <GatheringParams ID="Gather01" Class="Parameter" Locked="false"
Status="Available"/>
      <Component ID="Sheets01" Class="Quantity" Status="Available" PartID-
Keys="SheetName" ComponentType="Sheet" DescriptiveName="printed insert
sheets">
         <Component SheetName="Sheet1"/>
         <Component SheetName="Sheet2"/>
         <Component SheetName="Sheet3"/>
```

```
        </Component>
    </ResourcePool>
    <ResourceLinkPool>
        <GatheringParamsLink rRef="Gather01" Usage="Input"/>
        <!—the ComponentLink implicitly references all three parts -->
        <ComponentLink rRef="Sheets01" Usage="Input"/>
    </ResourceLinkPool>
</JDF>
```

3.9.4 Splitting and Combining Resources

Depending on the circumstances, it may be appropriate either to split a resource into multiple new nodes or to specify multiple locations or parts for an individual resource. There are four possible methods for splitting and combining resources, each of which is illustrated in Figure 3.8, below. Both Case A and Case B in Figure 3.8 represent workflows that use the *Amount* attribute of their resource links to share resources. This method is practical when one controller controls all aspects of resource consumption or production. In Case A, the resource amount is split between subsequent processes. In Case B, individual processes produce amounts that are then combined into a unified resource that is, in turn, used by a single process. In both cases, a single, shared resource is employed. To enable independent parallel processing by multiple controllers, however, independent resources are required. To create independent resources from one resource, the *Split* process is used, as shown in Case C (for further details, see Section 6.2.9 Split). This process allows multiple processes to be spawned off, after which multiple processes can consume the same resource in parallel and may therefore run in parallel. Case D demonstrates the reverse situation, which occurs if resources have been produced by multiple processes and are then consumed, as a unified entity, by a single subsequent process. To accomplish this, the *Combine* process combines multiple resources to create the single resource.

A: brief workflow for splitting by a shared input resource

Node A —Amount 1+2+3→ Res 1+2+3 —Amount 1→ Node 1
—Amount 2→ Node 2
—Amount 3→ Node 3

B: brief workflow for combining by a shared output resource

Node 1 —Amount 1
Node 2 —Amount 2→ Res1+2+3 —Amount 1+2+3→ Node B
Node 3 —Amount 3

C: exact workflow for splitting

Node A → Res 1+2+3 → Split-Node → Res 1 → Node 1
Res 2 → Node 2
Res 3 → Node 3

D: exact workflow for combining

Node 1 → Res 1
Node 2 → Res 2 → Combine-Node → Res1+2+3 → Node B
Node 3 → Res 3

Figure 3.8 Splitting and combining physical resources

3.10 AuditPool

Audit elements contain the post-facto recorded results of a process such as the execution of a JDF node or modification of the JDF itself. Audit elements become static after a process has been finished. They cannot ever be modified after the process has been aborted or completed. Therefore, if Audit elements link to resources, those resources should be locked in order to inhibit accidental modification of audited information, which is why JDF includes a locking mechanism for resources. The *ID* of all resources that are referenced by Audit elements must be included in the *rRefs* attribute of the AuditPool in order to enable spawning and merging. Audit elements record any event related to the following situations:

Audit Pools

Audit information is the Job's history and can support your daily, quality control and troubleshooting management reporting needs.

1. The creation of a JDF node by a Created element.

2. Spawning and merging, including resource copying by spawned and merged elements.

3. Errors such as unnecessary ResourceLink elements, wrongly linked resources, missing resources, or missing links, which may be detected by agents during a test run or by a Notification element.

4. Actual data about the production and resource consumption by a ResourceAudit element.

5. Any process phase times. Examples include setting up a device, maintenance, and washing, as well as downtimes as a result of failure, breaks, or pauses. Changes of implementation resource usage, such as a change of operators by a PhaseTime element, would also constitute an example of a phase time.

6. Actual process scheduling data. For example, the process start and end times, as well as the final process state, as determined by a ProcessRun element.

7. Any modification of a JDF node not covered by the preceding items, as recorded by a Modified element.

Audit information may be used by MIS for operations such as evaluation or invoicing. Figure 3.9 depicts the structure of the AuditPool and Audit element types derived from the abstract audit type.

```
AuditPool?              Audit*
•   rRefs?              •   Author?
                        •   SpawnID?
                        •   TimeStamp


Created*                              Spawned *              Part*
•    ref                              •   Independent?
                                      •   jRef
Modified*                             •   jRefDestination?
•    jRef                             •   NewSpawnID
                                      •   rRefsROCopied?
                                      •   rRefsRWCopied?
                                      •   Status?
                                      •   URL?


                                      Merged*               Part*
                                      •   Independent?
                                      •   jRef
                                      •   jRefSource?
                                      •   MergeID
                                      •   rRefsOverwritten?
                                      •   URL?


                                                            Comment*

                                                            CostCenter?

                                      Notification*         Employee*
                                      •   Class
                                      •   Type?             NotificationDetails?

                                                            Part*


                                                            Device*

                                                            Employee*
                                      PhaseTime*
                                      •   End               ModulPhase*          Employee*
                                      •   Start             •   DeviceID?
                                      •   Status            •   DeviceStatus
                                      •   StatusDetails?    •   End
                                                           •   ModulIndex
                                                           •   ModulType
                                                           •   Start
                                                           •   StatusDetails?

                                                            Part*

                                                            ResourceLink*


                                      ResourceAudit*        ResourceLink
                                      •   ContentsModified?
                                      •   Reason?           ResourceLink?


                                      ProcessRun*           Part*
                                      •   Duration?
                                      •   End
                                      •   EndStatus
                                      •   Start
```

Attributes:
ref   = reference via ID to a resource or a JDF-node
jRef   = reference via ID to a JDF-node

Notification:
Class   = Event | Information | Warning | Error | Fatal

Figure 3.9 Structure of Audit element types derived from the abstract Audit type

Audit entries are ordered chronologically, with the last entry in the AuditPool representing the newest. A Processor ProcessRun element containing the scheduling data finalizes each process run. All subsequent entries belong to the next run. The following table defines the contents of the AuditPool element.

Table 3-27 Contents of the AuditPool element

Name	Data Type	Description
rRefs ?	IDREFS	List of all resources that are referenced from within the AuditPool. Needed for Spawning.
Audit *	element	Chronologically ordered list of Audit elements. The Audit elements are abstract and serve as placeholders for any audit. Audit elements are described in the sections that follow.

3.10.1 Audit Elements

All Audit elements inherit the content from the abstract Audit data type, described in the following table.

Table 3-28 Contents of the abstract Audit type

Name	Data Type	Description
Author ? Modified in JDF 1.1A	string	Text that identifies who made the entry. This can describe a person, an agent, or both. While the *Author* attribute is optional, it is strongly recommended to include this attribute whenever possible.
SpawnID ? New in JDF 1.1	NMTOKEN	Text that identifies the spawned processing step when the entry was generated. This is a copy of the SpawnID attribute of the root JDF node of the process that generates the Audit at the time the Audit is generated.
TimeStamp	dateTime	In case of the audits Created, Modified, Spawned, Merged, and Notification, this attribute records the date and time when the related event occurred. In case of the audits PhaseTime, ProcessRun, and ResourceAudit, the attribute describes the time when the entry was appended to the audit pool.

It is anticipated that additional attributes that clearly define the agent that inserted an Audit will be defined in JDF 1.2. The following table describes the preliminary syntax within the JDF extension namespace and is provided as a preview only that is subject to change. For details on extension namespaces see 3.11.1.2 JDF Extension Namespace:

Table 3-28- 1 Extensions of the abstract Audit type

Name	Data Type	Description
JDFX:AgentName ?	String	The name of the agent application that added the audit element to the audit pool (and was responsible for the creation or modification). Both the company name and the product name can appear, and should be consistent between versions of the application.
JDFX:AgentVersion ?	String	The version of the agent application that added the audit element to the audit pool (and was responsible for the creation or modification). The format of the version string can vary from one application to another, but should be consistent for an individual application.

Listed in the following sections are the elements derived from the abstract Audit type. Following the description of each element is a table outlining the attributes associated with that element.

3.10.1.1 ProcessRun

This element serves two related functions. Its first is to summarize one complete execution run of a node. It contains attributes that record the date and time of the start, the end time, the final process state when the run is finished, and, optionally, the process duration of the process run. These attributes are described in Table 3-29.

Table 3-29 Contents of the ProcessRun element

Name	Data Type	Description
Duration ?	duration	Time span of the effective process runtime without intentional or unintentional

Name	Data Type	Description
		breaks. That time span is the sum of all process phases when the *Status* is *InProgress*, *Setup* or *Cleanup*.
End	dateTime	Date and time at which the process ends.
EndStatus	enumeration	The *Status* of the process at the end of the run. For a description of process states, see Table 3-3 Contents of a JDF node. Possible values are: *Aborted* *Completed* *FailedTestRun* *Ready* *Stopped* – The execution of the node is stopped and may commence at a later time, e.g., on another device.
Start	dateTime	Date and time at which the process starts.
Part * New in JDF 1.1	element	Describes which parts of a process this ProcessRun belongs to. If Part is not specified for a ProcessRun, it refers to all parts. For example, imagine a print job that should produce three different sheets. All sheets are described by one partitioned resource. The Part elements define, unambiguously, the processing of the sheet to which the ProcessRun refers.

The second function of a ProcessRun element is to delimit a group of audits for each individual process run. Every group of audits terminates with a ProcessRun element, which contains the information described above. If a process must be repeated (as a result of a late change in the order, for example), all audits belonging to the new run will be appended after the last ProcessRun element that terminates the audits of the previous run. The number of ProcessRun elements is, therefore, always equivalent to the number of process runs.

If a node describes partitioned resources, one ProcessRun may be specified for each individual part.

3.10.1.2 Notification
This element contains information about individual events that occurred during processing. For a detailed discussion of event properties, see Section 4.6 Error Handling.

Table 3-30 Contents of the Notification element

Name	Data Type	Description
Class	enumeration	Class of the notification. Possible values, in order of severity from lowest to highest, are: *Event* – Indicates that a pure event due to any activity has occurred, for example, machine events, operator activities, etc. This class is used for the transfer of conventional event messages. In case of *Class* = *Event*, further event information should be provided by the *Type* attribute and NotificationDetails element. *Information* – Any information about a process which cannot be expressed by the other classes. No user interaction is required. *Warning* – Indicates that a minor error has occurred and an automatic fix was applied. Execution continues. *Error* – Indicates that an error has occurred that requires user interaction. Execution cannot continue. *Fatal* – Indicates that a fatal error led to abortion of the process.
Type ?	NMTOKEN	Identifies the type of notification. Also defines the name of the abstract NotificationDetails element.[2] A list of predefined Notification types is compiled in Appendix J **NotificationDetails**.

---

[2] Type allows parsers that do not have access to the schema to find the instance of NotificationDetails.

Name	Data Type	Description
Comment *	telem	The Notification element may contain Comment elements with a verbose, human-readable description of the event. If the value of the Class attribute is one of *Information*, *Warning*, *Error*, or *Fatal*, it should provide at least one Comment element. In case of *Class* = *Event*, Comment elements are optional.
CostCenter ?	element	The cost center to which this event should be charged.
Employee *	refelement	The Employee associated with this event.
Notification-Details ?	element	Abstract element which is a placeholder for additional structured information. It provides additional information beyond the *Class* and *Type* attribute and beyond the Comment element. For a list of supported NotificationDetails elements, see Appendix J **NotificationDetails**.
Part * New in JDF 1.1	element	Describes which parts of a process this Notification belongs to. If Part is not specified for a Notification, it refers to all parts. For example, imagine a print job that should produce three different sheets. All sheets are described by one partitioned resource. The Part elements define, unambiguously, the sheet to which the audit refers.

Table 3-31 Redundant table removed

Name	Data Type	Description

3.10.1.2.1 NotificationDetails

The abstract NotificationDetails element is a placeholder only with no additional attributes. For a list of supported NotificationDetails elements, see Appendix J **NotificationDetails**.

3.10.1.3 PhaseTime

This element contains audit information about the start and end times of any process states and substates, denoted as phases. Phases may reflect any arbitrary subdivisions of a process, such as maintenance, washing, plate changing, failures, and breaks.

PhaseTime elements may also be used to log the actual time spans when implementation resources are used by a process. For example, the temporary necessity of a fork lift can be logged if a PhaseTime element is added that contains a link to the fork lift device resource and specifies the actual start and end time of the usage of that fork lift.

The times specified in the PhaseTime elements should not overlap with each other and should cover the complete time range defined in the ProcessRun element that identifies the end of the run.

Table 3-32 Contents of the PhaseTime element

Name	Data Type	Description
End	dateTime	Date and time of the end of the phase.
Start	dateTime	Date and time of the beginning of the phase.
Status	enumeration	Status of the phase. Possible values of JDF node states are: *TestRunInProgress* *Setup* *InProgress* *Cleanup* *Spawned* *Stopped* The states listed above are a subset of the possible states of a JDF node. For all possible states of a JDF node see Table 3-3. The remaining set of states, i.e. the end states — *Ready*, *FailedTestRun*, *Aborted* and *Completed*—must be logged by the ProcessRun audit element that terminates the list of audits for one process run.

Name	Data Type	Description
StatusDetails ?	string	Description of the status phase that provides details beyond the enumerative values given by the Status attribute. For a list of supported values, see **Appendix G**.
Device *	refelement	Links to Device resources that are working during this phase.
Employee *	refelement	Links to Employee resources that are working during this phase.
ModulePhase *	element	Additional phase information of individual device modules, such as print units.
Part *	element	Describes which parts of a job is currently being logged. If Part is not specified for a node that modifies partitioned resources, PhaseTime refers to all parts. For example, imagine a print job that should produce 3 different sheets. All sheets are described by one partitioned resource. In order to separate the different print phases for each sheet, the Part elements define, unambiguously, the sheet to which the audit refers.
ResourceLink * New in JDF 1.1	element	These resource links specify the actual consumption/usage or production of resources during this production phase.

It is possible to monitor the states of individual modules of a complex device, such as a printer with multiple print units, by defining ModulePhase elements. One PhaseTime element may contain multiple ModulePhase elements and can, therefore, record the status of multiple units in a device. In contrast to PhaseTime audit elements ModulePhase elements are allowed to overlap in time with one another. ModulePhase elements are defined in the following table.

Table 3-33 Contents of the ModulePhase element

Name	Data Type	Description
DeviceID	string	Name of the device. This must be the DeviceID attribute of one of the Device elements specified in the PhaseTime audit.
DeviceStatus	enumeration	Status of the device module. Possible values are: *Unknown* – The module status is unknown. *Idle* – The module is not used, for example, a color print module that is inactive during a black-and-white print. *Down* – The module cannot be used. It may be broken, switched off etc. *Setup* – The module is currently being set up. *Running* – The module is currently executing. *Cleanup* – The module is currently being cleaned. *Stopped* – The module has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as running can be easy resumed. These states are analog to the device states of **Table 5-46**.
End	dateTime	Date and time of the end of the module phase.
ModuleIndex	Integer-RangeList	0-based indices of the module or modules. If multiple module types are available on one machine, it is device dependent whether the indices of each type restart at 0 or simply continue indexing.
ModuleType	NMTOKEN	Module description. The allowed values depend on the type of device that is described. The predefined values are listed in **Appendix A**.
Start	dateTime	Date and time of the beginning of the module phase.
StatusDetails ?	string	Description of the module status phase that provides details beyond the enumerative values given by the *DeviceStatus* attribute. For a list of supported values, see **Appendix G**.

Name	Data Type	Description
Employee *	refelement	Links to Employee resources that are working during this module phase on this module (the module is specified by the attributes *ModuleIndex* and *Module-Type*).

3.10.1.4 ResourceAudit

The ResourceAudit element describes the usage of resources during execution of a node or the modification of the intended usage of a resource, in other words the modification of a resource link. It logs consumption and production amounts of any quantifiable resources, accumulated over one process run or one part of a process run. It contains one or two abstract ResourceLink elements. The first is required and specifies the actual consumption/usage or production of the resource. The second ResourceLink is optional and used to store information about the original resource link, which also refers to the original resource. If the original resource does not need to be saved, a boolean *ContentsModified* attribute in the ResourceAudit should be used to indicate that a change has been made.

Table 3-34 Contents of the ResourceAudit element

Name	Data Type	Description
ContentsModified ?	boolean	Specifies that a modification has occurred but that the original resource has been deleted.
Reason ? New in JDF 1.1	enumeration	Reason for the modification. One of: *PlanChange* – The resource was modified due to a change of plan before actual processing. *ProcessResult* – The default.
ResourceLink	element	The first resource link specifies the actual consumption/usage or production of a resource.
ResourceLink ?	element	The second optional resource link logs the modification of a resource link and the modification of the resource it refers to. It holds the planned resource link which also refers to the planned resource. The planned and actual resource may be the same.

For details on ResourceLink elements and ResourceLink subclasses, see Section 3.8 Resource Links. The partitioning of resources using Part elements is defined in Section 3.9.2 Description of Partitionable Resources.

3.10.1.4.1 Logging Machine Data by Using the ResourceAudit

If a resource is modified during processing, any nodes that also reference the resource may also be affected. The following logging procedure is recommended in order to track the resource modification and to insure consistency of the job:

1. Create a copy of the original resource with a new ID.
2. Modify the original resource to reflect the changes.
3. Insert a ResourceAudit element that references the modified original resource with the *first* ResourceLink and the copied resource with the second ResourceLink attribute.

The following example describes the logging of a modification of the media weight and amount. The JDF document before modification requests 400 copies of 80 gram media:

```
<JDF … >
  <ResourceLinkPool>
    <MediaLink rRef="RLink" Usage="Input" Amount="400"/>
  </ResourceLinkPool>
  <ResourcePool>
    <Media Weight="80" ID="RLink" Amount="400" (…)/>
  <ResourcePool/>
</JDF>
```

The JDF after modification specifies that 421 copies of 90-gram media have been consumed:

```
<JDF … >
  <ResourceLinkPool>
```

```
     <MediaLink rRef="RLink" Usage="Input" Amount="400"/>
<!—note that the ResourceLink has not changed -->
  </ResourceLinkPool>
  <ResourcePool>
    <Media Weight="80" ID="RPrev" Amount="400" (…) /> <!—Copy of the original re-
source-->
    <Media Weight="90" ID="RLink" Amount="421" (…)/> <!—modified resource-->
  <ResourcePool/>
  <AuditPool>
    <ResourceAudit (…)>
      <MediaLink rRef="RLink" Usage="Input" Amount="421"/>
      <MediaLink rRef="RPrev" Usage="Input" Amount="400"/>
    </ResourceAudit>
  </AuditPool>
</JDF>
```

3.10.1.4.2 Logging Changes in Product Descriptions by Using the ResourceAudit

ResourceAudit elements may also be used to store the original intent resources of a product specification in a change order or request for requote. The mechanism is the same as above. The following example shows the structure of a MediaIntent with Option partitions, where a late change of options from Option1 (80 gram paper) to Option2 (90 gram paper) is requested.

```
<JDF … >
  <ResourceLinkPool>
    <MediaIntentLink rRef="id" Usage="Input">
      <Part Option="Option2"/>
    </MediaIntentLink>
  </ResourceLinkPool>
  <ResourcePool>
    <MediaIntent PartIDKeys="Option" (…)>
      <!— the common MediaIntent resource details -->
      <MediaIntent Option="Option1" (…)>
        <Weight Preferred="80"/>
      </MediaIntent>
      <MediaIntent Option="Option2" (…)>
        <Weight Preferred="90"/>
      </MediaIntent>
    </MediaIntent>
  <ResourcePool/>
  <AuditPool>
    <ResourceAudit (…)>
      <!— the actual MediaIntent resource link -->
      <MediaIntentLink rRef="id" Usage="Input">
        <Part Option="Option2"/>
      </MediaIntentLink>
      <!— the original MediaIntent resource link -->
      <MediaIntentLink rRef="id" Usage="Input"/>
        <Part Option="Option1"/>
      </MediaIntentLink>
    </ResourceAudit>
  </AuditPool>
</JDF>
```

3.10.1.5 Created

This element allows the creation of a JDF node or resource to be logged. If the element refers to a JDF node, it can be located in the AuditPool element of the node that has been created or in any ancestor node. If the element refers to a resource it must be located in the node where the resource resides so that the spawning and merging mechanism can work effectively.

Table 3-35 Contents of the Created element

Name	Data Type	Description
ref ?	IDREF	Represents the ID of the created element. Defaults to the ID of the local JDF node.

3.10.1.6 Modified

This element allows any modifications affecting a JDF node, such as changes made to the NodeInfo element or CustomerInfo element, to be logged. Changes that can be logged by other audit element types, such as resource changes, must not use this common log entry. The modification can be described textually by adding a generic Comment element to the Modified element. The location of the element in the node tree is the same as the location of the corresponding Created element.

Table 3-36 Contents of the Modified element

Name	Data Type	Description
jRef ?	IDREF	The ID of the modified node. The modified element resides in the modified node. Defaults to the ID of the local JDF node.

3.10.1.7 Spawned

This element allows a job that has been spawned to be logged in the AuditPool of the parent node of the spawned job-part or in the AuditPool of the node that has been spawned in case of spawning of individual partitions. For details about spawning and merging, see Section 4.4 Spawning and Merging.

Table 3-37 Contents of the Spawned element

Name	Data Type	Description
Independent ?	boolean	Declares that independent jobs that have previously been merged into a big job are spawned. If it is set to true, the attributes jRefDestination, *rRefsROCopied* and *rRefsRWCopied* have no meaning and should be omitted. Default = *false*
jRef	IDREF	ID of the JDF node that has been spawned.
jRefDestination ?	NMTOKEN	ID of the JDF node to which the job has been spawned.[3] This attribute must be specified in the parent of the original node if independent jobs are spawned.
NewSpawnID New in JDF 1.1	NMTOKEN	Copy of the *SpawnID* of the newly spawned node. Note that a Spawned audit may also contain a *SpawnID* attribute, which is the *SpawnID* of the node that this audit is being placed into prior to spawning.
rRefsROCopied ?	IDREFS	List of IDs separated by whitespace. Identifies the resources copied to the ResourcePool element of the spawned job during spawning. These resources should NOT be modified by the spawned job.
rRefsRWCopied ?	IDREFS	List of IDs separated by white spaces. Identifies the resources copied to the ResourcePool element of the spawned job during spawning. These resources may be modified by the spawned job and must be copied back into their original location by the merging agent. Resource copying is required if resources are referenced simultaneously from spawned nodes and from nodes in the original JDF document.
Status ? New in JDF 1.1	enumeration	*Status* of the spawned node at the time of spawning. Allowed values are defined in Table 3-3 Contents of a JDF node, *Status*.
URL ? New in JDF 1.1	URL	Locator that specifies the location where the spawned node was stored by the spawning process.
Part *	element	Identifies the parts that were selected for spawning in case of parallel spawning of partitionable resources (see Section 4.4.3).

3.10.1.8 Merged

This element logs a merging event of a spawned node. For more details, see Section 4.4 Spawning and Merging.

---

[3] The data type is NMTOKEN and not IDREF because the attribute refers to an external ID.

Table 3-38 Contents of the Merged element

Name	Data Type	Description
Independent ?	boolean	Declares that independent jobs are merged into a big job for common production. If it is set to true, the attributes jRefSource and *rRefsOverwritten* have no meaning and should be omitted. Default = *false*
jRef	IDREF	ID of the JDF node that has been returned or merged.
jRefSource ?	NMTOKEN	ID of the JDF root node of the big job from which the spawned structure has been returned. [4]
MergeID New in JDF 1.1	NMTOKEN	Copy of the *SpawnID* of the merged node. Note that a Merged audit may also contain a *SpawnID* attribute, which is the *SpawnID* of the node that this audit is being placed into prior to merging.
rRefsOverwritten ?	IDREFS	Identifies the copied resources that have been overwritten during merging. Resources are usually overwritten during return if they have been copied during spawning with read/write access.
URL ? New in JDF 1.1	URL	Locator that specifies the location of the merged node prior to merging by the merging process.
Part *	element	Specifies the selected parts of the resource that were merged in case of parallel spawning and merging of partitionable resources (see Section 4.4.3).

3.11 JDF Extensibility

JDF is meant to be flexible and therefore useful to any vendor, as each vendor will have specific data to include in the JDF files. JDF is able to provide this kind of versatility by using the XML namespaces. This chapter describes how JDF uses the XML extension mechanisms.

3.11.1 Namespaces in XML

JDF Extensibility is implemented using XML Namespaces. The Namespaces in XML specification is found at http://www.w3.org/TR/REC-xml-names/.

XML namespaces are defined by *xmlns* attributes. A general example is provided below. The example illustrates how private namespaces are declared and used to extend an existing JDF resource by adding private attributes and a private element.

```
<JDF xmlns="http://www.CIP4.org/JDFSchema_1_1"
xmlns:foo="fooschema URI" … >
  …
  <SomeJDFDefinedResource name="abc"
foo:specialname="cba">
    …
    <foo:PrivateStuff type=""/>
    …
  </SomeJDFDefinedResource>
  …
</JDF>
```

Using Namespaces In JDF

It is *required* to define the default namespace in a JDF document, even if no non-JDF extensions are used. JDF may be defined either in the default namespace or in a qualified namespace.

Namespaces are inserted in front of attribute and element names. The associated namespace of element names with no prefix is the default namespace defined by the xmlns attribute. The associated namespace of attributes with no

---

[4] The data type is NMTOKEN and not IDREF because the attribute refers to an external ID.

prefix is that one of the element (see Appendix A.2 XML Namespace Partitions in the specification Namespaces in XML). All namespaces prefixes must be declared using standard *xmlns*:xxx attributes.

3.11.1.1 JDF Namespace
The official namespace URI for JDF Version 1.0 is: "http://www.CIP4.org/JDFSchema_1".
The official namespace URI for JDF Version 1.1 is: "http://www.CIP4.org/JDFSchema_1_1".
It is strongly recommended to use either the default namespace with no prefix or a prefix of "JDF" as the jdf namespace prefix.

3.11.1.2 JDF Extension Namespace
CIP4 defines an extension namespace where new features that are anticipated to be included in a future version of the specification are defined.
The official extension namespace URI for JDF Version 1.1 is: "http://www.CIP4.org/JDFSchema_1_1_X".
It is strongly recommended to use a prefix of "JDFX" as the jdf extension namespace prefix.

3.11.2 Extending Process Types
JDF defines a basic set of process types. Because JDF allows flexible encoding, however, this list, by definition, will not be complete. Vendors that have specific processes that do not fit in the general JDF processes and that are not combinations of individual JDF processes (see Section 3.2.3 Combined Process Nodes) can create JDF process nodes of their own type. Then the content of the *Type* attribute may be specified with a prefix that identifies the organization. The prefix and name must be separated by a single colon (':') as shown in the following example:

```
<JDF Type="myCompaniesNS:MyVeryImportantProcess" xmlns=
"http://www.CIP4.org/JDFSchema_1_1" xmlns:myCompaniesNS="my companies namespace URI" …
>
   …
</JDF>
```

The use of namespace prefixes in the *Type* attribute is for extensions only. Standard JDF process types must be specified without a prefix in the *Type* attribute or the *Types* attribute of a combined node.
If a process is simply an extension of an existing process, it is possible to describe the private data by extending the existing resource types. This is described in greater detail in the sections below.

Extensibility Caution

JDF's "Extensibility" simply means that you can add your own XML elements, attributes, and enumerations to a JDF application. Although JDF is quite extensive, odds are you'll find that your current databases and workflow systems use information elements that are unique to your client market or company … *they may have even been defined by your internal MIS staff.* CIP4 acknowledges that it can't define everything, nor should it prevent innovation by codifying everything in a static manner, and JDF's extensibility provides both printers and technology providers with the flexibility they need to make JDF a success.

However, if you or your technology vendors extend JDF, please do so with caution. JDF's success depends on the ability of MIS systems and JDF-enabled devices to write, read, parse, and use JDF. Extensions are *custom* integration applications and great care needs to be made to ensure that extensions made for one systems or device will not *jam* the JDF workflow or other JDF enabled systems and devices. If they use extensions to JDF, your technology providers should be able to provide you with a fully validated JDF schema and documentation that includes the use of their extensions. Extensions that are not documented, or that may not be disclosed to third parties for integration purposes, should be viewed skeptically.

Extending the NodeInfo and CustomerInfo nodes is achieved in a manner analogous to the extension of resources, which is described below. On the other hand, extending the direct contents of JDF nodes by adding new elements or attributes is discouraged.

3.11.3 Extending Existing Resources

All resources defined by JDF may be extended by adding attributes and elements using one's own namespace for these resource extensions. This is useful when the predefined resource types need only a small amount of private data added, or if those resources are the only appropriate place to put the data. The namespace of the resource extended must not be modified. However, the mechanism for creating new resources in a separate namespace is provided in the next section.

This does not mean that duplicate functionality may be added into these resource types. You must make sure to use the JDF-defined attributes and elements where possible and extend them with additional information that cannot be described using JDF-defined constructs. For example, it is not allowed to extend the RIP resource that controls the resolution with a foo:Resolution or foo:Res attribute that overrides the JDF defined resolution parameter (see attribute *Resolution* of resource **RenderingParams** in Section 7.2.112).

3.11.4 Extending NMTOKEN Lists

Many resources contain attributes of type NMTOKEN and some of these have a set of predefined, suggested enumerative values. These lists may be extended with private keywords. In order to identify private keywords, it is strongly suggested to prefix these keywords with a namespace-like syntax, i.e., a namespace prefix separated by a single colon (':'). Implementations that find an unknown NMTOKEN prefixed by a namespace prefix may then attempt to use the default value of that attribute. For instance, if a JDF instruction contains the following text:

```
<TrappingParams TrapEndStyle="HDM:FooBar" (…)/>
```

Based of the definition of **TrappingParams**, the best assumption is to use *TrapEndStyle* = *"Miter"*.

Example from TrappingParams

Name	Data Type	Description
TrapEndStyle ?	NMTOKEN	Instructs the trap engine how to form the end of a trap that touches another object. Possible values include: *Miter* *Overlap* Other values may be added later as a result of customer requests. Default = *Miter*

3.11.5 Creating New Resources

There are certain process implementations that have functionality that cannot be specified by the predefined Resource types. In these cases, it is necessary to create a new Resource-type element, which must be clearly specified using its own namespace. These resource types may only be linked to custom type JDF process nodes.

3.11.6 Future JDF Extensions

In future versions, certain private extensions will become more widely used, even by different vendors. As private extensions become more of a general rule, those extensions will be candidates for inclusion in the next version of the JDF specification. At that time the specific extensions will have to be described and will be included into the JDF namespace.

3.11.7 Maintaining Extensions

Given the mix of vendors that will use JDF, it is likely that there will be a number of private extensions. Therefore, JDF controllers must be prepared to receive JDF files that have extensions. These controllers can and should

Submit Your Extensions to CIP4

Writing JDF extensions? CIP4 encourages you to become part of the standard and submit your private extensions for review and possible inclusion in future versions of the JDF standard. Not only may adoption of extensions into the JDF standard help make it easier for customers to decide to buy your products, but CIP4 is also considering adopting a formal review process for extensions with future editions of the JDF standard; by participating in JDF's development now you could save time and customer confusion in the future.

ignore all extensions they don't understand, but under no circumstance are they allowed to remove these extensions when making modifications to the JDF. If they do, it will break the extensibility mechanism. For example, imagine that JDF Agent A creates a JDF and inserts private information for Process P. Furthermore, the information is only understood by agent A and the appropriate device D for executing P. If the JDF needs to be processed first by another Agent/Device C, and that process removes all private data for P, Process P will not be able to produce the correct results on device D that were specified by Agent A.

3.11.8 Processing Unknown Extensions

If a node is processed by a controller or device and it encounters an unknown extension in one of its input resources, the expected behavior depends on the current value of *SettingsPolicy*.

If *SettingsPolicy* ="*BestEffort*", a Notification audit element with *Class* = *warning* should be logged.

If *SettingsPolicy* ="*MustHonor*" the process must not continue and a Notification audit element with *Class* = *error* should be logged.

If *SettingsPolicy* ="*OperatorIntervention*" the process must stop and wait for an operator intervention and a Notification audit element with *Class* = *warning* should be logged.

3.11.9 Derivation of Types in XMLSchema

The XML Schema definition http://www.w3.org/TR/xmlschema-1/ describes a mechanism to create new types by derivation from old types. This is an alternative to extend or create new elements and is described in Section 4 of http://www.w3.org/TR/xmlschema-0/. This mechanism is not allowed to be applied to any elements defined by JDF because such new element types can only be understood by agents/devices that know the extension. The use of the derivation mechanism is allowed only for private extensions but not required.

Chapter 4 Life Cycle of JDF

Introduction

This chapter describes the life cycle of a JDF job, from creation through modification to processing. Information is provided about the spawning of individual aspects of jobs and in what way they are reincorporated into the job once the process is completed. Ancillary aspects of the life cycle, such as test running and error handling, are also discussed.

4.1 Creation and Modification

The life cycle of a JDF job will likely follow one of two scenarios. In the first scenario, a job is created all at once, by a single agent, and then is consumed by a set of devices. More often, however, a job is created by one agent and is then transformed, or modified, over time by a series of other agents. This process may require specification of product intent, which is defined in Section 4.1.1, below.

Jobs can be modified in a variety of ways. In essence, any job is modified as it is executed, since information about the execution is logged. The most common instance of modification of a JDF job, however, occurs during processing, when more detailed information is learned or understood and then added along the way. This information may be added because an agent knows more about the processing needed to achieve some result specified in a JDF node than the original, creating agent knew. For example, one agent may create a product node that specifies the product intent of a series of pages. This product node may include information about the number of pages and the paper properties. Another node may then be inserted that includes a resource describing how the pages should be Ripped. Later, another agent may provide more detail about the RIP'ing process by appending optional information to the RIP parameter resource.

Regardless of where in the life cycle they are written, nodes and their required resources must be valid and include all required information in order to have a *Status* of *Ready* (in case of nodes) or *Available* (in case of resources). This restriction allows for the definition of incomplete output resources. For example, a URL resource without a file name may be completed by a process. On the other hand, it is impossible to define a valid and executable node with insufficient input parameters.

Once all of the inputs and parameters for the process requested by a node are completely specified, a controller can route the JDF job containing this node to a device that can execute the process. When the process is completed, the agent/controller in charge of the device will modify the node to record the results of the process.

4.1.1 Product Intent Constructs

JDF jobs, in essence, are requests made by customers for the production of quantities of some product or products. In other words, a job begins with a particular goal in mind. In JDF, product goals are often specified by using a construct known as product intent, represented by intent resources. In contrast to process resources that define precise values, intent resources allow ranges or sets of preferred values to be specified. Resources of this kind include **FoldingIntent**, **ColorIntent**, **MediaIntent**, and **ShapeCuttingIntent**, all of which are described in Chapter 7 Resources.

The product intent of a job is like a plan of action. The plan may be extremely vague, detailing only the general goal, or it may be very specific, stipulating the specific requirements inherent in meeting that goal. Product intent may be defined for an end product about which little is known or about which the processing details for the job are entirely unknown. Product intent constructs also allow agents to describe jobs that comprise multiple product components, and that may share some parts.

> **Product Intent**
>
> "Product Intent" is another way of saying "Job Specifications." Rather than describing how a job will be made, "Product Intent" describes what a job (or some aspect of a job) will look like when it is completed. "Product Intents" may initiate with the customer and in rather vague terms and they may be later flushed out or completed by a printer's customer service representative, estimating department or production planners.

Product intent is defined by the initiating agent of a job. It is not required, however. Many JDF jobs are written with full knowledge of the necessary processes, and are therefore comprised entirely of the various kinds of process

nodes described in Sections 3.2.1, 3.2.2, and 3.2.3. Any job that specifies product intent, however, must include nodes whose *Type* = *Product*. This representation is described in the following section.

4.1.1.1 Representation of Product Intent

The product description of a job is a hierarchy of *Product* nodes, and the bottom-most level of the product hierarchy represents portions of the product that are each homogeneous in terms of their materials and formats. All nodes below these *Product* nodes begin specifying the processes required to produce the products.

Product nodes are required to contain only one thing, and that is a resource that represents the physical result specified by the node. This resource is generally a **Component**. In addition, somewhere in the hierarchy of product nodes, it is a good idea to include an intent resource to describe the characteristics of the intended product. Although these are the only resources that should occur, product nodes can contain multiple resources. For example, some *ResourceTypes*, such as **MediaIntent** and **LayoutIntent,** are defined to provide more general mechanisms to specify product intent.

In some cases, more than one high level product node will use the output of a product node. These high level nodes represent the combination of homogeneous product parts. In this case, the *Amount* attribute of the ResourceLinks that connect the nodes will identify how the lower level product is shared.

4.1.1.2 Representation of Product Binding

Some product intent nodes, such as **BindingIntent**, define how to combine multiple products. To accomplish this, the respective **Component** resources must be labeled according to their usage. For example, the *Cover* and *Insert* attributes use the *ProcessUsage* attribute of the respective resource links. For more information about product intent, see Section 3.2.1 Product Intent Nodes.

4.1.2 Defining Business Objects Using Intent Resources

Business objects like requests for quote, quote, invoice, etc. need to reference processes at a level that is well represented by product intent nodes. It is assumed that business object metadata such as financial information, business document type, customer information, etc. is defined by an XML envelope that contains JDF as a job description. If this is not the case, the business related metadata may be placed into the BusinessInfo element of the NodeInfo element of the root JDF and the customer related data may be placed into the CustomerInfo element of the root JDF.

PrintTalk Implementation

A PrintTalk implementation guide can be found at http://www.printtalk.org/implementation.html

This section sketches the usage of JDF in an eCommerce environment using the business object model that was defined by the PrintTalk www.PrintTalk.org consortium.

The following table describes the individual business objects and their relationships. Object Type defines the name of the XML element that defines the metadata. All object types are inherited from the abstract PrintTalk Request element. References defines the business objects that are responded to when generating the business object and buyer-provider arrow defines the direction of the transaction.

Table 4-1. Business Objects as defined by PrintTalk

Object Type	Description	References	Direction
Request for Quote (RFQ)	Initiated by a buyer to a print supplier. It may instigate a new product process or it may supersede an existing RFQ. The Change Order and Request for Requote variations are included within Request for Quote.	None, Quote, Confirmation	B→P
Quote	Normally sent in response to a RFQ. The Requote and Change Order Quote variations are included within Quote. A Quote may supersede an existing Quote before the Print Buyer has answered with a RFQ or an Order.	RFQ, PO, Confirmation	B←P
Purchase Order	Typically sent as a response to a quote, but may	None, Quote, Confirmation	B→P

Object Type	Description	References	Direction
	be the initial document in a well defined buyer / print supplier relationship or when ordering finished goods items. The Change Order variation is included within Purchase Order. An order may supersede an existing Order prior to the Print Provider having confirmed it.		
Order Confirmation	Sent by the print supplier to the buyer acknowledging receipt of the purchase order. It may contain information about expected due dates and final pricing that were undetermined at the time of the quote.	PO	B←P
Cancellation	Cancels a complete job. If only parts of a job should be cancelled, one must send a new RFQ, Quote, or PO. In case of canceling parts of a confirmed order the Change Order variations of these Business Objects must be sent.	RFQ, Quote, PO, Confirmation	B↔P
Refusal	Used to explicitly decline a Business Object sent by the counter party. Alternatively, the non-accepted Business Object expires.	RFQ, Quote, PO	B↔P
Order Status Request	Generated anytime one party requests status from another party.	Confirmation	B↔P
Order Status Response	An Order Status Response can be sent as a response to an Order Status Request or it can be sent automatically.	Confirmation, Order Status Request	B↔P
Proof Approval Request	Provides a transport for proofing from supplier to buyer. This may contain MIME data or a URL where the proof is located.	Confirmation	B←P
Proof Approval Response	Contains buyer's approval or denial of a proof.	Proof Approval Request	B→P
Invoice	Typically sent once the job is shipped, but can also be sent several times, when certain milestones during production are reached. May include additional charges or discounts.	Confirmation, Cancellation	B←P

In the following figure the workflow of these business objects is partly illustrated in a simplified manner. See the PrintTalk specification at www.printtalk.org for a complete picture.
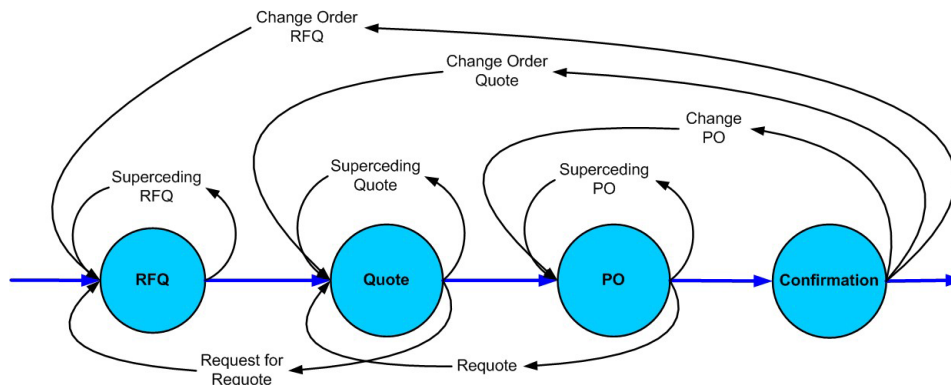
Figure 4.1 Simplified PrintTalk workflow (negotiation phase)

The node that defines an RFQ must contain one or more DeliveryIntent resources that define the amounts and methods of delivery. The *Usage* of the ResourceLinks is *Input,* its *Type* is "*Product*" and the Business object is an RFQ.

The examples quoted in this section use an object model as defined by PrintTalk with the business objects defined in BusinessInfo. This does not preclude the use of other eCommerce systems. The following examples show equivalent PrintTalk and pure JDF document text. The <mark>highlights</mark> show the respective position of an RFQ.

PrintTalk example

```
<PrintTalk>
  <Header>
   Standard CXML header
  </Header>
  <Request>
    <RFQ AgentID="Lara" RequestDate="2002-04-05T1700-0800" Expires="2002-04-15T1700-0800" Esti-
mate="false" AgentDisplayName="Lara Garcia-Daniels" Currency="EUR" BusinessID="RFQ_ID">
      <JDF ID="ScreenTest" Type="Product" JobID="ScreenJob" Status="Waiting" Version="1.1"
xmlns="http://www.CIP4.org/JDFSchema_1_1">
        <NodeInfo LastEnd="2000-12-24T06:02:42+01:00"/>
      (…)
      </JDF>
    </RFQ>
  </Request>
</PrintTalk>
```

Equivalent pure JDF Example

```
<JDF ID="ScreenTest" Type="Product" JobID="ScreenJob" Status="Waiting" Version="1.1"
xmlns="http://www.CIP4.org/JDFSchema_1_1">
  <NodeInfo LastEnd="2000-12-24T06:02:42+01:00">
    <BusinessInfo>
      <RFQ AgentID="Lara" RequestDate="2002-04-05T1700-0800" Expires="2002-04-15T1700-0800" Esti-
mate="false" AgentDisplayName="Lara Garcia-Daniels" Currency="EUR" BusinessID="RFQ_ID"/>
    </BusinessInfo>
  </NodeInfo>
 (…)
</JDF>
```

4.1.3 Specification of Delivery of End Products

A job may define one or more products and specify a set of deliveries of end products. To accomplish this, a node of *Type* = *Product* is created to define each delivery mode to be made. A delivery contains a set of drops, which in turn contain a set of drop items. Each drop has a common delivery address and each package contains the amount of an individual **Component** or **ComponentRef** that is to be delivered to this address. Quote generation as defined in the previous chapter includes the specification of delivery addresses. For more information, see section 6.2.4 **Delivery**.

4.1.4 Specification of Process Specifics for Product Intent Nodes

Product intent nodes are designed to represent a customer's view of the product. In some instances, a knowledgeable customer may want to specify production details that are only available in JDF process resources for a given product. Examples include scanning or screening parameters. This customer will still have no knowledge or control of the process workflow.

Individual JDF nodes can be inserted into a product intent node. These nodes will contain the requested process resource definitions as input resource links. The *Status* attribute of these resources should be "*Incomplete*". No output resources should be defined. In other words the actual specification of the process workflow should be left undefined. The application that sets up the actual workflow can then use these resource templates as a starting point for defining the process. The following example shows how an ellipse spot function is requested within a simple product description. The JDF node in <mark>yellow highlight</mark> defines the screening parameters of the product.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20001106181236" Type="Product" JobID="HDM20001106181236"
Status="Waiting" Version="1.0">
```

```
<ResourcePool>
    <Component ID="Link0003" Class="Quantity" Amount="10000"
Status="Unavailable" DescriptiveName="complete 16-page Brochure"/>
    <LayoutIntent ID="Link0004" Class="Intent" Status="Available">
      <Dimensions Range="576 720~648 864" DataType="XYPairSpan" Pre-
ferred="612 792"/>
      <Pages DataType="IntegerSpan" Preferred="16"/>
    </LayoutIntent>
    <MediaIntent ID="Link0005" Class="Intent" Status="Available" PartID-
Keys="Option">
      <FrontCoatings DataType="NameSpan" Preferred="None"/>
      <MediaIntent Option="1">
        <FrontCoatings DataType="NameSpan" Preferred="Glossy"/>
      </MediaIntent>
      <BackCoatings DataType="NameSpan" Preferred="None"/>
    </MediaIntent>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="Link0003" Usage="Output"/>
    <LayoutIntentLink rRef="Link0004" Usage="Input"/>
    <MediaIntentLink rRef="Link0005" Usage="Input"/>
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="Rainer's JDFWriter 0.2000" TimeStamp="2000-11-
06T18:12:36+01:00"/>
  </AuditPool>
  <JDF ID="Link0006" Type="Screening" Status="Waiting">
    <ResourcePool>
      <ScreeningParams ID="ScreenID" Class="Parameter" Status="Incomplete">
        <ScreenSelector SpotFunction="Ellipse" ScreeningFamily="My favorite
screen"/>
      </ScreeningParams>
    </ResourcePool>
    <ResourceLinkPool>
      <ScreeningParamsLink rRef="ScreenID" Usage="Input"/>
    </ResourceLinkPool>
  </JDF>
</JDF>
```

4.2 Process Routing

A controller in a JDF workflow system has two tasks. The first is to determine which of the nodes in a JDF document are executable, and the second is to route these nodes to a device that is capable of executing them. Both of these procedures are explained in the sections that follow.

In a distributed environment with multiple controllers and devices, finding the right device or controller to execute a specific node may be a non-trivial task. Systems with a centralized, smart master controller may want to route jobs dynamically by sending them to the appropriate locations. Simple systems, on the other hand, may have a static, well defined routing path. Such a system may, for example, pass the job from hot folder to hot folder. Both of these extremes are valid examples of JDF systems that have no need for additional routing metadata.

In order to accommodate systems between these extremes, the NodeInfo element of a node contains optional *Route* and *TargetRoute* attributes that let an agent define a static process route on a node-by-node basis. If no *Route* or *TargetRoute* attribute is specified and if a controller has multiple options where to route a job, it is up to the implementation to decide which route to use.

The controller or device reading the JDF job is responsible for processing the nodes. A device examines the job and attempts to execute those nodes that it knows how to execute, whereas a controller routes the job to the next controller or device that has the appropriate capabilities.

4.2.1 Determining Executable Nodes

In order to determine which node should be executed, the controller/device uses the following procedures:

1. First, it searches the JDF document for node types it can execute by comparing the *Type* attribute of the node to its own capabilities, and by determining the *Activation* of the nodes. It should also verify that the *Status* of the node is either *Waiting* or *Ready*.

2. The controller/device may then determine whether no resources have a *Status* of *Incomplete* or a *SpawnStatus* of *SpawnedRW*. It should also determine whether all of the input resources of the respective nodes have a *Status* of *Available* and that all processes that are attached through pipes are ready to execute. A controller may optionally skip these checks and expect the lower level controller or device that it controls to perform this step and return with an error if it fails.

3. Finally, if scheduling information is provided in the NodeInfo element, the specified start and/or end time must be taken into account by the executing device. If no process times are specified, it is up to the device in charge of queue handling to execute the process node.

The node will go through various stati during its life time as is described in the following diagram:
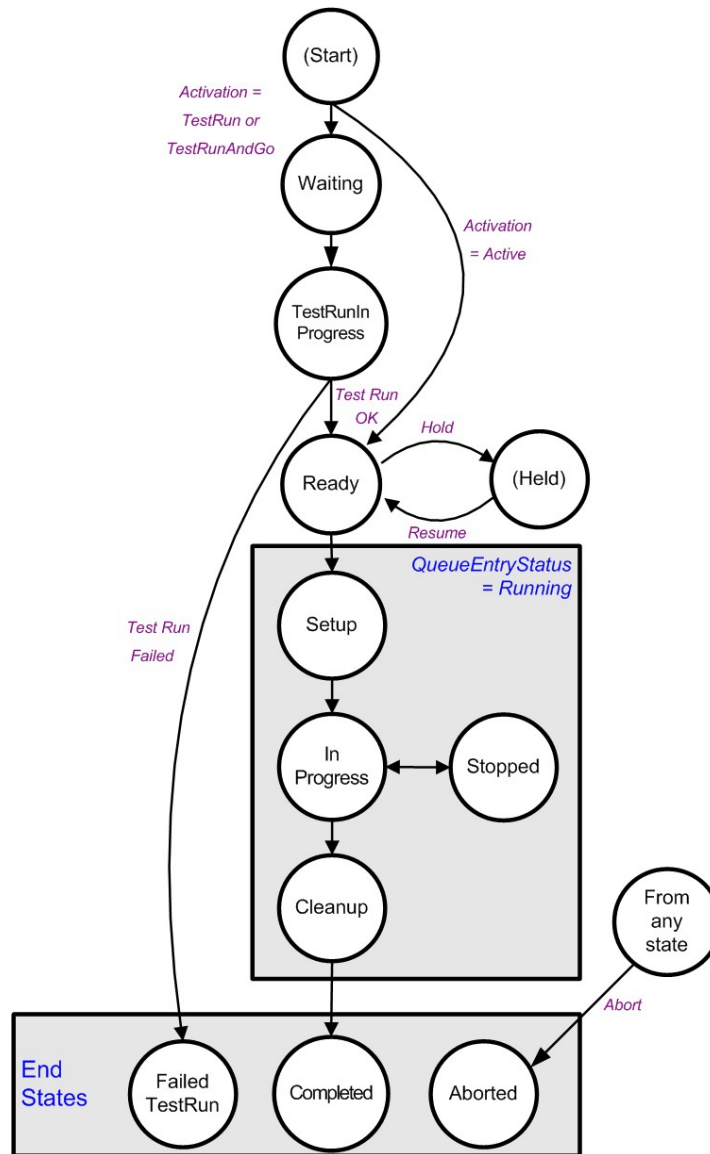
Figure 4.2 Life Cycle of a JDF node

4.2.2 Distributing Processing to Work Centers or Devices

JDF syntax supports two means of distributing processes to work centers or devices. Its first option is to use a "smart" controller that has the ability to parse a JDF job and identify individual processes or process groups that may be distributed to a particular work center or device. This smart controller may use spawning and merging facilities to subdivide the job ticket and pass specific instructions to a work center or device.

The second option, which is applicable when the controller being used isn't "smart," is to employ a simple controller implementation that routes the entire job to each workcenter or device, thus leaving it up to the recipient to determine which processing it can accomplish. For this option to work, each JDF-capable device must be able to identify process nodes it is capable of executing. Furthermore, each device must have sufficient JDF-handling capabilities to identify processes that are ready to run.

4.2.3 Device / Controller Selection

The method used to determine which is the appropriate device or lower level controller to use to execute a given node depends greatly on the implemented workflow being used. Although JDF provides a method for storing routing information in the *Route* attribute of the NodeInfo element of a node, it does not prescribe any specific routing methods. However, some of the tools available to figure out alternative workflows are described below.

Knowledge of the capabilities of lower level controllers/devices either may be hard-wired into the system or gained using the KnownJDFServices message. Since JDF does not yet provide mechanisms to determine whether a given device is capable of processing a node without actually performing a test run, a controller must either have a priori knowledge of the detailed capabilities of devices that it controls or it must perform a test run to determine whether a device is capable of executing a node. Furthermore, in addition to the explicit routing information in the *Route* attribute of the NodeInfo element of a node, JDF may contain implicit routing information in the form of **Device** implementation resources.

JMF defines the KnownControllers query to find controllers and the KnownDevices query to find devices that are controlled by a controller. The information provided by these queries can be used by a controller to infer the appropriate routing for a node. In a system that does not support messaging, this information must be provided outside of JDF.

4.3 Execution Model

JDF provides a range of options that help controllers tailor a processing system to the needs of the workflow and of the job itself. The following sections explain the ways in which controllers execute processes using these various options.

The processing model of JDF is based on a producer/consumer model, which means that the sequencing of events is controlled by the availability of input resources. As has been described, nodes act both as producers and consumers of resources. When all necessary inputs are available in a given node, and not before, the process may execute. The sequence of processing, therefore, is implied by the chain of resources in which the output resources of one node become the input resources of a subsequent node.

JDF supports four kinds of process sequences: serial processing, overlapping processing, parallel processing, and iterative processing. All four are described in the following sections.

4.3.1 Serial Processing

The simplest kind of process routing, known as serial processing, executes nodes sequentially and with no overlap. In other words, no nodes are executed simultaneously. Once the process has acted upon the resource in some way, the resource availability is described by the *Status* attribute of the resource, as described above. When the process state is *Ready* or *Waiting*, the process can begin executing.

In a workflow using serial processing, the controller is responsible for comparing the actual amount available with the specified amount in the corresponding PhysicalLink element to determine whether or not the input resource can be considered available. If no amount is specified in the PhysicalLink, the process is assumed to consume the entire resource.
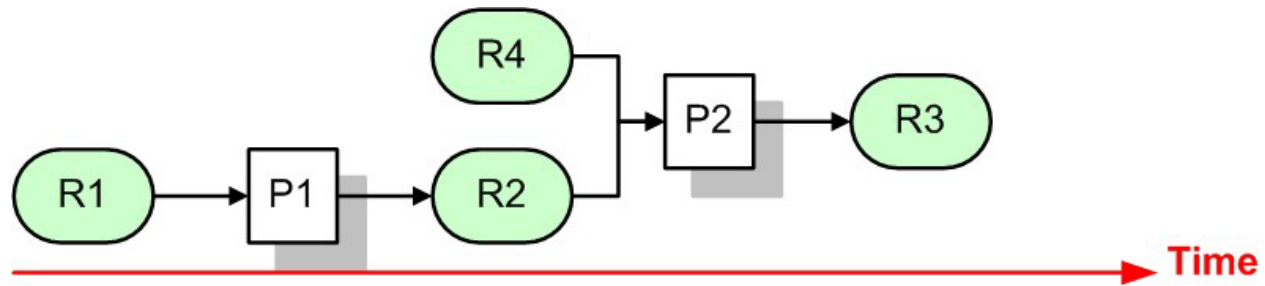
Figure 4.3 Example of a simple process chain linked by resources

Figure 4.3 depicts a simple process chain that produces and consumes *Quantity* resources and uses an implementa-
tion resource. The resources R1, R2, and R3 represent *Quantity* resources. Process P1 consumes resource R1 and
produces resource R2. R2 is then completely consumed by P2, which also requires the implementation resource R4
for processing. Process P2 uses these two resources and produces resource R3. All of this is accomplished along a
linear time axis.

   Table 4-2 shows the value of the *Status* attribute of each of the resources and processes used in Figure 4.3.
The time axis runs from left to right both in Figure 4.3 and in Table 4-2. Note that no process may execute until all
resources leading up to that process are in place. In other words, the job executes serially and sequentially. For
more information about the values of the *Status* attribute of resources, see Table 3-11. For more information about
the values of the *Status* attribute of processes, see Table 3-3.

Table 4-2 Examples of resource and process states in the case of simple process routing

Object Status	before running P1	during running P1	after running P1, be-fore P2	during P2	after P2
resource R1	*Available*	*InUse*	*Unavailable*	*Unavailable*	*Unavailable*
resource R2	*Unavailable*	*Unavailable*	*Available*	*InUse*	*Unavailable*
resource R3	*Unavailable*	*Unavailable*	*Unavailable*	*Unavailable*	*Available*
resource R4	*Available*	*Available*	*Available*	*InUse*	*Available*
process P1	*Waiting* or *Ready*	*InProgress*	*Completed*	*Completed*	*Completed*
process P2	*Waiting* or *Ready*	*Waiting* or *Ready*	*Waiting* or *Ready*	*InProgress*	*Completed*

When the attribute *Amount* is used in connection with the quantifiable resources R1, R2, or R3 and their links, then
the controller must decide whether or not a resource is available by comparing the individual values. If the amounts
are used to define the availability, then the resource *Status* may be set to *Available* for all *Quantity* resources. Note
that when the value of the *Status* attribute of the resource is *Unavailable,* the resource is not available even if a suf-
ficient amount is specified.

   If amounts are specified in the resource element, they represent the actual available amount. If they are not
specified, the actual amount is unknown, and it is assumed that the process will consume the entire resource.
Amounts of PhysicalLink elements must be specified for output resources that represent the intended production
amount. The specification of the *Amount* attribute for input resources is not required, although it can be specified.
If the controller cannot determine the amounts, this constitutes a JDF content error, which is logged by error han-
dling. This process is described in Section 4.6 Error Handling.

   If a process in a serial processing run does not finish successfully, the final process status is designated as
aborted. In an aborted job, only a part of the intended production may be available. If this occurs, the actual pro-
duced amount is logged into the audit pool by a resource audit element.

4.3.2 Overlapping Processing Using Pipes

Whereas pipes themselves are identified in the resource that represents the pipe, pipe dynamics are declared in the re-
source links that reference the pipe. This allows multiple nodes to access one pipe, each of them with its own pipe
buffering parameters.

In some situations, resource linking is a continuous process rather than a chronological one. In other words, one process may require the output resources of another process before that process has completely finished producing them. The ability to accomplish this kind of resource transfer is known as overlapping processing, and it is accomplished with the use of a mechanism known as pipes. Pipes are considered to be **active** if any process linking to the pipe simultaneously consumes or produces that pipe resource.

Any resource may be transformed into a pipe resource. All that is required is that the *PipeID* attribute be specified in the resource. Pipes of quantifiable resources resemble reservoir tanks that hang between processes. Processes connected to the pipe via output links fill the tank with necessary resources, while processes connected via input links deplete it (see Figure 4.4). The level is controlled by the PhysicalLink attributes *PipeResume*, *PipePause*, *RemotePipeEndPause*, and *RemotePipeEndResume* (see Table 3-21). If none of them are specified, any produced *Quantity* may be immediately consumed by the consuming end of the pipe. The unit of the buffers is defined by the *Unit* attribute of the resource.

Pipe Resources

A pipe resource is simply an input to a process that can be exhausted and may be replenished. Examples may include rolls of paper feeding into a press, ink well levels, fountain solution, or even proofing stock loaded into a proofer.

Another type of pipe resource in every-day use is a "hot-folder" or "watched file." Hot folders are used to automate functions such as preflighting. When a file is saved to a hot-folder, the system knows to automatically apply a defined process to the new file. When the folder is empty the processing stops.

The two following diagrams show the ways in which pipes mediate between the process producing the resource and the process consuming the resource. The following optional attribute values are defined for pipes: *PipePartIDKeys*, *PipePause*, *PipeResume*, *RemotePipeEndPause*, and *RemotePipeEndResume*. The latter two—*RemotePipeEndPause* and *RemotePipeEndResume*—are use to control the level in context with pipe command messages which will be described in Section 4.3.2.2 Dynamic Pipes. The specified value of each of these attributes in any given node dictates the levels at which a pipe should resume or pause execution. Figure4.5 gives an example of a view on the dynamics of a pipe resource. The available level of the pipe resource, represented as R2, and the availability status of two entity resources, represented as R1 and R3, are changing along a consistent time line. Below the progressions of these resources is the status of two processes—P1 and P2. P1 represents the process producing the pipe resource and P2 represents the process consuming that resource. The resource status of a active pipe (here R2) is defined to be *Status* = *InUse* (see also Table 3-11).

Pipe Resource R2

Figure 4.4 Example of a Pipe resource linking two processes

Figure 4.4 is a view on the structure and Figure4.5 a view on the dynamics of the pipe example considered here. R1 represents an input resource for P1, which feeds into the intermediate pipe resource R2. Once the tank R2 is filled to the predetermined level, it is used as the input resource for P2, which in turn produces output resource R3.
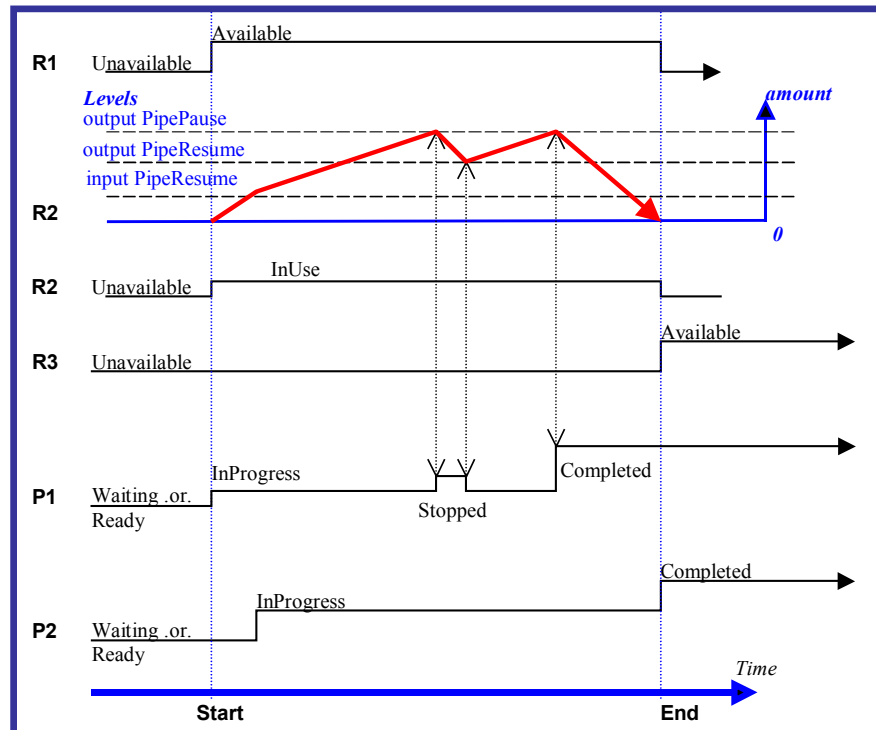
Figure4.5 Example of status transitions in case of overlapping processing

Resource linking through pipes is controlled through the specification of the *PipePause* and *PipeResume* attributes. The intended amount of a resource must be specified in advance in the output link. Whenever the level representing the available quantity of the pipe resource exceeds the *PipePause* level of the output link, the process P1 is halted (*Status* = *Stopped*) so that the process does not overproduce. Once the level falls below the *PipeResume* value, the process P1 resumes execution. P1 is completed when it has produced the intended amount. Once P1 has performed its task, the resources still in the pipe are consumed by the subsequent process without level control. In other words, after a process filling a pipe buffer has completed, pipe buffering becomes disabled.

Conversely, if the level representing the actual amount exceeds the *PipeResume* level of the input link, P2 can start or resume execution. If it falls below the *PipePause* level, P2 is halted (*Status* = *Stopped*) unless the intended amount of the pipe resource R2 has already been produced. Then the *PipePause* level is ignored and the pipe resource is completely consumed.

In the case of output links, the *PipeResume* value must be smaller than the *PipePause* value, whereas in the case of input links, the *PipeResume* value must be greater than the *PipePause* value. If *PipePause* is specified for an input or an output link and *PipeResume* is not specified, the related process may run into a deadlock state. In other words, the process stops and cannot resume execution automatically. Once a process is stopped under these circumstances it can only be resumed manually or by sending a pipe control message for resumption that allows interconnected execution control (halting and resumption of processes by pipe control messages is described in Section 5.5.3 **Pipe Control**). If the attributes *PipeResume* or *PipePause* of links to pipe resources are not specified, the controller is responsible when the linked processes start and stop in dependence of the level.

4.3.2.1 Pipes of Partitionable Resources

Pipes of partitionable resources may also define the granularity of the resources that are considered to be one part. To accomplish this, the *PipePartIDKeys* attribute must be specified in the appropriate ResourceLink element. For instance, a partitioned *ImageSetting* process may be defined for multiple sheet separations, but a complete set containing all separations of both sides of a single sheet should be sent to the pressroom as one pipe request. In this case, the value of the *PartIDKeys* attribute of the ExposedMedia resource would be *SheetName Side Separation* and the value of the *PipePartIDKeys* attribute of the resource link to the pipe would be *SheetName*.

4.3.2.2 Dynamic Pipes

In addition to abstractly declaring pipe properties, JMF provides pipe messages that allow dynamic control of pipes. Dynamic pipes can be used to model situations where the required amount of resources is not known beforehand but becomes known during processing. An example of this behavior is a long press run where new plates are required during a press run because of quality deterioration. The exact point in time where quality becomes unacceptable is not predetermined and may even vary from separation to separation. Dynamic pipes provide the flexibility to adjust to changing situations of this nature.

Dynamic pipes provide a *PipeURL* attribute that allows dynamic requests for a status change of the pipe while a process is executing. Dynamic requests use JMF pipe control messages (see Section 5.5.3 **Pipe Control**) sent to another controller whose URL address is specified by the *PipeURL* attribute of the respective resource link. Depending on the values of the resource link's *Usage* attribute, the following actions are possible:

- *Input* – The consumer sends a PipePull message to its *PipeURL* in order to request additional resources or a PipePause to halt production by the creator. The consumer sends a PipeClose message to the producer if the consumer does not require any further resources.

- *Output* – The creator sends a PipePush message to its *PipeURL* in order to deliver additional resources or a PipePause to halt consumption by the consumer.

When dynamic pipes are used—i.e., when the *PipeURL* attribute is specified—the pipe buffering parameters *RemotePipeEndResume* and *RemotePipeEndPause* define the buffering parameters of the remote (controlled) end. *PipeResume* and *PipePause*, meanwhile, define the buffering parameters of the local node as described in Section 4.3.2. The buffering parameters of a non-dynamic pipe may control the process that contains the resource link, whereas the buffering parameters of a dynamic pipe control the process at the other end of the pipe. The pipe control messages described later in Section 5.5.3 **Pipe Control** are designed to establish communication between processes at both ends of dynamic pipe, even if the corresponding processes are spawned separately.

The following table summarizes the actions to be taken when the buffer in a dynamic pipe reaches a certain level L:

J. 1 Actions generated when a dynamic-pipe buffer passes various levels

Controlling Pipe End	Situation	Message	Description
Output (creator)	L > *RemotePipeEndResume*	PipePush	Sufficient resources have been produced by the creator and are ready for delivery to the consumer.
Output (creator)	L < *RemotePipeEndPause*	PipePause	The consumer has consumed to the low water mark and must pause until a sufficient amount of resources have been produced.
Input (consumer)	L < *RemotePipeEndResume*	PipePull	More resources are requested from the creator and processing may continue by the consumer.
Input (consumer)	L > *RemotePipeEndPause*	PipePause	The creator has produced to the high water mark and must wait until a sufficient amount of resources have been consumed.

Dynamic pipes are initially dormant, and must be activated by an explicit request. Dynamic pipe requests may be initiated by both ends of the pipe. For example, a print process may notify an off-line finishing process when a certain amount is ready by sending a PipePush message, or the printing process may request a new plate by sending a PipePull message.

4.3.2.3 Comparison of Non-Dynamic and Dynamic Pipes

The resource link between non-dynamic pipes provides the buffering parameters for the process to which the link belongs. Therefore, many processes can link to the same pipe resource. Furthermore, each process has its own buffering parameters, whether it is a consumer or a producer. In order to control non-dynamic pipes, one master-controller must control all processes linked to the pipe resource.

In contrast, dynamic pipes provide a URL address to control a process at the other pipe end. Then the buffering parameters of the resource link control the process at the other end. In the case of dynamic pipes, no master-controller is required in order to control the pipe. Control is accomplished by sending pipe messages. If pipe resources are linked to multiple consumers or producers, such as two finishing lines that consume the output of one press one palette at a time, it is up to implementation to ensure consistency of the processes.

When using pipe resources, it is recommended that scheduling data for the process be specified only in the NodeInfo element of the parent node of the processes linked by pipe resources in order to avoid scheduling deadlocks. In Figure4.5, for instance, the actual start and end time of the corresponding parent of P1 and P2 are marked on the time axis.

4.3.3 Parallel Processing

While serial processing assumes that all resources will be produced and consumed in a linear fashion, and while overlapping processing uses multiple processes that work together to use and create resources, there are times when it makes sense to run more than one process simultaneously, creating a more multi-pronged workflow. This kind of process routing is known as parallel processing. Subsections of jobs are spawned off so that nodes may be executed individually and simultaneously by the appropriate devices. Once the processes are complete, the spawned nodes are merged back into the original job. The output resources of the merged nodes become inputs for later processes. For example, an insert may be produced independently of a cover, and both will be bound together later.

In parallel processing, processes can be run in a coordinated parallel fashion by using independent resources. An independent resource is a resource that is not shared between multiple processes. Implementation resources, for example, cannot be shared and are therefore always independent, and *Consumable* and *Quantity* resources can each be split to function as independent resources. Individual partitions of partitionable resources are independent and may be processed in parallel. Read-only resources, such as parameters, can be shared without any restrictions, and can therefore be used in read-only mode for parallel processing. Process chains created using independent resources are known as independent process chains.

Parallel processing can proceed in one of two ways. Either a controller may organize the JDF nodes in a way that allows it to initiate parallel processing or it can use the spawning-and-merging mechanism to field out chunks of the job to execute simultaneously. If a controller chooses the latter method, parent nodes that contain independent process chains can be spawned off and processed independently. For example, in order to improve production capacity, an agent may split consumable resources and create independent process chains in which each chain consumes its own resource part. Afterwards, the agent can submit one of the created job parts to a subcontractor and process the other part with its own facilities.

Parallel processing is used only to process multiple aspects of a job simultaneously; it is not used to process multiple copies of a JDF job. In other words, a job must not be copied and sent to different controllers for parallel processing. For more information about spawning of jobs, see Section 4.4 Spawning and Merging.

4.3.4 Iterative Processing

Some processes, especially in the prepress area of production, cannot be described as a serial or parallel set of process steps. Instead, a set of interdependent processes is iterated in a non-deterministic order. These processes are known as iterative processes. For example, an advertisement is laid out that requires a photographic image. During the layout phase, changes must be made to the color settings of the image, which is the reinserted to the layout. Changes such as these can be described in a high level fashion by defining a resource *Status* attribute of *Draft*. As long as an input resource to a process has a status of *Draft*, the *Status* of the output resource must not be *Available*.

The ResourceLink that links to a draft input resource must include a *DraftOK* attribute to state that a draft input resource is acceptable for a process. Thus a prepress layout process can be abstractly defined to work on draft resources until an acceptable output has been achieved, but the output PDL file must not be used for printing until it is *Available* and no longer designated as a *Draft*.

Iterative processes may be set up in a formal fashion using dynamic pipes to convey parameter change requests or in an informal way that assumes that the operators of the various processes have an informal communication channel. Both are described in greater detail below.

4.3.4.1   Informal Iterative Processing

Informal iterative processing does not require a complete redefinition of the required resources at every iteration. This kind of processing is generally used in a creative workflow, where a job is defined and gets refined in a series of steps until it is completed. The information about the changes is transferred through channels that bypass JDF. Nonetheless, the description of these processes in JDF is useful for accounting purposes, as the status of each process may be monitored individually.

The ResourceLink elements for informal processing contain an additional *DraftOK* attribute, but in all other ways they are identical to the ResourceLink elements used in simple sequential processing. Furthermore, the nodes run through the same set of phases as they would in sequential processing. Nodes are designated only as *Stopped* and not as *Completed* after being processed for an iterative cycle. They are marked as completed after their output resources lose their *Status* of *Draft.*

4.3.4.2   Formal Iterative Processing

In formal iterative processing, all ResourceLink elements between interacting processes are dynamic pipes. Every request for a new resource is initiated by a PipePush or PipePull message that contains at least one Resource element with the updated parameters. This resource is used by the process, and the resulting new output resource can be consumed by the requesting process. The *Status* of *Draft* can be removed from a resource by sending the creator a PipeClose message that has the optional *UpdatedStatus* attribute set to *Available*. A node can only reach a *Status* of *Completed* if it has no remaining draft resources. Another method to remove the draft status is to define a node for an *Approval* process that accepts draft resources as inputs and has non-draft resources representing the same entities as outputs.

4.3.5   Proofing and Verification

In many cases, it is desirable to ensure that an executed process or set of processes have been executed correctly. In the graphic arts industry this is verified by generating approvals and signing them. JDF allows modeling of the proof process and modeling of the verification processes by allowing an optional ApprovalSuccess input resource in any process. An **ApprovalSuccess** resource may only be set as *Available* if it has been signed by an authorized person.

If an approval fails and one or more processes that create the approved resource must be rerun, an agent must modify the job appropriately and resubmit it to the corresponding controllers and devices. All interchange resources from the first unsuccessful process to the approval must be designated as *Unavailable*.

4.4 Spawning and Merging

JDF spawning is the process of extracting a JDF subnode from a job and creating a new, complete JDF document that contains all of the information needed to process the subnode in the original job. Merging is the process of recombining the information from a spawned job part with the original JDF job, even after both documents have evolved independently. By using the mechanism for spawning and merging different parts of a job, it is possible to submit job parts to distributed controllers, devices, other work areas, or other work centers.

The JDF spawning-and-merging mechanism can be applied recursively, which means that subjobs that have already been spawned may in turn spawn other sub-subjobs, and so on. This does not mean, however, that a node may be respawned. If a node is spawned a second time, the previously submitted version must first be deleted and the spawning procedure must be applied again to the original node.

No matter how many job parts have been spawned, however, merging is realized by copying nodes back to their original location and synchronizing the appropriate resources. Therefore, each spawning must be logged in the job by the agent performing the actions that result in a spawned job. Furthermore, in order to avoid inconsistent JDF states after merging, each merging should be logged, or the appropriate spawn audit must be removed from the AuditPool element.

Figure 4.6, shows, schematically, the spawning and merging of a subjob, designated as P.b. The following three phases are defined on a the demonstrated time scale:

1.   The first phase occurs <u>before</u> the subjob is spawned off.

2. The second phase occurs <u>during</u> the spawn phase, when the spawned subjob is executed separately.

3. The third phase occurs <u>after</u> the spawned job has been merged back into the original job.
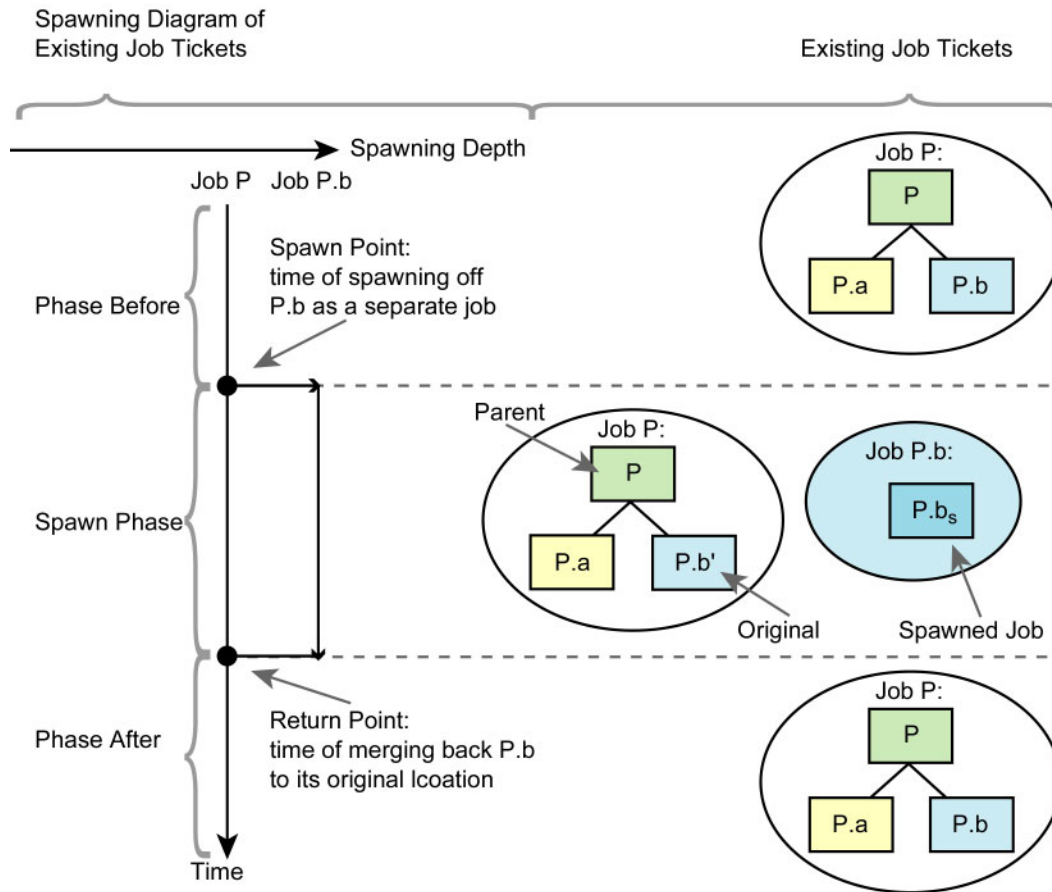
Figure 4.6 The spawning and merging mechanism and its phases

The three phases of the job part are bordered by the spawning point and the merging point. On a job scale, denoted as spawning depth in Figure 4.6, one job ticket exists during the phases before and after spawning, and the following two job tickets exist during the spawning phase: The job with the **parent** (P) of the **original** job part (P.b', also denoted as a subjob) that has been spawned; and the **spawned job** (P.b$_s$) itself.

This section provides examples that outline the various ways in which spawning and merging can be applied. The six following cases are considered in the next six sections:

1. Standard spawning and merging.
2. Spawning and merging with resource copying.
3. Parallel spawning and merging of partitioned resources.
4. Nested spawning and merging in reverse sequence.
5. Spawning and merging of independent job tickets.
6. Simultaneous spawning and merging of multiple nodes.

JDF can support any combination of the cases described, but these six represent a cross-section of likely scenarios. Case one is the simplest of all of the cases and is required in every instance of spawning and merging, regardless of the circumstances surrounding the process. Each subsequent case requires additional processing that builds upon the processing described in the cases that precede it.

4.4.1 Case 1: Standard Spawning and Merging

The actions described in this case must be applied in every spawning and merging process. All cases described in this chapter, as well as any other that may be invented, begin with these procedures.

Spawning

When spawning a JDF subnode, the JDF elements CustomerInfo and NodeInfo elements of the spawned job may be created and/or filled with the appropriate information (for details, see Sections 3.4 Customer Information and 3.5 Node Information). All resources that are referenced in the spawned node and its subnodes are located in the ResourcePool containers of the nodes in which they reside.

To indicate that a process has been spawned, the *Status* attribute of the original JDF node must be set to the value *Spawned* (see Table 3-3). The *Status* attribute of the spawned node remains unchanged.

A unique *SpawnID* attribute should be set in the spawned node and a copy of its value should be set in the *NewSpawnID* of the newly created Spawned audit. This simplifies bookkeeping of audits and merging in case a node is multiply spawned, either due to error conditions or in parallel with individual partitions. The value of *SpawnID* should also be appended to the *SpawnIDs* list of all spawned resources.

In order to identify all of the ancestors of job that has been spawned, an AncestorPool element is included in the root node every spawned job. This element contains an Ancestor element that identifies every parent, grandparent, great-grandparent, and so on of the spawned subnode. In this way, the family tree of every spawned node is tracked in an ordered sequence that allows an unbroken trace back through all predecessors. Consequently, the elements that comprise the AncestorPool of a spawned job must be copied into the AncestorPool element of the newly spawned job before the ancestor information of the previously spawned job is appended to the AncestorPool element of the newly spawned job. The last Ancestor element in each AncestorPool is the parent, the second-to-last the grandparent, and so on. The following code is an example of a family tree:

```
<AncestorPool>
  <Ancestor NodeID="p_01" FileName="file://grandparent.jdf"/>
  <Ancestor NodeID="p_02" FileName="file://parent.jdf"/>
</AncestorPool>
```

The complete ancestor information is required in order to merge back semi-finished jobs with nested spawns. If the last spawn is always merged first (LIFO) then knowing the direct parent is sufficient, as each parent will in turn know its own parent back to the original and a complete ancestor line may be inferred.

When a job is spawned, the action must be logged in the parent node of the spawned node in the original job. This is accomplished by creating a Spawned element with the *jRef* attribute set to the ID of the spawned JDF node. This Spawned element must be appended to the AuditPool container of the original parent node. If no AuditPool container exists in the parent node, one must be created for the purpose.

After a node has been spawned, it is legal although not necessary, to remove all contents of the spawned node in the original node except for the required attributes *ID, Status,* and *Type*. It is not, however, possible to undo the spawning operation without accessing the spawned node once the contents of the spawned node have been removed.

Merging

After processing, the spawned job must be merged back to its original location. Before this can occur, however, duplicate information contained in any elements that are not required for further processing (such as CustomerInfo or NodeInfo) may optionally be deleted by the agent executing the spawning and merging. Once this has been accomplished, the spawned node is copied to the location of the original node, completely overwriting the original node. The *Status* of the original node is then overwritten with the result.

To complete the merging process, the merging agent must add a Merged audit to the AuditPool (see Section 3.10 AuditPool). The *MergeID* of the Merged audit should be set to the value of the *SpawnID* attribute of the merged node. Furthermore, the AncestorPool container with all child elements must be removed and the value *SpawnID* of should be removed from the *SpawnIDs* attribute of the appropriate resources.

4.4.2 Case 2: Spawning and Merging with Resource Copying

Figure 4.7, shown below, represents an example of a job that requires that resources be copied during spawning. In this job, the nodes B_1 and B_2 are linked to the same resource, which is localized in the resource pool of an ancestor node, denoted as node A. This node is the parent node.
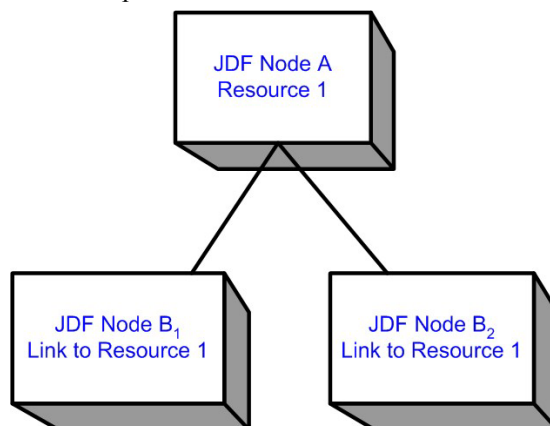
Figure 4.7 JDF node structure that requires resource copying during spawning and merging

When node B_1 is spawned, its resources must also be duplicated. To accomplish this, the affected resources must be copied to the spawned job and purged during merging, a process that is described below.

4.4.2.1 Spawning of Resources with Inter-Resource Links

Resources may be linked to a node by three mechanisms:

- Explicit links defined by a ResourceLink in the ResourceLinkPool of the node.
- Implicit links defined by the *rRefs* attribute of linked Resources. Implicit links are recursive.
- Implicit links defined by the *rRefs* attribute of the AuditPool, CustomerInfo or NodeInfo element of the node.

A spawning or merging agent must resolve all of these links by copying any non-local resources into the local ResourcePool.

Spawning

Spawning begins as it did in Case 1. The affected resources must then be copied to the resource pool of the spawned job. The copied resources retains the same *ID* values as the original resources. These resources can be spawned for read-only access, which allows multiple simultaneous spawning of one resource, or for read/write access, in which case a resource may only be spawned one time. The read/write spawning of a resource locks the resource in the original file in order to avoid conflicts that result from simultaneous modification or reading and modification of a resource. The *SpawnStatus* attribute of the original resource must be set to *SpawnedRW* (which stands for "spawned read/write ") or *SpawnedRO* (which stands for "spawned read-only ") to indicate that the resource is spawned. In other words, a copy of the resource is spawned together with the spawned job. Read/write access effectively locks the original resources, just as if the attribute *Locked* = *true*[1] were present. If a resource is spawned as read-only, it is not a good idea to modify the original resource that remains in the parent job ticket as this may lead to inconsistencies. The *Locked* attribute of spawned resources that are copied read-only should also be set *true*. Furthermore, the value of the *ID* attribute of each copied resource must be appended to the appropriate *rRefsROCopied* or *rRefsRWCopied* values of the Spawned element that resides in the AuditPool of the parent node.

Merging

Merging begins as it did in Case 1. Then, if resources have been copied for spawning, they must be purged after merging. Read-only resources may simply be deleted in the spawned node before merging. If the original resource and the spawned resource are not identical, however, a JDF content error should be logged by a Notification element of *Class* = *Error* (see Section 4.6 Error Handling). Read/write resources must be copied into their original location, completely overwriting the original resource. The *ID* attributes of the overwritten resources must be specified in the *rRefsOverwritten* attribute of

---

[1] Usually resources become locked (*Locked* = *true*) if they are referenced by audit elements (see also Section 3.10 AuditPool).

the Merged element. The Merged element is then inserted into the AuditPool container of the parent during the usual merging procedure, which is shown as the return point in the spawning diagram.

4.4.3 Case 3: Parallel Spawning and Merging of Partitioned Resources

In many cases, it is desirable to define a parallel workflow for partitioned resources. This is modeled by spawning a node that defines the process for each part that is to be processed individually.

Spawning

Spawning begins as it did in Case 1 or Case 2. Then the spawning agent must loop over all ResourceLinks and en-sure that the appropriate Part element or elements exist in any resources in the spawned ticket, where only the indi-vidual parts are required. This is accomplished either by adding Part elements if none exist in ResourceLinks of the parent node or by modifying the copies of existing Part elements. Part elements must be included in all Re-sourceLinks that point to resources that are spawned with write access. Part elements may be included in Resour-ceLinks that point to resources that are spawned with read only access, e.g. Physical resources where only a part is provided to a process as input. In addition, copies of the Part elements are appended to the Spawned audit element. The *Status* of any partitioned resource is defined individually for each partition. The *Status* of the parent node is set to "*Pool*" and a StatusPool is generated with the appropriate information. The PartStatus that describes the newly spawned node is set to "*Spawned*".

The spawning procedure described in this section can be performed iteratively for multiple parts, effectively generating one Spawned audit element and one PartStatus in the StatusPool per part. The Spawned and Merged audit elements are not placed in the parent node of the node to be spawned, but rather in the node itself.

Merging

After an individual partitioned spawned node has been processed, it is merged back to the parent as was described in Case 1. In addition, a copy of the Part elements of the corresponding **Spawned** audit is appended to the Merged element and any read/write resources are merged into their appropriate parts. The *Status* of the spawned node is copied into the appropriate PartStatus in the StatusPool.
An example of partitioned Spawning and Merging can be found in K.3 Spawning and Merging.

4.4.4 Case 4: Nested Spawning and Merging in Reverse Sequence

Figure 4.8 shows an example of nested spawning and merging in reverse sequence. Process A spawns node B, and node B spawns node C. Even if B is merged back to A for any reason before C is merged back to B, C still contains the infor-mation of its grandparent in the AncestorPool element. In this way, C can trace back its ancestors and find the localiza-tion of its parent, node B, in node A even though the spawned job, with B as root node, has already been deleted.
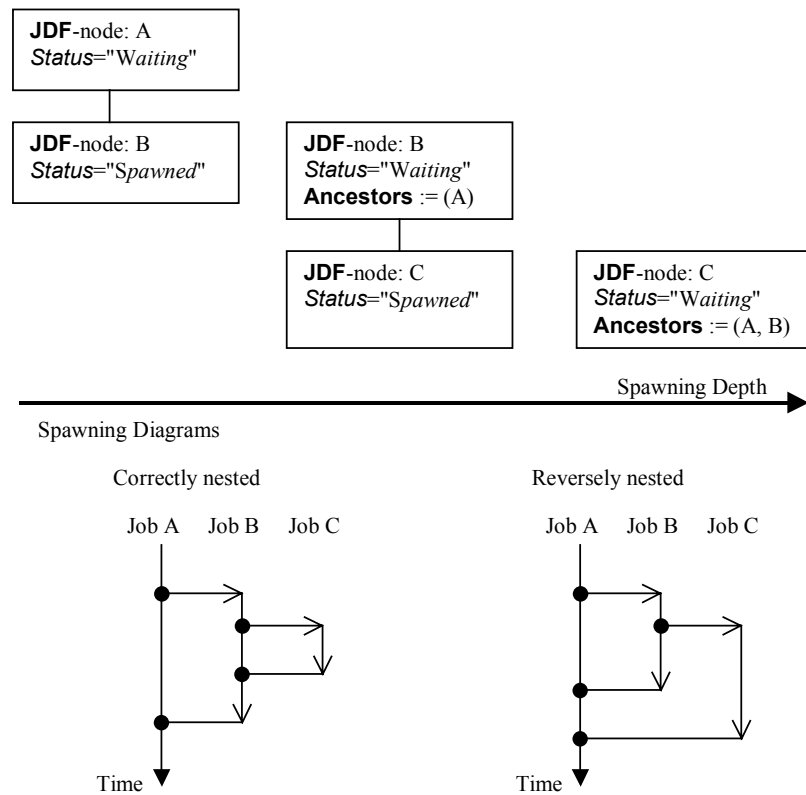
Figure 4.8 Example for a JDF node structure with nested spawning

4.4.5 Case 5: Spawning and Merging of Independent Jobs

Note that Spawning and Merging of Independent Jobs is under development and subject to major changes in a future release of this specification.

It is useful to spawn and merge independent jobs in situations where the execution of separate, independent small jobs is not efficient in a commercial sense. Business cards for individual customers that are printed on one set of sheets and subsequently cut are an example of this kind of situation. In cases such as these, small jobs can be collected in order to form a big job that may then be executed as a whole. This allows job aspects such as production, equipment load, and balancing of implementation resources to be performed more efficiently.

Note that production devices will generally require their resources to unambiguously define the production details. Thus a JDF Agent must prepare the resources in a way that the exact positioning of the contents of individual small jobs is specified. It is therefore recommended to use the procedure that is described in this section for Product intent nodes only.

In this example, diagrammed in Figure 4.9, nodes C and E represent small jobs of identical type. Node bigF represents a big job, which may exist already or which may have been created for the purposes of this spawning-and-merging process. Once nodes C and E are gathered beneath node bigF, as described below, a big job may then be executed as a whole for the sake of efficiency. When the big job is executed, the small jobs are effectively executed simultaneously. Nodes A, B, and D are provided to demonstrate that spawned nodes in this example may be related to other nodes in various ways.

```
JDF-node: A
Status=" Waiting "
```

```
JDF-node: B
Status=" Waiting "
```

```
JDF-node: D
Status="Waiting"
```

```
JDF-node: C
Status="Spawned"
Type="XYZ"
```

```
JDF-node: E
Status="Spawned"
Type="XYZ"
```

```
JDF-node: bigF
Status="Waiting"
Type="XYZ"
AuditPool:
   Merged (C, Indep.="true")
   Merged(E, Indep.="true")
```

```
JDF-node: C
Status="Waiting"
Type="XYZ"
Activity="Inactive"
```

```
JDF-node: E
Status="Waiting "
Type="XYZ"
Activity="Inactive"
```

Spawning Diagram

independent job X   independent job Y   big job bigZ

Nodes A, B, C   Nodes D, E   Node bigF

small Job X

small Job Y

execution phase

Time   Time

Figure 4.9 Example of the spawning and merging of independent jobs

Spawning

Spawning begins as it did in Case 1 or Case 2. Then, the process to be spawned (job C in Figure 4.9) is copied into a newly created, or already existing, big job (big job bigZ in Figure 4.9). The process type of the root node of the big job must be identical to that of the spawned processes. The *Activation* state of the spawned processes is set to *Inactive*, and an AncestorPool element is added to the inactive spawned job to define the ancestry (as was described above). A Merged element containing information about the spawned independent jobs and when they have been received is added to the big job.

In the original jobs, the *Status* of the process is designated as *Spawned*, and a Spawned element with the optional attribute *jRefDestination* specified is added to the parent of the original job. The attribute *jRefDestination* contains the ID of the big job beneath which the spawned process has been placed. The changes in the parent are the equivalent of those described in Case 1, except for the specification of the attribute *jRefDestination* in the Spawned element.

Where necessary, resource instances must be copied and logged as in Case 2 by appending the IDs to the appropriate attribute (*rRefsROCopied* or *rRefsRWCopied*) of the Spawned element in the parent of the original job. This is required in single spawning and merging. Furthermore, the ResourceLink elements of the spawned process must be copied to the ResourceLinkPool of the active, big process node. In this way, the input resources and the resources to be produced are linked to the big job.

Merging

For each of the spawned small jobs, the return procedure is performed as it was in the preceding cases. Once the process explained in Case 1 is performed, the completed job is copied back to its original location and the attribute *Activation* is restored by setting it to the activation of the big-job node after completion.

Eventually, copied resources must be purged and handled just as they were in Case 2. Then, the merging must be logged by appending the Merged element to the AuditPool container of the parent of the original node. In independent spawning and merging, the attribute *jRefSource* must be specified in the appropriate Merged element.

If the big job is retained, a Spawned element with the attribute *Independent* = *true* must be appended to the AuditPool of the big job. For instance, saving the finished big job may be desirable if the audit information con-

tained in the big job should be available for an individual invoicing. Finally, the newly created big JDF should be deleted to avoid the double existence of nodes.

4.4.6 Case 6: Simultaneous Spawning and Merging of Multiple Nodes

It is not possible to explicitly spawn multiple nodes simultaneously. The nodes must be grouped into a single *ProcessGroup* node, and this node can then be spawned and merged as described in the previous sections.

4.5 Node and Resource IDs

All nodes and resources must contain a unique identifier, not only because it is important to be able to identify individual components of a job, but because JDF uses these IDs for internal linking purposes. Each agent that creates resources and subnodes or that performs spawning and merging is responsible for providing IDs that are unique in the scope of the file, taking into account all of the phases of a job's life cycle.

IDs come in two flavors: pure and composite. A **pure ID** is an ID that does not contain the character period "." A **composite ID** is made up of pure IDs delimited by periods. For example:

pureID :: = ID − {'.'}
compositeID :: = pureID ['.'pureID]+
ID :: = pureID | compositeID

IDs are used differently under different circumstances. Several different circumstances are described below.

In case of no spawning — If an agent inserts new elements requiring IDs into an original job, then the agent assigns pure IDs to the new elements and must guarantee their uniqueness.

In case of single spawning — If an agent inserts new elements into a spawned job, then the agent creates composite IDs by using the ID of the root node and appending a unique pure ID delimited by a period. For example:
- ID of spawned root node: *ID* = "Job_01234.Proc1"
- ID used for new element: *ID* = " Job_01234.Proc1.*newpureID*"

In case of independent spawning — The agent that merges the independent jobs beneath a big job inserts a unique, pure ID (delimited by a period) in front of all IDs of each small job it receives. That means that the agent must replace all IDs of each job it receives whenever it encounters an ID collision. If an agent inserts new elements into a spawned job, then the agent creates composite IDs by using the ID of the respective root node of the small job and appends unique pureID, delimited by a period. For example:
- ID of the big job with node *ID* = "A"
- Receives small job A_1 with some IDs: *ID* = "A" *ID* = "A.A" *ID* = "A.B" where the first is the ID of the root node.
- Receives small job A_2 with some IDs: *ID* = "A" *ID* = "A.A" *ID* = "anything" …
- The agent creates locally unique pure IDs: *ID* = "A1" and *ID* = "A2" each prepended to all IDs of each received small job; the IDs of the small job A_1 become: *ID* = "A1.A" *ID* = "A1.A.A" *ID* = "A1.A.B" and the IDs of the small job A_2 become: *ID* = "A2.A" *ID* = "A2.A.A" *ID* = "A2.anything". All IDs in the big job are unique.
- The agent creates a new element added to the small job A_1 with ID: *ID* = "A1.A.C". Here the agent must resolve the possible conflict if it would append the pure ID = "A" to the root ID = "A1.A". That means the agent has to check the uniqueness of each created ID.
- Before merging the jobs back to its original location the agent must remove the prepended pure IDs of all IDs, here "A1", "A2" respectively. Then the newly created element will be merged back with the *ID* = "A.C".

4.6 Error Handling

Error handling is an implementation-dependent feature of JDF-based systems. The AuditPool element provides a container where errors that occur during the execution of a JDF may be logged using Notification elements. Notification elements may also be sent in JMF Signal messages. The content of the Notification element is described in Table 3-30. Further details about error handling are provided in the next four sections.

4.6.1 Classification of Notifications

Notification elements are classified by the attribute *Class*. Every workflow implementation must associate a class with all events on an event-by-event basis. The following list shows the possible values for *Class*:

- *Event*   Indicates a **pure event** which occurred due to a certain operation-related action, for example, machine events, operator activities, etc. This class is used for messaging.

- *Information*   Indicates not an error, but rather any information about a process that cannot be expressed by the other classes, for example, the beginning of execution.

- *Warning*   Indicates that a minor error has occurred and an automatic fix was applied. Execution continues. The node's *Status* is unchanged. Appears in situations such as A4-Letter substitutions, when toner is low, or when unknown extensions are encountered in a required resource

- *Error*   Indicates that an error has occurred that requires user interaction. Execution cannot continue until the problem has been fixed. The node's *Status* is *Stopped*. This value appears in situations such as when resources are missing, when major incompatibilities are detected, or when the toner is empty.

- *Fatal*   Execution must be aborted. The node's *Status* is *Aborted*. This value is seen with most protocol errors or when major device malfunction has occurred.

4.6.2 Event Description

A description of the event is given by a generic Comment element, which is required for the notification classes *Information*, *Warning*, *Error*, or *Fatal*. For example, after a process is aborted, error information describing a device error may be logged in the Comment element of the Notification element. If phase times are logged, the PhaseTime element that logged the transition to the *Aborted* state may also contain a local Comment element that describes the cause of the process abortion. PhaseTime and Notification elements are optional subelements of the AuditPool, which is described in Section 3.10.

4.6.3 Error Logging in the JDF File

A JDF-compliant controller/agent should log an error by inserting a Notification element in the AuditPool of the node that generated the error. The NodeInfo element may contain NotificationFilter elements to define the notification events (or, more specifically, errors) that should be logged.

4.6.4 Error Handling via Messaging (JMF)

A JMF Signal message with a Notification element in the message body should be sent through all persistent channels that subscribed events of class *error*. How to subscribe error events via JMF, see Sections 5.2.2.3 **Persistent Channels** and 5.5.1.1 **Events**. Note that this is different from the NotificationFilter elements of the NodeInfo element, which is defined for logging events by Notification elements to the AuditPool.

4.7 Test Running

In JDF, the notion of a test run is similar to the press notion of preflight. The goal is to detect JDF content errors and inconsistencies in a job before the job is executed.

The ability to perform a test run may be built into individual devices or controllers. Alternatively, a controller implementation may perform test runs on behalf of its devices. A test run may be routed through all of the different devices and controllers in a workflow, just as if the test run were a standard execution run. For the routing of jobs and nodes through different devices and controllers for a test, the spawning and merging mechanism may also be applied. The devices/controllers receiving a job read it and analyze WITHOUT initiating execution. Rather, they investigate the content of the node they would execute. A device/controller with agent capabilities may record results into the audit pool associated with a given process.

During test running, the requirements of the processes specified are compared to the capabilities of the devices targeted. A device or controller explicitly tests whether the inputs that have been specified as required are actually the inputs that are required, and that none are missing or in error. For example, an input requirement may be a URL that, when a test run is performed, is found to point to an item that no longer exists in that location. Test running is

meant to prevent errors as a result of that kind of misinformation. It is particularly useful when running expensive or time-consuming jobs.

It is also possible to test run specific parts of a workflow, or even individual nodes. An agent may request a test of certain nodes by setting the JDF attribute *Activation* to *TestRun* (see Table 3-3), which is inherited by all descendent nodes that are not inactive (*Activation* = *Inactive*). If a device or controller[2] detects an error in a node a Notification element containing a textual description should be appended to the AuditPool element of the node in which the error occurred, and, if messaging is supported, the error should be also communicated to the connected listeners via messaging (for more information see Section 5.4 **Error and Event Messages**). If an error has been detected, the agent can modify the job in order to correct the error. Once a test run has been completed successfully, the device/controller with agent capabilities changes the *Status* attribute of the tested node to *Ready*. If a test run fails, the device/controller is required to record the process status as *FailedTestRun*. After the test run has finished, the agent should log the result by appending a ProcessRun element to the AuditPool element. For more information about audits, see Section 3.10 AuditPool.

In principle, execution and test runs may be run simultaneously. For example, one job part may be executed while another part requests only a test. JDF also defines an *Activation* value of *TestRunAndGo* that requests a test run and, upon successful completion, automatically initiates processing.

4.7.1 Resource Status During Testrun

In order to test run a complete set of nodes, it is sometimes necessary to imply the *Status* of resources that are produced by prior nodes. Successful test running does *not* set the *Status* attribute of a resource to *Available* unless the resource actually is available. Nodes that require an output resource of a node that has completed test running for purposes of test running may assume that these resources have a *Status* of *Available* for the purpose of test running as long as the producing node has a *Status* of *Ready*.
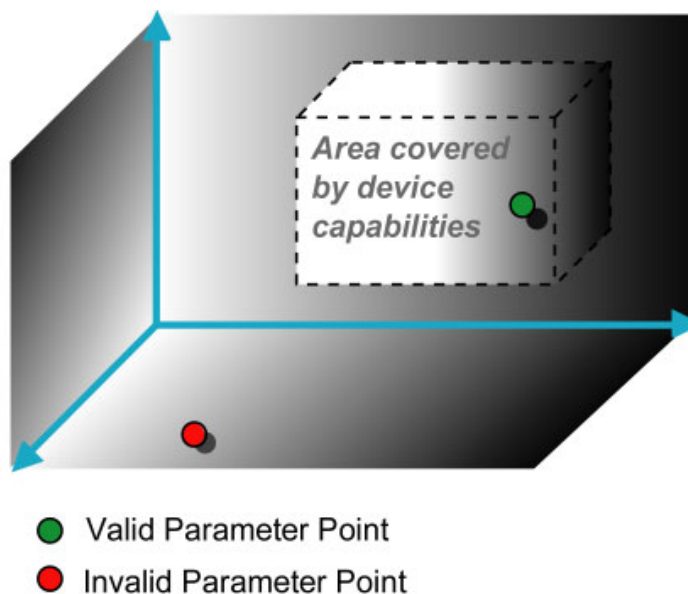
Figure 4.10 Parameter Space in device Capabilities[3]

4.8 Describing Device Capabilities with JDF

Device capabilities are described as a space of allowed resource parameter values within JDF. A device in this context is assumed to execute one or more JDF nodes. Its capabilities are defined by the space of acceptable JDF re-

---

[2] Note that only devices and controllers with agent capabilities can write in a JDF document.

[3] Note that the restriction to three dimensions is for graphical demonstration purposes only.

sources for the product intent or process described by the node. An individual JDF job description can be compared to the device capabilities of a device by looping over all resource parameters of a JDF node that is to be executed by a device. The job can be executed if all job parameter values are within the ranges specified by the device capabilities. Note that only orthogonal parameter relationships can be described. Functional dependencies between individual resources and resource attributes are NOT supported. If the device capabilities describe product intent, the job is executable when all product intent ranges overlap with the device capabilities description.

Details of the elements needed for device capability description are specified in Section 7.3 Device Capability Definitions.

It is assumed that **Device** elements that describe device capabilities will be transported in JMF **KnownDevices** messages. It is not recommended to specify the capabilities of a **Device** that is linked to a process to specify that it should execute the given process.

Chapter 5   JDF Messaging with the Job Messaging Format

Introduction

A workflow system is a dynamic set of interacting processes, devices and MIS systems. For the workflow to run efficiently, these processes and devices must communicate and interact in a well defined manner. Messaging is a simple but powerful way to establish this kind of dynamic interaction. The JDF-based Job Messaging Format (JMF) provides a wide range of capabilities to facilitate interaction between the various aspects of a workflow, from simple unidirectional notification through the issuing of direct commands. This chapter outlines the way in which JMF, accomplishes these interactions. The following list of use cases is considered:

J M F = R O I

In order to automate aspects of your production with out JDF, your technical staff must become proficient in each of the command languages that each of your devices employ. By only buying JDF-enabled devices that use JMF as their control language, you only have to learn one new device command language … eventually, the *only one* your MIS staff will need.

- System setup

- Dynamic status and error tracking for jobs and devices

- Pipe control

- Device setup and job changes

- Queue handling and job submission

- Device Capability description

Both Controllers and Devices may support JMF.   This support requires hosting by a Web server. JMF messages are most often encoded in pure XML, without an additional MIME/Multipart wrapper. Only controllers that support JDF job submission via the message channel must support MIME for messages.

5.1 JMF Root

JMF and JDF have an inherently different structure. In order to allow immediate identification of messages, JMF uses the unique name JMF as its own root-element name.

The root element of the XML fragment that encodes a message, like the root element of a JDF fragment, contains a series of predictable attributes and instances of Message elements. These contents are defined in the tables that follow, and are illustrated in Figure 5.1. Message elements are abstract, as is indicated by the dashed line surrounding the Message element in Figure 5.1.

Table 5-1 Contents of the JMF root

Name	Data Type	Description
DeviceID ?	string	Identifies the recipient device or controller. The envelope of the message contains the URL address of the controller that receives the message via HTTP. Therefore, if *DeviceID* does not specify a recipient, that controller is assumed to be the recipient.
SenderID	string	String that identifies the sender device, controller or agent.
TimeStamp	dateTime	Time stamp that identifies when the message was created.
Version ? Modified in JDF 1.1	string	JMF version. The current and default version is "1.1".
xmlns ? New in JDF 1.1	URI	JDF supports use of XML namespaces. The namespace must be declared. For details on using namespaces in XML, see http://www.w3.org/TR/REC-xml-names/.
Message +	element	Abstract message element(s).

The following table describes the contents of the abstract **Message** element. All messages contain an *ID* and a *Type* attribute.

Table 5-2 Contents of the abstract Message element

Name	Data Type	Description
ID	ID	Identifies the message.
Time ?	dateTime	Time at which the message was generated. This attribute is only required if this time is different from the time specified in the *Time-Stamp* attribute of the JMF element.
Type	NMTOKEN	Name that identifies the message type. Message types are described in Sections 5.5 and 5.6.

The following figure depicts the basic messaging structure and the message families.
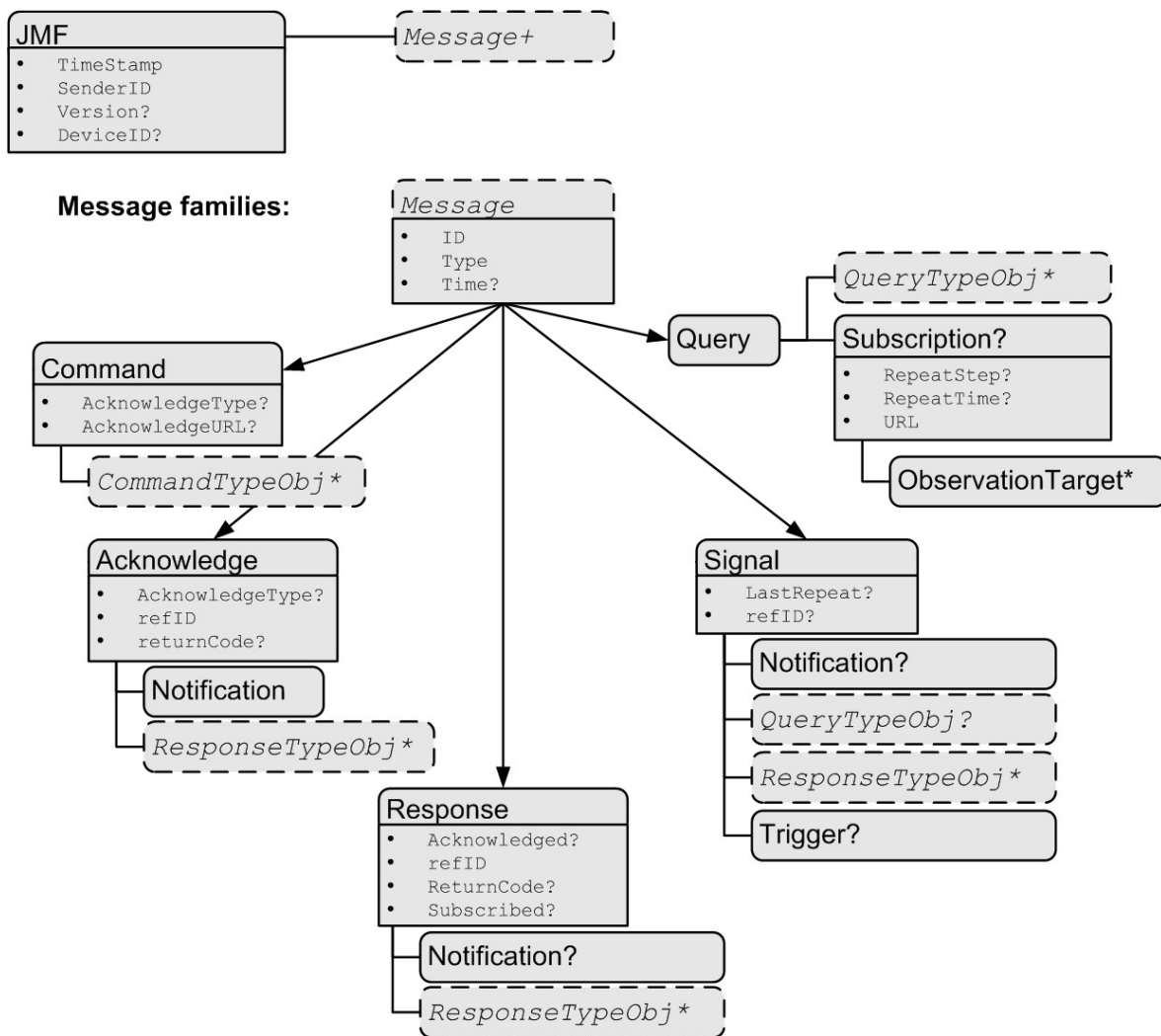
Figure 5.1 Contents of a JMF root element and the message families

5.2 JMF Semantics

JMF encodes messages of several types. The first part of this section describes message elements that contain and convey content, while the second describes the way in which these element types can be used to establish communication.

5.2.1 Message Families

A message contains one or more of the following five high level elements, referred to as **message families,** in the root node. These families are Query, Command, Response, Acknowledge, and Signal. An explanation of each family is provided in the following sections, along with an encoding example.

Response & Acknowledgement

The terminology used for message families contradicts common usage but will be retained for backwards compatibility. The Response actually functions as an *Acknowledgement* that a Command will be acted upon, while the Acknowledge could more properly be named *Completion* or *Result*. The naming was defined to be consistent with HTTP naming conventions so that a Response is always transported on an HTTP response.

5.2.1.1 Query

A Query is a message that retrieves information from a controller without changing the state of that controller. A query is sent to a controller. After a Query is sent, a Response is returned. If the Query included a Subscription, Signals are sent to the designated URL until a StopPersistentChannel Command is sent.
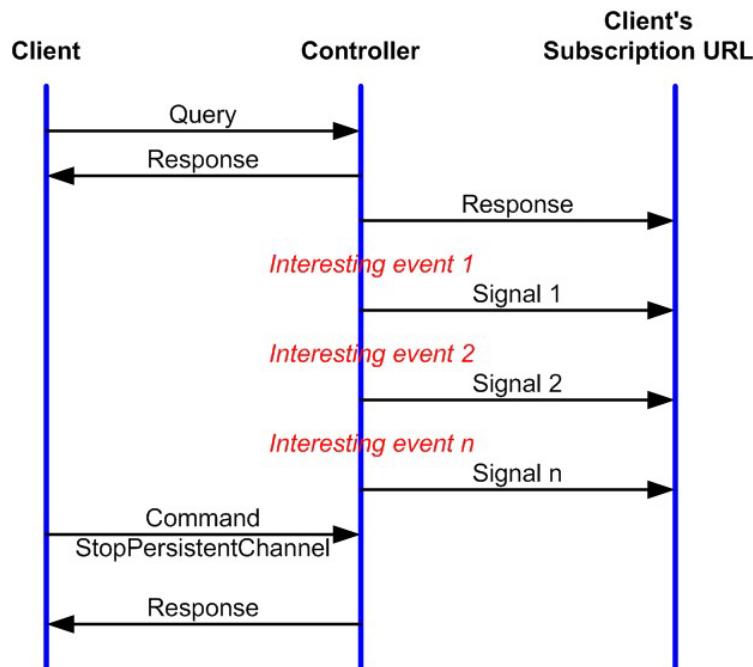
Figure 5.2 Interaction of Messages with a subscription

It contains an *ID* attribute and a *Type* attribute, which it inherits from the abstract message type described in Table 5-2 Contents of the abstract Message element. JMF supports a number of well defined query types, and each query type can contain additional descriptive elements, which are described in Sections 5.5 and 5.6. The following table shows the content of a Query message element.

Table 5-3 Contents of the Query message element

Name	Data Type	Description
QueryTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details required for the query. The element type of QueryTypeObj is defined by the *Type* attribute of the abstract **Message** element.
Subscription ?	element	If specified creates a persistent channel. For the structure of a *Subscription* element, see Section 5.2.2.3 Persistent Channels.

The following is an example of a query message:

```
<JMF TimeStamp="2000-07-25T11:38:23.3+02:00" SenderID="Controller-1">
xmlns="http://www.CIP4.org/JDFSchema_1_1"
  <Query Type="KnownJDFServices" ID="M007"/>
</JMF>
```

5.2.1.2 Response

A **Response** to a **Query** or a **Command** is always a direct answer of a **Query** or a **Command**. A response is returned from a controller to the controller that put the query/command. Responses are not acknowledged themselves.

A command response indicates that the command has been received and interpreted. The response of commands with short latency also includes the information about the execution. Commands with long latency may additionally generate a separate **Acknowledge** message (see Section 5.2.1.5 Acknowledge) to broadcast the execution of the command. Command responses should comprise a **Notification** element that describes the return status in text. Responses contain an attribute called *refID*, which identifies the initiating query or command. The following table shows the content of a **Response** message.

Table 5-4 Contents of the Response message element

Name	Data Type	Description
Acknowledged ?	boolean	Used only in responses to command messages. Indicates whether the command will be acknowledged separately. If *true*, an **Acknowledge** message will be supplied after command execution. If *false*, no **Acknowledge** message will be supplied. Default = *false*
refID	NMTOKEN	Copy of the *ID* attribute of the initiating query or command message to which the response refers.
ReturnCode ?	integer	Describes the result. 0 indicates success. For all other possible codes see Appendix I. Default = 0
Subscribed ?	boolean	If a **Subscription** element has been supplied by the corresponding query, this attribute indicates whether the subscription has been refused or accepted. If *true*, the requested subscription is accepted. If *false*, the subscription is refused because the controller does not support persistent channels. For details, see Section 5.2.2.3 Persistent Channels. Default = *true*
Notification ?	element	Additional information including textual description of the return code. The **Notification** element should be provided if the *ReturnCode* is greater than 0, which indicates that an error has occurred, or if the initiating message is a command.
ResponseTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details queried for or details about command execution.

An example of a response on a command is provided in the Section 5.2.1.4 Command. The encoding example for the query, shown above, might generate the following response:

```
<JMF TimeStamp="2000-07-25T11:38:25+02:00" SenderID="RIP-1">
  <Response Type="KnownJDFServices" ID="M107" refID="M007">
```

```
      <JDFService Type="Rendering"/>
      <JDFService Type="Imposition"/>
      <JDFService Type="Trapping"/>
   </Response>
</JMF>
```

5.2.1.3 Signal

A signal message, which is syntactically equivalent to a combination of a Query message and a Response message, is a unidirectional message sent on any event to other controllers. This kind of message is used to automatically broadcast some status changes.

Controllers can get signal messages in one of three ways. The first way is to subscribe for them with an initiating query transmitted via a message channel that includes a Subscription element. The second way is to subscribe for them with an initiating query defined in the NodeInfo element of a JDF node that also includes a Subscription element (see JMF elements in Table 3-7). The first query is transmitted separately via a mechanism such as HTTP, whereas the second is read together with the corresponding JDF node. Once the subscription has been established, signals are sent to the subscribing controllers via persistent channels. In both cases, however, the Signal message contains a *refID* attribute that refers to the persistent channel. The value of the *refID* attribute identifies the persistent channel that initiated the Signal.

The third way in which a controller may receive a signal is to have the signal channels hard-wired, for example, by a tool such as a list of controller-URLs read from an initialization file. For example, signals may be generated independently when a service is started, or when subcontrollers that are newly connected to a network want to inform other controllers about their capabilities. Hard-wired signals, however, must not have a *refID* attribute. If no *refID* is specified, the corresponding query parameters must be specified instead.

Table 5-5 Contents of the Signal message element

Name	Data Type	Description
LastRepeat ?	boolean	If *true*, the persistent channel is being closed by the controller and no further messages will be generated that fulfill the persistent channel criteria. If *false*, further signals will be sent. For further details, see Section 5.2.2.3 Persistent Channels. Default = *false*
refID ?	NMTOKEN	Identifies the initiating query message that subscribed this signal message. Hard-wired signals must not contain a *refID* attribute.
Notification ?	element	Textual description of the signal. The Notification element should be provided if the severity of the event that caused this signal is greater than *warning*, or if pure events have been subscribed. For details about subscribing pure events see Section 5.5.1.1 Events.
QueryTypeObj ?	element	If no *refID* is specified, the corresponding query parameters must be specified instead by providing this element.
		This element is an abstract element and a placeholder for any descriptive elements that provide details for the virtual Query, which, if sent, would convey the same ResponseTypeObj elements. The element type of QueryTypeObj is defined by the *Type* attribute of the abstract Message element.
ResponseTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details subscribed. These element types are the same as in the Response message element.
Trigger ?	element	Describes the trigger event which caused this signal. The Trigger element recalls some information provided during the subscription of the signal messages. For details on subscribing signals see Section 5.2.2.3 Persistent Channels.

The following table describes the structure of the Trigger element.

Table 5-6 Contents of the Trigger element

Name	Data Type	Description
RepeatStep ?	integer	Recalls the *RepeatStep* attribute specified during subscription of the signal. For details see Table 5-12.
RepeatTime ?	number	Recalls the *RepeatTime* attribute specified during subscription of the signal. For details see Table 5-12.
ChangedAttribute *	element	If a change of an attribute triggered this signal, this element describes the attribute that changed.
Added ?	element	A pool that contains the description of trigger events caused by the adding of elements like services, controllers, devices, or messages.
Removed ?	element	A pool that contains the description of trigger events caused by the removal of elements like services, controllers, devices, or messages.

The following describes the structure of the ChangedAttribute element referenced in the table above.

Table 5-7 Contents of the ChangedAttribute element

Name	Data Type	Description
AttributeName	NMTOKEN	Name of the attribute that changed.
ElementID ?	NMTOKEN	ID of the element that changed. Used only in conjunction with a change of a certain resource or node which cannot uniquely be addressed by the other attributes of this element. Default = none.
ElementType	NMTOKEN	Name of the element which contains the changed attribute.
OldValue	string	Old value. The string has to be cast to the appropriate data type that depends on the attribute's data type.
NewValue	string	New value of the attribute.

The following describes the structure of the Added element referenced in Table 5-6.

Table 5-8 Contents of the Added element

Name	Data Type	Description
AddedElement *	element	If the appending of an element like a service, controller, device, or message triggered this signal, this element describes which service, controller, device, or message etc. has been added.
		This is an abstract element. It is a placeholder for a ResponseTypeObj like NotificationDef, a JDFController, a Device, a JDFService, or a MessageService.
		For details on these elements see Section 5.5.1 Controller Registration and Communication Messages.

The following describes the structure of the Removed element referenced in Table 5-6.

Table 5-9 Contents of the Removed element

Name	Data Type	Description
RemovedElement *	element	If the removal of an element like a service, controller, device, or message triggered this signal, this element describes which service, controller, device, or message etc. has been removed.
		This is an abstract element. It is a placeholder for a ResponseTypeObj like NotificationDef, a JDFController, a Device, a JDFService, or a MessageService.
		For details on these elements see Section 5.5.1 Controller Registration and Communication Messages.

The following is an example of a signal message:

```
<JMF TimeStamp="2000-07-25T12:28:01+02:00" SenderID="Press 45">
  <Signal Type="Status" ID="s123">
    <StatusQuParams JobID="42" JobPartID="66"/>
    <DeviceInfo DeviceStatus="Setup"/>
  </Signal>
</JMF>
```

5.2.1.4 Command

A command is syntactically equivalent to a Query, but rather than simply retrieving information, it also causes a state change in the target device. The following table contains the contents of a Command message. A Response is returned immediately after a Command. If the Command included an *AcknowledgeURL*, and the Command was going to take a while, the device controller may elect to return the Response with *Acknowledge* = *true*, and send an Acknowledge to the *AcknowledgeURL* when the Command completes.

Table 5-10 Contents of the Command message element

Name	Data Type	Description
AcknowledgeURL ?	URL	URL of the recipient of any Acknowledge. If specified, the command requests for a Acknowledge message depending on the value of *AcknowledgeType.*
AcknowledgeType ? New in JDF 1.1	enumerations	Defines the actions that should be acknowledged. This is necessary mainly for device-machine pairs where the machine is not accessible online. *Received*: The Command has been received and understood, e.g. by an operator. *Applied*: The Command has been applied to the machine, e.g. by an operator. *Completed*: The Command has been executed. The default.
CommandTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details of the command.

The following example demonstrates how a ResumeQueueEntry command may cause a job in a queue to begin executing:

```
<JMF DeviceID="A3 Printer" TimeStamp="2000-07-25T12:32:48+02:00" SenderID="MIS master A">
  <Command ID="M009" Type="ResumeQueueEntry">
    <QueueEntryDef QueueEntryID="job-0032"/>
  </Command>
</JMF>
```

The following example shows a possible response to the command example above:

```
<JMF … SenderID="A3 Printer">
  <Response ID="M109" Type="ResumeQueueEntry" refID="M009">
    <Queue DeviceID="A3 Printer">
      <QueueEntry QueueEntryID="job-0032" Status="Running" JobID="job-0032"/>
    </Queue>
  </Response>
</JMF>
```

5.2.1.5 Acknowledge

An Acknowledge message is an asynchronous answer to a Command issued by a controller. Each Acknowledge message is unidirectional and syntactically equivalent to a command Response, and the *refID* attribute of each refers to the initiating command. Acknowledge messages are generated if commands with long latency have been executed in order to inform the command sender about the results. Acknowledge messages are only generated if the initiating command has specified the attribute *AcknowledgeURL*.

Command with Acknowledge
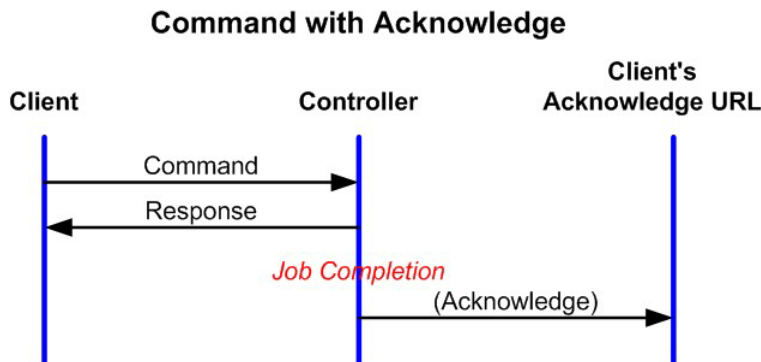
Figure 5.3 Interaction of Command and Acknowledge Messages

They are announced in the Response message to the command by the setting the attribute *Acknowledged* = *true*.

Table 5-11 Contents of the Acknowledge message element

Name	Data Type	Description
AcknowledgeType ? New in JDF 1.1	enumerations	Defines the context of this message. This is necessary mainly for device-machine pairs where the machine is not accessible online. *Received* – The initiating Command has been received and understood, e.g. by an operator. *Applied* – The initiating Command has been applied to the machine, e.g. by an operator. *Completed* – The initiating Command has been executed. The default.
Notification ? Modified in JDF 1.1A	element	Textual description of the command execution.
refID	NMTOKEN	Identifies the initiating command message the acknowledge refers to.
ReturnCode ?	integer	Describes the result. 0 indicates success. For all other possible codes see Appendix I. Default = 0
ResponseTypeObj *	element	Abstract element that is a placeholder for any descriptive elements that provide details about command execution. Delayed Acknowledge messages contain the same ResponseTypeObj elements as direct Response messages.

The following is an example of an Acknowledge message:

```
<JMF … >
  <Acknowledge ID="M109" Type="PipePush" refID="M010">
    <JobPhase … />
  </Acknowledge>
</JMF>
```

5.2.2 JMF Handshaking

JMF can seek to establish communication between system components in several ways. This section describes the actions and appropriate reactions in a communication using JMF.

5.2.2.1 Single Query/Command Response Communication

The handshaking mechanisms for queries and commands are equivalent. The initiating controller sends a Query or Command message to the target controller. The target parses the Query or Command and immediately issues an

appropriate Response message. If a Command with long latency is issued, an additional Acknowledge message may be sent to acknowledge when the command has been executed.

5.2.2.2 Signal

JMF signal messages are "fire and forget." In other words, no acknowledgment is sent by the receiver besides the standard protocol HTTP response that is sent when a communication link is sought.

5.2.2.3 Persistent Channels

Queries may be made persistent by including a Subscription element that defines the persistent channel-receiving end (see also Figure 5.1). The responding controller should initially send a Response to the subscribing controller. Then the responding controller should send Signal messages whenever the condition specified by one of the attributes in the following table is true. This is referred to as a **persistent channel**. The *refID* attribute of the Signal is defined by the *ID* attribute of the Query. In other words, the *refID* of the signal identifies the persistent channel. Any Query may be set up as a persistent channel, although in some cases this may not make sense.

Table 5-12 Contents of the Subscription element

Name	Data Type	Description
RepeatStep ?	integer	Requests an update signal whenever the *Amount* associated with the query is an integer multiple of *RepeatStep.*
		Default = 0, which means no repeat. Then it is up to the sending controller to generate Signals.
RepeatTime ?	number	Requests an update signal every *RepeatTime* seconds. If defined, the Signal is generated periodically independent of any other trigger conditions. Default = no repeat
URL	URL	URL of the persistent channel receiving end.
ObservationTarget *	element	Requests an updating Signal message whenever the value of one of the attributes specified in ObservationTarget changes.

Table 5-13 Contents of the ObservationTarget element

Name	Data Type	Description
ElementType ?	NMTOKEN	Name of the element that contains attributes that may change. Defaults to the abstract ResponseTypeObj of the message.
Attributes ?	NMTOKENS	Requests an update signal whenever the value of one of the attributes specified by *Attributes* is modified. A value of "*" denotes a message request for any attribute change which is the default.
ElementIDs ?	NMTOKENS	IDs of the elements that contain attributes that may change. Used only in conjunction with a query of the state change of a certain resource or node which cannot uniquely be addressed by the other attributes of this element. Default = none.

If a persistent signal channel has been set up and the device knows that this is the last time that the condition for signaling will be *true*, it should set the *LastRepeat* flag of the corresponding Signal message to *true*. In general, this will happen for a Status query, as when the job that has been tracked is completed. It may also happen when a device is shut down and will, therefore, not send any further updates. If a controller that does not support persistent channels is queried to set up a persistent channel, it must answer the query with a Response, set Subscribed to "false", and set the ReturnCode to "111".

   Multiple attributes of a Subscription element are combined as a boolean OR operation of these attributes. For instance, if *RepeatStep* and *ObservationTarget* are both specified, messages fulfilling either of the requirements are requested. If the subscription element contains only a URL, it is up to the emitting controller to define when to emit messages.

Creating Persistent Channels in a JDF Node

The NodeInfo element of a JDF node may contain JMF elements that contains a set of queries (not commands) that define persistent channels. Parsing a JDF that contains a JMF with a Subscription element is equivalent to receiving the messages that are specified in the JMF node. If the parsing controller cannot handle the request, it may generate a Response with *ReturnCode* = "111" and *Subscribed* = "*false*", accompanied by a Notification element

describing the rejection. It is not required to emit the Response, e.g., if the agent parses a Resource request but has no access to the device information.

Deleting Persistent Channels

A persistent channel may be deleted by sending a StopPersistentChannel command, as described in Section 5.5.1.7 StopPersistentChannel.

5.3 JMF Messaging Levels

A JDF-conforming controller may opt to support one of the following messaging compliance levels offered by JMF:

> **What's your JMF SOP?**
>
> As part of your strategic equipment purchasing procedures and requirements, consider what the JDF Messaging Levels are desired, and what the minimum level of conformance will be for your new equipment purchases.

- **No messaging (Level 0)** Controllers have the option of supporting no messaging at all. For this level, JDF includes Audit records for each process that allow the results of the process to be recorded.

- **Notification   (Level 1)** Most controllers will choose to support some level of messaging capability. Notification is the most basic level of support. Devices that support notification provide unidirectional messaging by sending Signal messages. Notification messages inform the controller when they begin and complete execution of some process within a job. They may also provide notice of some error conditions. Setup of the notification channel can be defined in a JDF node or hard-wired. In order to set up notification messages via a NodeInfo element, the controller must be able to read JMF query elements from a JDF document.

- **Query support (Level 2)** The next level of communication supports queries. Controllers that support queries respond to requests from other controllers by communicating their status using such tools as current *JobID* attributes, queued *JobID* attributes, or current job progress. Queries require bi-directional communication capabilities.

- **Command support (Level 3)** This level of support provides controllers with the ability to process commands. The controller can receive commands, for instance, to interrupt the current job, to restart a job, or to change the status of jobs in a queue.

- **Submission support (Level 4)** Finally, controllers may accept JDF jobs via an HTTP post request to the messaging channel. In this case, the messaging channel must support MIME/Multipart/Related documents. For more details on submission, see Section 5.6.3.8 SubmissionMethods.

Each messaging level encompasses all of the lower messaging levels.

5.4 Error and Event Messages

If a command or a query message is not successfully handled, a processor must reply with a standardized response that may contain a Notification element. Notification elements, described in detail in Section 3.10.1.2 Notification, convey a textual description. The information contained in the Notification element may be used by a user interface to visualize errors.

The response messages Response and Acknowledge contain a *ReturnCode* attribute. *ReturnCode* defaults to 0, which indicates that the response is successful. In case of success and in responses to commands an informational Notification element (*Class* = "*Information*") may be provided. In case of a warning, error or fatal error, the *ReturnCode* is greater than 0 and indicates the kind of error committed. In this case, a Notification element should be provided. Error codes are defined in Appendix I. The following example uses a Notification element to describe an error:

```
<JMF … >
  <Response ID="M109" Type="ResumeQueueEntry" refID="M009" ReturnCode="5">
    <Notification Class="Error" Type="Error">
      <Comment>StartJob unsuccessful – Device does not handle commands</Comment>
      <Error ErrorID="1234"/>
```

```
      </Notification>
      …
    </Response>
</JMF>
```

5.4.1 Pure Event Messages

Notification elements are also used to signal usual events due to any activities of a device, operator, etc., e.g., scanning a bar code. Such pure events can be subscribed by the Events message described in Section 5.5.1.1 Events. These Signals always have a *Type*="*Notification*":

```
<JMF … >
  <Signal ID="S1" Type="Notification" ReturnCode="0">
    <Notification Class="Event" Type="Barcode">
      <Comment>Palette completed</Comment>
      <Barcode Code="99923AAA123"/>
    </Notification>
  </Signal>
</JMF>
```

5.5 Standard Messages

The previous sections in this chapter provide a description of the overall structure of JMF messages. This section contains a list of the standard messages that are defined within the JDF framework. It is not required that every JDF-compliant application support every one of the signals and queries described in this list. It is, however, possible to discover which messages are supported in a workflow. A controller responds to the KnownMessages query by publishing a list of all the messages it supports (see Section 5.5.1.3 KnownDevices, below).

At the beginning of each section there is a table that lists all of the message types in that category. These tables contain three columns. The first is entitled "Message Type," and it lists the names of each message type. The second column is entitled "Family." The values in this column describe the kind of message that is applicable in the circumstance being illustrated. The following abbreviations are used to describe the values:

Q: Query
C: Command
R: Response
S: Signal

More than one of these values may be valid simultaneously. If that is the case, then all applicable letters are included in the column. Additionally, there are a few special circumstances indicated by particular combinations of these letters. The letters "QR" or "CR" indicate that all Query and Command messages cause a Response message to be returned. If the message may occur as a Signal, either from a subscription or independently, the "Family" field in the table also contains the letter "S". Finally, the third column provides a description of each element.

At the beginning of each section describing the contents and function of the message types listed in the tables described above is a table containing the instantiation (i.e., the type) of all of the abstract subelements applicable to the message being described. Each table contains an entry that describes the details of the query or command as well as an additional entry that describes the details of the corresponding response. The tables resemble the following template:

Table 5-14 Messaging table template

Object Type	Element name	Description
Abstract subelement of the **query** or **command**:	Name and type of the subelement that defines specifics of the query or command, followed by a cardinality symbol.	Short description of the subelement(s), if applicable.
Abstract subelement of the **response** to a query or command:	Name and type of subelement that contains specific information about the response to the query or command followed by cardinality symbol.	Short description of the subelement(s), if applicable.

The name of the abstract subelement of a Query element is QueryTypeObj, the name of the abstract subelement of a Command element is CommandTypeObj, and the name of the abstract subelement of a Response as well as an Acknowledge element is ResponseTypeObj.

5.5.1 Controller Registration and Communication Messages

The message types of the following table are defined in order to exchange metadata about controller or device abilities and for general communication.

Table 5-15 Process registration and communication messages

Message type	Family	Description
Events	QRS	Used to subscribe pure events occurring randomly like scanning of a bar code, activation of function keys at a console, error messages, etc.
KnownControllers	QRS	Returns a list of JMF-capable controllers.
KnownDevices	QRS	Returns information about the devices that are controlled by a controller.
KnownJDFServices	QRS	Returns a list of services (JDF Node Types) that are defined in the JDF specification.
KnownMessages	QRS	Returns a list of all messages that are supported by the controller.
RepeatMessages	QR	Returns a set of previously sent messages that have been stored by the controller.
StopPersistentChannel	CR	Closes a persistent channel.

5.5.1.1 Events

Table 5-16 Contents of the Events message

Object Type	Element name	Description
QueryTypeObj	NotificationFilter ?	Refines the list of events queried.
ResponseTypeObj	NotificationDef *	List of Notification types that match NotificationFilter.

The Events message type is intended to be used to query for supported event messages and to subscribe for randomly occurring events of a device or controller. These events are described in Section 4.6.1 Classification of Notifications and can only be transmitted via Signal messages. If the query contains a Subscription element, a NotificationFilter element is combined by a logical AND operation with the Subscription element for selective subscriptions. An empty Events message (without a Subscription and NotificationFilter element) can be used to query for all events, which are supported by a device or controller.

The controller that subscribes for Events messages receives Signal messages that convey only Notification elements containing information about the event. The event type and values of these messages may then be provided by specifying a *Type* attribute and an abstract NotificationDetails element in the Notification element, as described in Section 3.10.1.2 Notification. Possible NotificationDetails elements are defined in Appendix J NotificationDetails. Example of a subscription of Events and the response:

```
<JMF … >
  <Query Type="Events" ID="M170">
    <Subscription URL="http://www.anycompany.com/MIS/JMF/JobTracker"/>
    <NotificationFilter Classes ="Event Warning Error Fatal"/>
  </Query>
</JMF>
<JMF … >
  <Response ID="M1001" refID="M170" Type="Events">
    <NotificationDef Classes="Warning Error Fatal" Type="Error"/>
    <NotificationDef Classes="Event" Type="FCNKey"/>
    <NotificationDef Classes="Event Error" Type="Barcode"/>
    <NotificationDef Classes="Event" Type="SystemTimeSet"/>
    <NotificationDef Classes="Event" Type="anycompany:PrivateEvent_1"/>
```

```
    <NotificationDef Classes="Event" Type="anycompany:PrivateEvent_2"/>
  <Response/>
</JMF>
```

Structure of the NotificationFilter Element

Table 5-17 Contents of the NotificationFilter element

Name	Data Type	Description
DeviceID ?	string	ID of the device whose messages are queried/subscribed. May be specified for device selection if the controller controls more than one device.
JobID ?	string	JobID of the job whose messages are queried/subscribed.
JobPartID ?	string	JobPartID of the job whose messages are queried/subscribed.
Types ?	NMTOKENS	Possible notification type names are defined in Appendix J NotificationDetails. Matching notification types are returned/subscribed. Defaults to all supported notification types.
Classes ?	enumerations	Defines the set of notification classes to be queried/subscribed for. Possible values are: *Event* *Information* *Warning* *Error* *Fatal* Default = all. If both **Classes** and **Types** are a list, the NotificationFilter defines an OR of all permutations.

Structure of the NotificationDef Element

Table 5-18 Contents of the NotificationDef element

Name	Data Type	Description
Classes	enumerations	Possible values are: *Event* *Information* *Warning* *Error* *Fatal* For details, see Section 4.6.1 Classification of Notifications.
Type	NMTOKEN	Notification type, that is the name of the element derived from the abstract **NotificationDetails** element. For a list of predefined names see Appendix J NotificationDetails.

5.5.1.2   KnownControllers

Table 5-19 Contents of the KnownControllers message

Object Type	Element name	Description
QueryTypeObj	-	-
ResponseTypeObj	JDFController *	Known controllers.

The KnownControllers query requests information about the controllers and devices that are known to the controller and may be directly accessed by JMF messaging. KnownControllers is designed to define a registration server. A processor that needs information about its system environment can query a registration server for a list of known controllers. This list can subsequently be iterated using the other process registration queries in this section. The URL of the master registration server must be defined using a method outside of JDF.

JDFController

Table 5-20 Contents of the JDFController element

Name	Data Type	Description
URL	URL	URL of the controller.

The following is an example of a response to a KnownControllers query:

```
<Response ID="M1" refID="Q1" Type="KnownControllers">
  <JDFController URL="http://www.anycompany.com/controller" DescriptiveName="Printer
Controller"/>
  …
</Response>
```

5.5.1.3   KnownDevices

Table 5-21 Contents of the KnownDevices message

Object Type	Element name	Description
QueryTypeObj	DeviceFilter ?	Refines the list of devices queried. Only devices that match the **DeviceFilter** are listed. The default is to return a list of all known devices.
ResponseTypeObj Modified in JDF 1.1A	DeviceList ?	The list of known devices.[1]

The KnownDevices query requests information about the devices that are controlled by a controller. If a high level controller controls lower level controllers, it should also list the devices that are controlled by these. The response is a list of **Device** resources (see Section **7.2.44 Device**) controlled by the controller that receives the query, as demonstrated in the following example:

```
<Response ID="M1" refID="Q1" Type="KnownDevices">
  <DeviceList>
    <DeviceInfo DeviceStatus="Unknown">
      <Device DeviceID="Joe SpeedMaster" DeviceType="Heidelberg SM102/6 rev. 47" />
    </DeviceInfo>
  </DeviceList>
  …
</Response>
```

Structure of the DeviceFilter Element
The DeviceFilter element refines the list of devices that should be returned. Only devices that match all parameters of one of the **Device** resources specified in the DeviceFilter element are included.

---

[1] This was Device* prior to version 1.1 a. It was changed due to inconsistencies of the inheritance model in the JDF schema.

Table 5-22 Contents of the DeviceFilter element

Name	Data Type	Description
DeviceDetails ? New in JDF 1.1	enumeration	Refines the level of provided information about the device. Possible values are: *None* – Default value. *Brief* – Provide all available device information except for Device elements. *Modules* – ModuleStatus elements should be provided without module specific status details and without module specific employee information. *Details* – Provide maximum available device information excluding device capability descriptions. Includes Device elements which represent details of the device. *Capability* – Provide Device elements with DeviceCap subelements which represent details of the capabilities of the device. *Full* – Provide maximum available device information including device capability descriptions. Includes Device elements which represent details of the device.
Device *	element	Only devices that match the attribute values specified in one of these **Device** resources are included. Devices match the criteria if the attribute values specified here in the **Device** resource match the equivalent attribute values of the known devices. Unspecified attributes always match. If **Device** is not specified, all known **Device**s are returned.

Structure of the DeviceList Element
The DeviceList element contains a list of information about devices that are returned.
New in JDF 1.1 a

Table 5-23 Contents of the DeviceList element

Name	Data Type	Description
DeviceInfo *	element	List of information about known devices as requested by the DeviceFilter element. For details of the DeviceInfo element, see Table 5-44 Contents of the DeviceInfo element in the message description 5.5.2.3 Status.

5.5.1.4 KnownJDFServices
Table 5-24 Contents of the KnownJDFServices message

Object Type	Element name	Description
QueryTypeObj	-	-
ResponseTypeObj	JDFService *	Processes that the controller or device can execute.

The KnownJDFServices query returns a list of services that are defined in the JDF specification, such as ***ConventionalPrinting***, ***RIPping***, or ***EndSheetGluing***. It allows a controller to publish the services that the devices it controls are capable of providing. The response is a list of JDFService elements, one for each supported process type.

JDFService
JDFService elements define the node types that can be processed by the controller. A JDF processor should be capable of processing *Combined* nodes of any of the individual JDFService elements that are specified. It is therefore not necessary to define every permutation of allowed combinations. It need not be able to process individual nodes with a type defined in the *Types* attribute of a *Combined* JDFService element.

Table 5-25 Contents of the JDFService element

Name	Data Type	Description
CombinedMethod ? New in JDF 1.1	enumeration	Specifies how the processes specified in *Types* may be specified. One of: *Combined* – The list of processes in *Types* must be specified as a *Combined* process. *ProcessGroup* – The list of processes in *Types* must be specified as a *ProcessGroup* of individual processes. *CombinedProcessGroup* – The list of processes in *Types* may be specified either as a *Combined* process or as a *ProcessGroup* of individual processes. *None* – No support for *Combined* or *ProcessGroup*. Only the individual process type defined in *Types* is supported. The default.
Type	NMTOKEN	JDF *Type* attribute of the supported process. Extension types may be specified by stating the namespace in the value.
TypeOrder ? New in JDF 1.1	enumeration	Ordering restriction for combined nodes. Fixed – The order of process types specified in the Types attribute is ordered and each type can be specified only once, e.g. ,Cutting, Folding; order does matter. The default. Unordered – The order of process types specified in the Types attribute is unordered and each type can be specified only once, e.g., DigitalPrinting, Screening, Trapping; order does not matter. Unrestricted – The order of process types specified in the Types attribute is unordered and each type can be specified multiply, e.g., Cutting, Folding, where the device can do both processes, in any order and multiple times.
Types ?	NMTOKENS	If *Type* = *Combined*, or *Type* = *ProcessGroup* this attribute represents the list of combined processes. If any of the Services are in a namespace other than JDF, the namespace prefix should be included in this list. For details, see Section 3.2.3

The following is an example of a response to a KnownJDFServices query:

```
<Response ID="M1" refID="Q1" Type="KnownJDFServices">
  <JDFService Type="Rendering" />
  <JDFService Type="Folding" />
  <JDFService Type="Combined" Types="Gathering Stitching"/>
  <JDFService Type="AnyCompaniesNamespace:MyFolding" />
  …
</Response>
```

5.5.1.5 KnownMessages

Table 5-26 Contents of the KnownMessages message

Object Type	Element name	Description
QueryTypeObj	KnownMsgQuParams ?	Refines the query for known messages. If not specified, list all supported message types.
ResponseTypeObj	MessageService *	Specifies the supported messages.

The KnownMessages query returns a list of all message types that are supported by the controller.

KnownMsgQuParams

The flags of the KnownMsgQuParams element filter out the types of messages that should be included in the response list. Multiple flags are allowed.

Table 5-27 Contents of the KnownMsgQuParams element

Name	Data Type	Description
Exact ? New in JDF 1.1	boolean	Requests an exact description of the known messages. If *true*, the response should also return the requested *DevCaps* of the messages. Default = *false*
ListCommands ?	boolean	Lists all supported command types. Default = *true*
ListQueries ?	boolean	Lists all supported query types. Default = *true*
ListSignals ?	boolean	Lists all supported signal types. Default = *true*
Persistent ?	boolean	If *true*, only lists messages that may use persistent channels. If *false*, ignores the ability to use persistent channels. Default = *false*

MessageService

The response is a list of MessageService elements, one for each supported message type. The flags of the MessageService response element are set in each MessageService entry. They define the supported usage of the message by the controller. Note that no *Response* attribute is included in the list, since the capability to process one of the other message families implies the capability to generate an appropriate *Response*. Multiple flags are allowed.

Table 5-28 Contents of the MessageService element

Name	Data Type	Description
Acknowledge ? New in JDF 1.1	boolean	If *true* the device supports asynchronous Acknowledge answers to this message. Default = *false*
Command ?	boolean	If *true* the message is supported as a command. Default = *false*
Persistent ?	boolean	If *true* the message is supported as a persistent channel. Default = *false*
Query ?	boolean	If *true* the message is supported as a query. Default = *false*
Signal ?	boolean	If *true* the message is supported as a signal. Default = *false*
Type	NMTOKEN	Type of the supported message. Extension types may be specified by stating the namespace in the value.
DevCaps * New in JDF 1.1	element	Specifies the restrictions of the parameter space of the supported messages. For details on using DevCaps, see 7.3.3 Structure of the DevCaps Subelement.

The following is an example of a response to a KnownMessages query:

```
<Response ID="M1" refID="Q1" Type="KnownMessages">
  <MessageService Type="KnownMessages" Query="true"/>
  <MessageService Type="Status" Query="true" Signal="true" Persistent="true">
  …
</Response>
```

5.5.1.6   RepeatMessages

Table 5-29 Contents of the RepeatMessages message

Object Type	Element name	Description
QueryTypeObj	MsgFilter ?	A filter for the messages to be repeated. For details, see Section 5.5.1.1 Events.
ResponseTypeObj	Message *	The recent messages queried.

The RepeatMessages query returns a list of messages that have been previously sent by the controller. The optional MsgFilter element allows the list to be filtered. The list of JMF messages that fulfill the filter criteria may be sorted by time, with the most recent listed first. This specification places no requirements on the size of the message buffer of a controller that supports RepeatMessages.

Structure of the MsgFilter Element

Table 5-30 Contents of the MsgFilter element

Name	Data Type	Description
After ?	dateTime	Messages sent only after a certain time.
Before ?	dateTime	Messages sent only before a certain time.
Count ?	integer	Maximum number of messages, most recent first.
DeviceID ?	string	ID of the device whose messages are required.
Family ?	enumeration	Message family. Possible values are: *Acknowledge* *Response* *Signal* *All* – Default value. Response, Signal, and Acknowledge messages are queried.
MessageRefID ?	NMTOKEN	The *refID* attribute must match the value of *MessageRefID.*
MessageID ?	NMTOKEN	The *ID* attribute must match the value of *MessageID.*
MessageType ?	NMTOKEN	*Type* attribute of the requested messages.
ReceiverURL ?	URL	URL for which the messages are intended.

If the returned list is incomplete because the parameters supplied in the MsgFilter element cannot be fulfilled by the application, the *ReturnCode* may be 108 (empty list) or 109 (incomplete list) and should be flagged as a warning.

The following is an example of a response to a RepeatMessages query. Note the nesting of Response messages, where the first layer is the response to the RepeatMessages query and its contents are the repeated messages.

```
<JMF TimeStamp="2000-06-14T12:11+02:00" … >
  <Response … >
    <Response Time="2000-06-14T11:00+02:00" … />
    <Response Time="2000-06-14T10:50+02:00" … />
    <Signal Time="2000-06-14T08:20+02:00" … />
    <Signal Time="2000-06-14T03:01+02:00" … />
    …
  </Response>
</JMF>
```

5.5.1.7   StopPersistentChannel

Table 5-31 Contents of the StopPersistentChannel message

Object Type	Element name	Description
CommandTypeObj	StopPersChParams	Specifies the persistent channel and the message types to be unsubscribed.
ResponseTypeObj	-	-

The StopPersistentChannel command unregisters a listening controller from a persistent channel. No more messages are sent to the controller once the command has been issued. A certain subset of signals may be addressed for unsubscription by specifying a StopPersChParams element.

Structure of the StopPersChParams Element

If the optional attributes are not specified, those attributes default to match anything. Therefore it may be possible to cancel the persistent channel for messages belonging to a certain type of message or to a certain job.

Table 5-32 Contents of the StopPersChParams element

Name	Data Type	Description
ChannelID ?	NMTOKEN	*ChannelID* of the persistent channel to be deleted. If the channel has been created with a Query message, the *ChannelID* specifies the *ID* of the Query message (identical to the *refID* of the Response message).
MessageType ?	NMTOKEN	Only messages with a matching message type are suppressed. Message types are specified in the *Type* attribute of each Message element. Defaults to all message types.
DeviceID ?	string	Only messages from devices or controllers with a matching *DeviceID* attribute are suppressed.
JobID ?	string	Only messages with a matching *JobID* attribute are suppressed.
JobPartID ?	string	Only messages with a matching *JobPartID* attribute are suppressed.
URL	URL	URL of the receiving controller. This must be identical to the URL that was used to create the persistent channel. If no *ChannelID* is specified, all persistent channels to this URL are deleted.

5.5.2   Device/Operator Status and Job Progress Messages

JDF Messaging provides methods to trace the status of individual devices and resources and additional job-dependent job-tracking data.. The status of a job is described by the Status elements of that job.

Devices are uniquely identified by a name—that is, by the attribute *DeviceID* of the **Device** resource (see Section 7.2.44 **Device**)—while controllers are uniquely identified by their URL. In other words, controllers are implicitly identified as a result of the fact that they are responding to a message. One controller may control multiple devices. The following queries and commands are defined for status and progress tracking:

Table 5-33 Status and progress messages

Message type	Family	Description
Occupation	QRS	Queries the occupation of an employee.
Resource	QRSC	Queries and/or modifies JDF resources that are used by a device, such as device settings, or by a job. This message can also be used to query the level of consumables in a device.
Status	QRS	Queries the general status of a device, controller or job.
Track	QRS	Queries the location of a given job or job part.

5.5.2.1   Occupation

Table 5-34 Contents of the Occupation message

Object Type	Element name	Description
QueryTypeObj	EmployeeDef *	Defines the employees queried.
ResponseTypeObj	Occupation *	The occupation status of the employees.

Occupation queries the occupation status of an employee. No job context is required to issue an Occupation message.

Structure of the EmployeeDef Element

The Occupation query may be focused to certain employees specifying a EmployeeDef element. If no Employ-eeDef element is specified, a list of all known employees is returned.

Table 5-35 Contents of the EmployeeDef element

Name	Data Type	Description
PersonalID ?	string	*PersonalID* of the employee being tracked.

Structure of the Occupation Element

The response returns a list of Occupation elements for the queried employees. These elements consist of one entry for every job that is currently being executed. The list format accommodates both employees that service multiple jobs or job parts in parallel and multiple employees working on one job.

Table 5-36 Contents of the Occupation element

Name	Data Type	Description
Busy ?	number	Busy state of the employee in percentage. A value of 100, the default, means that the employee is fully occupied with this task. The sum of all *Busy* values should not exceed 100.
Device *	element	Devices that the employee is currently assigned to.
JobID ?	string	*JobID* of the JDF node that the employee is assigned to. If no JobID is specified but devices are, the employee is performing tasks not related to a job.
JobPartID ?	string	Job part ID of the JDF node that is currently being executed.
Employee	element	Description of the employee being tracked.

The following is an example of response to an Occupation query:

```
<Response ID="M1" refID="Q1" Type="Occupation">
  <!—Two jobs on one device with one operator-->
  <Occupation JobID="J1" Busy="30">
    <Employee PersonalID="P1234"/>
    <Device Name="Joe"/>
  </Occupation>
  <Occupation JobID="J2" Busy="70">
    <Employee PersonalID="P1234"/>
    <Device Name="Joe"/>
  </Occupation>
  <!—Another operator on job j2 -->
  <Occupation JobID="J2" Busy="50">
    <Employee PersonalID="P4321"/>
    <Device Name="Joe"/>
  </Occupation>
  <!—No Job context -->
  <Occupation  Busy="0">
```

```
    <Device Name="John"/>
    <Employee PersonalID="P5678"/>
  </Occupation>
</Response>
```

5.5.2.2 Resource

The Resource message can be used as a command or a query to modify or to query JDF resources. In both cases (query and command), it is possible to address either global device resources, such as device settings, or job-specific resources. The query simply retrieves information about the resources without modifying them, while the command modifies those settings within the resource that are specified. Settings that are not specified remain unchanged.

Structure of the Resource Query Message

Table 5-37 Contents of the Resource query message

Object Type	Element Name	Description
QueryTypeObj	ResourceQuParams ?	Specifies the resources queried.
ResponseTypeObj	ResourceInfo *	Contains the amount data of resources and, if requested, the resources itself.

The Resource query may be made selective by specifying a ResourceQuParams element. The presence of the *JobID* attribute determines whether global device resources or job-related resources are returned. If no ResourceQuParams element is specified, only the global device resources are returned.

The query response returns a list of ResourceInfo elements that contains the queried information concerning the resources. If the list is empty because the selective query parameters of the ResourceQuParams lead to a null selection of the known device/job resources, then the *ReturnCode* may be 103 (JobID unknown), 104 (JobPartID unknown) or 108 (empty list) and should be flagged as a warning.

Structure of the ResourceQuParams Element

Table 5-38 Contents of the ResourceQuParams element

Name	Data Type	Description
Classes ?	enumerations	List of the resource classes to be queried. For example, in order to query the actual level of consumables in a device outside of any job context, specify *Classes* = *Consumable* in the query without a *JobID* attribute. For possible resource class names, see the *Class* attribute in Table 3-11. Default = any class.
Exact ?	boolean	Requests an exact description of the JDF resource. If *true*, the response should also return the requested JDF resource. Default = *false*
JobID ?	string	Job ID of the JDF node that is being queried. If no *JobID* is specified, global device settings are queried.
JobPartID ?	string	Job part ID of the JDF node that is being queried.
Location ?	string	Identifies the location of a resource, such as paper tray, ink container, or thread holder. The name is the same name used in the Partition-key *Location* of distributed resources (see also Section 3.9.2.2 Locations of Physical Resources). Default = all locations
ProcessUsage ?	string	Selects a resource in which the value of the *ProcessUsage* attribute of the resource link (see Table 3-17) matches the token specified here in this attribute.

Only necessary if a resource name is used more than once by one node. For example, the **Component** output resources of a *ConventionalPrinting* process can be distinguished by specifying *ProcessUsage* = *Good* and *ProcessUsage* = *Waste*, respectively.

The *ResourceName, Usage* and *ProcessUsage* attributes are combined by a logical AND conjunction to select the resource to be |

Name	Data Type	Description
		queried.
ResourceName ?	NMTOKEN	Name of the resource being queried. For possible resource names, see titles in Chapter 7 Resources.
Usage ?	enumeration	*Input* – The resource is an input.
		Output – The resource is an output.
		Selects a resource in which the value of the *Usage* attribute of the resource link (see Table 3-17) matches the token specified here in this attribute. Only necessary if a resource name is used both as input and output by one node.

Structure of the Resource Command Message

Table 5-39 Contents of the Resource command message

Object Type	Element name	Description
CommandTypeObj	ResourceCmdParams	Specifies the resources to be modified.
ResponseTypeObj	ResourceInfo *	Contains information about the resources and the resources after modification.

The Resource command may be used to modify either global device settings or a running job. It may be made selective by specifying the optional attributes in the ResourceCmdParams element. The presence of the *JobID* attribute determines whether global device resources or job-related resources are modified.

The response contains a list of ResourceInfo elements with all resources and private extensions of the device after the changes have been applied. The type of the resource that is given as a response depends on the type of the resource given in the command.

If the resource command was successful, the value of the *ReturnCode* attribute is 0. If it is not successful, the value of *ReturnCode* may be one of those that have been described above in the section about the Resource query message, 200 (invalid resource parameters), or 201 (insufficient resource parameters). Partial application of the resource should also be flagged as a warning. If the value of *ReturnCode* is larger than 0, the controller that issued the command can evaluate the returned resource in order to find the setting that could not be applied.

Structure of the ResourceCmdParams Element

Table 5-40 Contents of the ResourceCmdParams element

Name	Data Type	Description
Activation ? New in JDF 1.1	enumeration	Describes the activation status of the uploaded resource. Allows for a range of activity, including deactivation and testrunning. Possible values, in order of involvement from least to most active, are:
		Held – Used for uploading a resource that requires operator intervention before being applied.
		TestRun – Used for a test run check by the controller or a device. This does not imply that the update should be automatically applied when the check is completed.
		TestRunAndGo – Similar to *TestRun,* but requests a subsequent automatic update of the resource if the testrun has been completed successfully.
		Active – Default value. The update must be applied immediately.
		Note that the Inactive value defined in JDF::*Activation* is not a valid value in this list.

Name	Data Type	Description
Exact ?	boolean	Requests an exact description of the JDF resource. If *true*, the response should also return the requested JDF-resource. Default = *false*
JobID ?	string	Job ID of the JDF node that is being modified. If no *JobID* is specified, global device settings are modified.
JobPartID ?	string	Job part ID of the JDF node that is being modified.
ResourceName ?	NMTOKEN	Name of the resource whose production amount will be modified. For possible resource names see titles in Chapter 7 Resources. Default = any name
ProcessUsage ?	NMTOKEN	Selects a resource in which the value of the *ProcessUsage* attribute of the resource link (see Table 3-17) matches the token specified here in this attribute. Only necessary if a resource name is used more than once by one node. For example, the **Component** output resources of a **ConventionalPrinting** process can be distinguished by specifying *ProcessUsage* = *Good* and *ProcessUsage* = *Waste*, respectively. The *ResourceName* and *ProcessUsage* attributes are combined by a logical AND conjunction to select the resource to be queried.
ProductionAmount ?	number	New amount of resource production. This value replaces the *Amount* in the output resource link of the resource specified by the *ResourceName* attribute.
UpdateIDs ? New in JDF 1.1	NMTOKENS	The *UpdateID* attributes of one or more ResourceUpdate that are defined in resources known to the recipient. The data type is NMTOKENS and not IDREFS because no matching IDs exist within this message. The order of tokens in defines the order in which the updates are applied.
Resource *	element	Resources to be uploaded to the controller. They completely replace the original resources with the same ID. The resources to be modified are identified by their ID values, which means that the ID attributes must be known to the controller that issued the Resource command.

Structure of the ResourceInfo Element

Table 5-41 Contents of the ResourceInfo element

Name	Data Type	Description
Amount ?	number	Intended amount for consumption or production of a resource in a job context. This corresponds to the value of the *Amount* attribute in the corresponding resource link of the resource.
AvailableAmount ?	number	Device-specific amount of the *Consumable* resource that is available in the device.
Level ?	enumeration	This attribute is device dependent. A device may specify the level status that describes a low or empty consumable level. Possible values are: *Empty* – Specification is left to the device manufacturer. *Low* – Specification is left to the device manufacturer. *OK* – Default value.

Name	Data Type	Description
Location ?	string	Device-specific string to identify the location of a given consumable, such as paper tray, ink container, or thread holder. The name is the same name used in the Partition-key *Location* of distributed resources (see also Section 3.9.2.2 Locations of Physical Resources). Default = all locations
ResourceName ?	NMTOKEN	Name of the resource if *Exact* = *false* in the query. Only one of Resource or *ResourceName* must be specified.
ProcessUsage ?	NMTOKEN	Selects a resource in which the value of the *ProcessUsage* attribute of the resource link (see Table 3-17) matches the token specified here in this attribute. Only necessary if a resource name is used more than once by one node. For example, the **Component** output resources of a **Conventional-Printing** process can be distinguished by specifying *ProcessUsage* = *Good* and *ProcessUsage* = *Waste*, respectively. The *ResourceName* and *ProcessUsage* attributes are combined by a logical AND conjunction to select the resource to be queried.
Unit ?	string	Unit of the amount attributes. In a job-context it is strongly discouraged to specify a unit other than the unit defined in the respective JDF resource, although this may be necessary due to technical considerations, such as when ink is specified in weight (g) and ink measurement is specified in volume (liter).
CostCenter ?	element	Cost center to which the resource consumption is allocated.
Resource ?	element	JDF description of the resource.

The following is an example for retrieving settings:

```
<Query ID="Q1" Type="Resource">
  <ResourceQuParams Classes="Consumable" Exact="true"/>
</Query>
```

The following is a possible response to the query above:

```
<Response ID="M1" refID="Q1" Type="Resource">
  <ResourceInfo Location="Paper Tray 1" AvailableAmount="2120" >
    <Media>
    ...  <!-- Media resource defined in JDF -->
    </Media>
  </ResourceInfo>
  <ResourceInfo Location="Ink1" AvailableAmount="0" Unit="l" Level="Empty">
    <Ink>
    ...  <!-- Ink description resource defined in JDF -->
    </Ink>
  </ResourceInfo>
</Response>
```

The following is an example for modifying the production amount of a specific job to produce brochures:

```
<Command ID="C1" Type="Resource">
  <ResourceCmdParams JobID="MakeBrochure 012" ResourceName="Component" Pro-
ductionAmount="7500"/>
</Command>
```

The following is a possible response to the resource command above:

```
<Response ID="M2" refID="C1" Type="Resource">
  <ResourceInfo Amount="7500" ResourceName="Component"/>
```

```
</Response>
```

5.5.2.3 Status

Table 5-42 Contents of the Status message

Object Type	Element name	Description
QueryTypeObj	StatusQuParams	Refines the query to include various aspects of the device and job states.
ResponseTypeObj	DeviceInfo	Describes the actual device status.
	Queue ?	Provides information about the queue and all its entries. This element will only be provided if the device has queue capabilities. The Queue element is described in Section 5.6.4 Queue-Handling Elements.

The Status message queries the general status of a device or a controller and the status of jobs associated with this device or controller. No job context is required to issue a Status message. The response contains one DeviceInfo element, which contains the device specific information and which may contain other JobPhase elements that in turn contain the job specific information. The response also provides a Queue element when commanded to do so.

Structure of the StatusQuParams Element

The various aspects of the device, queue, and job states may be refined by the StatusQuParams element. This element contains three groups of parameters. The first group serves to refine the device-specific status information queried. The parameters *EmployeeInfo* and *ModuleDetails* belong to this group. The second group serves to refine the job specific status information. These are *JobDetails*, *JobID*, and *JobPartID*. And the third determines simply whether a queue element should be appended. This is specified by the attribute *QueueInfo*.

In order to focus on the status of a certain job, the job must be uniquely identified using the *JobID* attribute. It may be necessary to define a process or a part of a job as the query target under certain circumstances, such as when a job is processed in parallel. This is accomplished using the *JobPartID* attribute of the StatusQuParams element. A value of *JobDetails* = *Full* requests a complete JDF description of a snapshot of the specified job or job part.

If the specified job or job part is unknown, the value of the *ReturnCode* attribute is 103 or 104 (for error codes, see Appendix I).

Table 5-43 Contents of the StatusQuParams element

Name	Data Type	Description
DeviceDetails ?	enumeration	Refines the provided status information about the device. Possible values are: *None* – Default value. *Brief* – Provide all available device information except for Device elements. *Modules* – ModuleStatus elements should be provided without module specific status details and without module specific employee information. *Details* – Provide maximum available device information excluding device capability descriptions. Includes Device elements which represent details of the device. *Capability* – Provide Device elements with DeviceCap subelements which represent details of the capabilities of the device. *Full* – Provide maximum available device information including device capability descriptions.. Includes Device elements which represent details of the device.
EmployeeInfo ?	boolean	If *true*, Employee elements may be provided in the response. Those elements describe the employees which are associated to the device independent on any job. Default = *false*.

Name	Data Type	Description
JobDetails ?	enumeration	Refines the provided status information about the jobs associated with the device. Each higher entry includes the values specified in the lower entries. Possible values are: *None* – Default value. Specify only *JobID*, *JobPartID* and *Amount* and/or *PercentCompleted*. *MIS* – Provide business with the relevant information contained in the CostCenter element and the *DeadLine*, *DeviceStatus*, *Status*, *StatusDetails,* and the various *Counter* attributes. *Brief* – Provide all available status information except for JDF. *Full* – Provide maximum available status information. Includes an actual JDF which represents a snapshot of the current job state.
JobID ?	string	Job ID of the JDF node whose status is being queried. Defaults to list all known jobs.
JobPartID ?	string	JobPart ID of the JDF node whose status is being queried.
QueueInfo ?	boolean	If *true*, a Queue element may be provided. This is analogous to a QueueStatus query (see Section 5.6.3.6 QueueStatus). Default = *false*.

Structure of the DeviceInfo Element

The response returns a DeviceInfo element for the queried device.

Table 5-44 Contents of the DeviceInfo element

Name	Data Type	Description
CounterUnit ?	string	The unit of the *ProductionCounter,* the *TotalProductionCounter* and nominator unit of *Speed*. The default unit is the default unit defined by JDF for the output resource of the node executed by the device. For example, in case of a sheet printer, it is the number of sheets; in case of a web printer, it is the length of printed web in meters.
DeviceStatus	enumeration	The status of a device. Possible values are: *Unknown* – No device is known or the device cannot provide a *DeviceStatus*. *Idle* – No job is being processed and the device is accepting new jobs. *Down* – No job is being processed and the device currently cannot execute a job. The device may be broken, switched off, etc. *Setup* – The device is currently being set up. This state is allowed to occur also during the execution of a job. *Running* – The device is currently executing a job. *Cleanup* – The device is currently being cleaned. This state is allowed to occur also during the execution of a job. *Stopped* – The device has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as execution has not been aborted.
HourCounter ?	duration	The total integrated time (life time) of device operation in hours. Default = unknown.
PowerOnTime ?	dateTime	Date and time when the device was switched on.

Name	Data Type	Description
		Defaults = unknown.
ProductionCounter ?	number	The current machine production counter. This counter can be re-set. Typically, it starts counting at power-on time. The reset of this counter may be signaled by an **Events** message of *Type* = *CounterReset* (see Appendix J NotificationDetails). Default = un-known.
Speed ?	number	The current machine speed. *Speed* is defined in the same units as *ProductionCounter* / hour. Default = unknown.
StatusDetails ?	string	String that defines the device state more specifically. For a list of supported values, see Appendix G.
TotalProductionCounter ?	number	The current total machine production counter. Default = un-known.
Device ?	element	A **Device** resource that describes details of the device.
Employee *	element	**Employee** resources that describe which employees are currently working at the device.
JobPhase *	element	Describes the actual status of jobs in the device. For details on us-ing **JobPhase** elements, see Table 5-45.
ModuleStatus *	element	Status of individual modules. For details on using **ModuleStatus** elements, see Table 5-46.

Structure of the JobPhase Element

A **Status** response may provide **JobPhase** elements. The **JobPhase** element represents the actual state of a job. The **JobPhase** element is an analogue to the **PhaseTime** audit element described in Section 3.10.1.3 PhaseTime. The main difference between a **JobPhase** element and a **PhaseTime** audit element is that a **Phase** message reflects a snapshot of the current job status whereas the **PhaseTime** audit reflects a time span bordered by two (sub-)status transitions.

For exact information about the job phase a **JobPhase** element may embed a copy of the current state of the job described as JDF. If an actual JDF is not supported by the controller, the same rules apply for the **Status** re-sponse as those which apply for the **Consumable** response.

Table 5-45 Contents of the JobPhase element

Name	Data Type	Description
Activation ? New in JDF 1.1	enumeration	The activation of the JDF node. Possible values are the same as the possible values of a JDF node's *Activation* attribute: For details, see Table 3-3 Contents of a JDF node.
Amount ?	number	Produced amount. If *Waste* is also specified, the value is without waste. The unit is specified in the *CounterUnit* attribute of the parent element De-viceInfo.
DeadLine ?	enumeration	Scheduling state of the job. Possible values are: *InTime* – The job or job part will probably not miss the deadline. *Warning* – The job or job part could miss the deadline. *Late* – The job or job part will miss the deadline. Default = *InTime* For more details on scheduling, see Section 3.5 Node Information.
JobID ?	string	Job ID of the JDF node the **JobPhase** belongs to.
JobPartID ?	string	Job part ID of the JDF node the **JobPhase** belongs to.
PercentCompleted ?	number	Node processing progress in % completed.

Name	Data Type	Description
QueueEntryID ?	string	If the job was submitted to a Queue, and the *QueueEntryID* is known, this attribute should be provided.
RestTime ? New in JDF 1.1	duration	Estimated duration required for finishing of this job.
Speed ?	number	The current job speed. *Speed* is defined in the same units as *Production-Counter* / hour. Defaults to the speed specified in the DeviceInfo element.
StartTime ? New in JDF 1.1	dateTime	Time when the job has been started.
Status	enumeration	The status of the JDF node. Possible values are the same as the possible values of a JDF node's *Status* attribute: For details, see Table 3-3 Contents of a JDF node.
StatusDetails ?	string	String that defines the job state more specifically. For a list of supported values, see Appendix G.
TotalAmount ? New in JDF 1.1	number	Amount that will be produced when this job phase is 100% completed. The unit is specified in the *CounterUnit* attribute of the parent element DeviceInfo.
Waste ? New in JDF 1.1	number	Produced amount of waste. The unit is specified in the *CounterUnit* attribute of the parent element DeviceInfo.
CostCenter ?	element	The cost center that the job is currently being charged to. Defaults to the cost center specified in the DeviceInfo element.
JDF ?	element	Complete JDF node that represents a snapshot of the job that is currently being processed.
Part * Modified in JDF 1.1	element	Describes which parts of a job are currently being processed.

Structure of the ModuleStatus Element

The ModuleStatus element is identical to the ModulePhase element of the PhaseTime audit element (see Table 3-33), except that the attributes *Start* and *End* are missing. These attributes specify the time interval in the audit pendant ModulePhase and the *DeviceID* attribute, which is unnecessary here. The ModuleStatus element is described in the following table.

Table 5-46 Contents of the ModuleStatus element

Name	Data Type	Description
DeviceStatus	enumeration	Status of the module. Possible values are: *Unknown* – The module status is unknown. *Idle* – The module is not used. An example is a color print module that is inactive during a black-and-white print. *Down* – The module cannot be used. It may be broken, switched off etc. *Setup* – The module is currently being set up. *Running* – The module is currently executing. *Cleanup* – The module is currently being cleaned. *Stopped* – The module has been stopped, but running may be resumed later. This status may indicate any kind of break, including a pause, maintenance, or a breakdown, as long as running can be easily resumed.
ModuleIndex	IntegerRange-List	0-based indices of the module or modules. If multiple module types are available on one machine, indices must also be unique.
ModuleType	NMTOKEN	Module description. The allowed values depend on the type of device that is described. The predefined values are listed in Appendix H.

Name	Data Type	Description
StatusDetails ?	string	Description of the module status phase that provides details beyond the enumerative values given by the *DeviceStatus* attribute. For a list of supported values, see Appendix G.
Employee *	element	Links to Employee resources that are working at this module (the module is specified by the attributes *ModuleIndex* and *ModuleType*).

The following is an example of a response to a Status query. The device in this example holds one job and executes another job that is currently printed duplex each side on four-color modules for the front and three-color modules for the back, with one idle:

```
<Response ID="M1" refID="Q1" Type="Status">
  <DeviceInfo JobID="678" JobPartID="01" DeviceStatus="Running" StatusDetails="Waste">
    <JobPhase Amount="2560" DeadLine="InTime" JobID="678" JobPartID="01" PercentCom-
pleted="52" QueueEntryID="Job-05" Status="InProgress" StatusDetails="Waste"/>
    <JobPhase Amount="0" DeadLine="Warning" JobID="679" JobPartID="01" PercentCom-
pleted="0" QueueEntryID="Job-06" Status="Ready"/>
    <ModuleStatus ModuleIndex="0~3 6~8" ModuleType="PrintModule" DeviceS-
tatus="Running"/>
    <ModuleStatus ModuleIndex="4" ModuleType="PrintModule" DeviceStatus="Idle"/>
    <ModuleStatus ModuleIndex="5" ModuleType="PerfectingModule" DeviceS-
tatus="Running"/>
  </DeviceInfo>
</Response>
```

5.5.2.4 Track

Table 5-47 Contents of the Track message

Object Type	Element name	Description
QueryTypeObj	TrackFilter ?	Refines the Track query.
ResponseTypeObj	TrackResult *	Details of the tracked jobs

The Track query requests information about the location of Jobs that are known by a controller. If a high level controller controls lower level controllers, it should also list the jobs that are controlled by these. The response is a list of TrackResult elements.

Structure of the TrackFilter Element
The TrackFilter element refines the list of **TrackResult**s that should be returned. Only jobs that match all parameters specified are included.

Table 5-48 Contents of the TrackFilter element

Name	Data Type	Description
JobID ?	string	Job ID of the JDF node that is being tracked. Defaults to list Job-Phase elements of all known nodes.
JobPartID ?	string	JobPart ID of the JDF node that is being tracked.
Status ?	enumerations	The status of the jobs being tracked. Possible values are a combination of any of the possible values of a JDF node's *Status* attribute. Default = all. Possible values are: *Waiting* *Ready* *FailedTestRun* *Setup* *InProgress* *Cleanup*

Name	Data Type	Description
		Spawned
		Stopped
		Completed
		Aborted
		For details, see Table 3-3 Contents of a JDF node.

Structure of the TrackResult Element

One **TrackResult** is returned for each known job or spawned job part. **TrackResult** elements contain information about the location of distributed jobs.

Table 5-49 Contents of the TrackResult element

Name	Data Type	Description
JobID	string	Job ID of the JDF node that is being tracked.
JobPartID ?	string	JobPart ID of the highest level node of the JDF node that is being tracked.
URL	URL	URL of the controller that owns this job.
IsDevice	boolean	If *true*, the controller that emitted this message is the device that has access to the job and may be queried for details of the job.

The following is an example of a response on a Track message:

```
<Response ID="M1" refID="Q1" Type="Track">
  <TrackResult URL="http://www.anycompany.com/controller" JobID="1" JobPar-
tID="42" IsDevice="true"/>
  …
</Response>
```

5.5.3 Pipe Control

JDF Messaging provides methods to control dynamic pipes. Dynamic pipes are described in detail in Section 4.3.2 Overlapping Processing Using Pipes.

Table 5-50 Dynamic pipe messages

Message type	Family	Description
PipeClose	CR	Closes a pipe because no further resources are required. This is typically used to terminate the producing process.
PipePull	CR	Requests a new resource from a pipe.
PipePush	CR	Notifies that a new resource is available in a pipe.
PipePause	CR	Pauses a process if no further resources can be consumed or produced.

5.5.3.1 PipeClose

Table 5-51 Contents of the PipeClose message

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the pipe resource. The PipeParams element is described in Section 5.5.3.2 PipePull.
ResponseTypeObj	JobPhase	The status of the responding process. The JobPhase element is defined in Table 5-45.

The PipeClose message notifies the process at the other end of a dynamic pipe that the sender of this message needs no further resources or will produce no further resources through the pipe. The PipeClose command response is equivalent to the PipePull and PipePush command responses described below.

5.5.3.2   PipePull

Table 5-52 Contents of the PipePull message

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the requested pipe resource.
ResponseTypeObj	JobPhase	The status of the responding process. The Job-Phase element is defined in Table 5-45.

The PipePull message requests resources that are described in a JDF dynamic pipe (see Sections 3.7.3   Pipe Resources and 4.3.2   Overlapping Processing Using Pipes). PipePull messages are the JMF equivalent of a dynamic input resource link. Figure 5.4, below, depicts the mode of operation of a PipePull message.

The PipePull command response returns a *ReturnCode* of 0 if the command has been accepted by the receiving controller. If not successful the *ReturnCode* may be one of the codes presented in Appendix I. The response may contain a Notification element. The JobPhase element (see Section 5.5.2.3 Status) returned should provide only the *Status* attribute that describes the job status of the responding process after receiving the command.
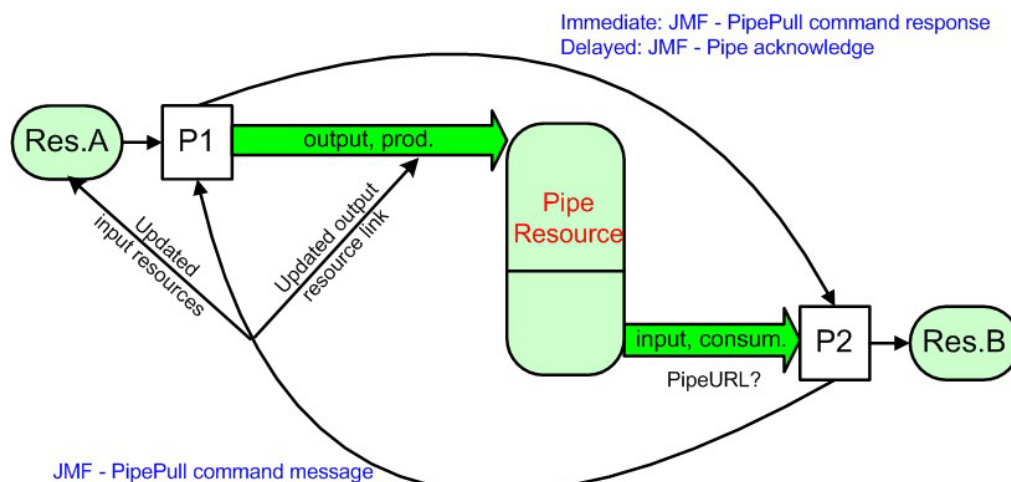
Figure 5.4 Mechanism of a PipePull message

Structure of the PipeParams Element
The PipeParams element is also used by the messages PipeClose, PipePush, and PipePause.

The URL where an optional Acknowledge should be sent when the pipe command has been executed may be defined in the initiating command message by the attribute *AcknowledgeURL*. The Acknowledge is sent for the following commands:

- for PipeClose: when the process has been finished,
- for PipePull: when the resource is available,
- for PipePush: when the resource has been accepted, and
- for PipePause: when the process has been stopped.

Table 5-53 Contents of the PipeParams element

Name	Data Type	Description
PipeID	string	PipeID of the JDF resource that defines the dynamic pipe.
Status ?	enumeration	Process status after the request. Possible values are defined in Table 3-3. Default = *InProgress*

Name	Data Type	Description
Resource *	element	Updated input resources to be used by the process that receives the pipe command: PipePull (the receiver creates the pipe resource), PipePush (the receiver consumes the pipe resource), and PipePause (the receiver only updates the inputs).
		The resource to be updated is identified by the *ID*, that means the *ID* attribute must be known to the controller that issued the pipe command. Possible commands are: PipePull, PipePush, or PipePause. In case of the PipeClose command, the resources are ignored.
ResourceLink ?	element	Updated resource link to the pipe resource: PipePull (it is an output link), PipePush (it is an input link), and PipePause (depends on the pipe end). This resource link may be used by the process that links to the pipe resource.
		The attributes *rRef* and *Usage* of a resource link must not be updated. For details see Section 3.7.4 ResourceUpdate Elements. In the context of dynamic pipes these two attributes have no meaning.
		In case of the PipeClose command, the resource link is ignored.
UpdatedStatus ?	enumeration	This value represents the actual status of the pipe resource and may be used by the receiving process for process termination control. For details see Section Formal Iterative Processing.
		For possible values of the resource *Status* attribute see Table 3-11.

5.5.3.3 PipePush

J. 2 Contents of the PipePush message

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the produced pipe resource. The PipeParams element is described in Section 5.5.3.2 PipePull.
ResponseTypeObj	JobPhase	The status of the responding process. The JobPhase element is defined in Table 5-45.

The PipePush message notifies the availability of pipe resources that are described in a JDF dynamic pipe (see Sections 3.7.3 Pipe Resources and 4.3.2 Overlapping Processing Using Pipes). PipePush messages are the JMF equivalent of a dynamic output resource link. Figure 5.5 depicts the mode of operation of a PipePush message. The PipePush command response is equivalent to the PipePull command response described above.
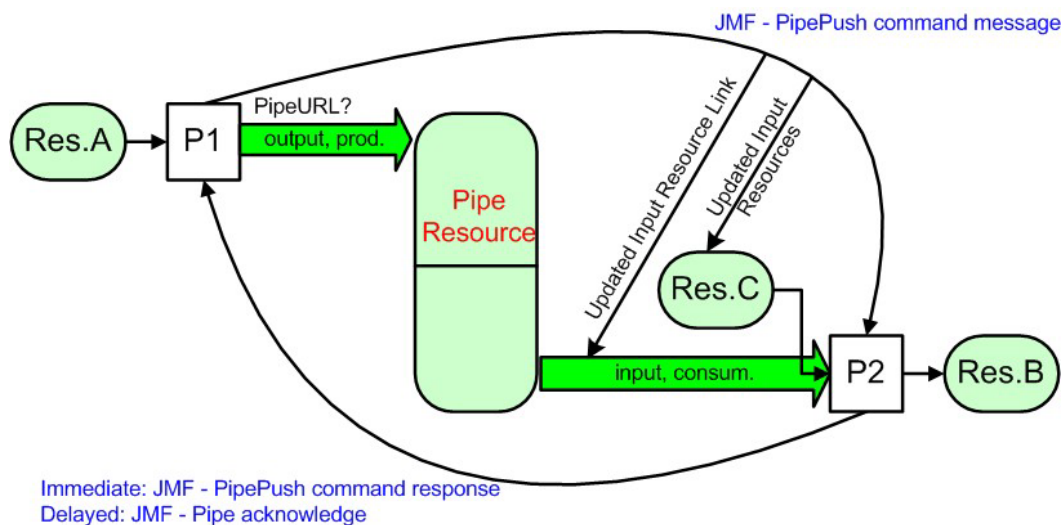
Figure 5.5 Mechanism of a PipePush message

5.5.3.4   PipePause

Table 5-54 Contents of the PipePause message

Object Type	Element name	Description
CommandTypeObj	PipeParams	Describes the pipe resource. The PipeParams element is described in Section 5.5.3.2 PipePull.
ResponseTypeObj	JobPhase	The status of the responding process. The JobPhase element is defined in Table 5-45.

The PipePause message pauses execution of a process that is at the other end of a dynamic pipe. The PipePause command response is equivalent to the PipePull command response described above.

5.6 Queue Support

In JMF, a device is assumed to have one input queue that accepts submitted jobs. If a real device supports multiple queues, it is represented by multiple logical devices in JDF. The simple case of a device with no queue can be mapped to a queue with two *Status* states: *Waiting* and *Full*. JMF supports simple handling of priority queues. The following assumptions are made:

- Queues support priority. Priority may only be changed for waiting jobs. A queue may round priorities to the number of supported priorities, which may be one, indicating no priority handling.

- Priority is described by an integer from 0 to 100. Priority 100 defines a job that should pause a job that is in progress and commence immediately. If a device does not support the pausing of running jobs, it should queue a priority-100 job before the last pending priority-100 job.

- A controller may control multiple devices/queues.

- Queue entries can be unambiguously identified by a *QueueEntryID*.

Some conventions used in the following sections have already been introduced in Section 5.5 Standard Messages. This affects the message families and the descriptive tables at the beginning of each message section that describe the type objects related to the corresponding message. The type objects are QueryTypeObj, CommandTypeObj, and ResponseTypeObj (see also Figure 5.1).

5.6.1   Queue Entry ID Generation

Queue entries are accessed using a *QueueEntryID* attribute, which is generated by the controller of the queue when the job is submitted. This attribute must uniquely identify an entry within the scope of one queue. An implementation is free to choose the algorithm that generates *QueueEntryID*s.

5.6.2   Queue Entry Handling Commands

Queue-entry handling is provided so that the state of individual jobs within a queue can be changed. Job submission, queue-entry grouping, priorities, and hold/resume of entries are all supported. The individual commands are defined in the table and explained in greater detail in the sections that follow.

Table 5-55 QueueEntry handling messages

Message type	Family	Description
AbortQueueEntry	CR	If a job is already running, it is aborted and removed. If it is not already running, it is removed from the queue. AbortQueueEntry is the only Queue manipulation message that has an effect on running queue entries.
HoldQueueEntry	CR	The entry remains in queue but is never executed.

Message type	Family	Description
RemoveQueueEntry	CR	A job is removed from the queue.
ResubmitQueueEntry	CR	Replaces a queue entry without affecting the entry's parameters. The command is used, for example, for late changes to a submitted JDF.
ResumeQueueEntry	CR	A held job is resumed. The job is requeued at the position defined by its current priority. Submission time is set to the current time stamp.
SetQueueEntryPosition	CR	Queues a job behind a given position n, where n represents a numerical value. 0 = pole position. Priority is set to the priority of the job at position n.
SetQueueEntryPriority	CR	Sets the priority of a queued job to a new value. This does not apply to jobs that are already running.
SubmitQueueEntry	CR	A job is submitted to a queue in order to be executed.

5.6.2.1 AbortQueueEntry

Table 5-56 Contents of the AbortQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the elements listed above, see Section 5.6.4.		

Once this command is issued, the entry specified by QueueEntryDef is removed from the queue. If the device on which the entry is running has already commenced processing, the entry is aborted. In this case the Audits and *Status* of the JDF that is being processed should be appropriately filled and the JDF should be delivered to the URL as specified by NodeInfo:*TargetRoute*.

5.6.2.2 HoldQueueEntry

Table 5-57 Contents of the HoldQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the elements listed above, see Section 5.6.4.		

The entry specified by QueueEntryDef remains in the queue but is never executed. The HoldQueueEntry command has no effect on running jobs.

5.6.2.3 RemoveQueueEntry

Table 5-58 Contents of the RemoveQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the elements listed above see, Section 5.6.4.		

This command causes the entry specified by QueueEntryDef to be removed from the queue. It does not affect running jobs.

5.6.2.4   ResubmitQueueEntry

Table 5-59   Contents of the ResubmitQueueEntry message

Object Type	Element name	Description
CommandTypeObj	ResubmissionParams	Defines the job resubmission.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

A job is resubmitted to a queue using the ResubmitQueueEntry message. This allows late changes to be made to a job without affecting queue parameters and without exporting the internal structure of a queue. Resubmission overwrites the job specified in the *URL* attribute of the ResubmissionParams element. The job must not run. Job resubmission does not affect other queue parameters as specified, for example, resubmission does not affect queue ordering.

Structure of the ResubmissionParams Element

Table 5-60 Contents of the ResubmissionParams element

Name	Data Type	Description
QueueEntryID	string	ID of the queue entry to be replaced.
URL	URL	Location of the JDF to be submitted. May be either a URL or, in the case of MIME/Multipart/Related, a CID.

5.6.2.5   ResumeQueueEntry

Table 5-61   Contents of the ResumeQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueEntryDef	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the elements listed above, see Section 5.6.4.		

The hold status of the queue entry specified by QueueEntryDef is removed.

5.6.2.6   SetQueueEntryPosition

Table 5-62   Contents of the SetQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueEntryPosParams	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The position of the queue entry is modified. The QueueEntryPosParams element provides the required parameters.

Structure of the QueueEntryPosParams Element

QueueEntryID specifies the queue entry to be moved. Jobs may either be set to a specific position within the queue or positioned next to an existing queue entry. The priority of the entry matches the priority of the entry that precedes it, after it has been repositioned. Only one of *NextQueueEntryID, PrevQueueEntryID* or *Position* may be specified.

Table 5-63   Contents of the QueueEntryPosParams element

Name	Data Type	Description
NextQueueEntryID ?	string	ID of the queue entry that should be ordered directly behind the entry.

Name	Data Type	Description
QueueEntryID	string	ID of a queue entry. The ID is generated by the queue owner.
PrevQueueEntryID ?	string	ID of the queue entry that should be ordered directly in front of the entry.
Position ?	integer	Position in the queue. 0 = pole position. Note that the position is based on the queue before modification. Thus if a queue entry is moved back in the queue, its final position is one lower than specified in *Position.*

5.6.2.7 SetQueueEntryPriority

Table 5-64 Contents of the SetQueueEntryPriority element

Object Type	Element name	Description
CommandTypeObj	QueueEntryPriParams	Defines the queue entry.
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The priority of the queue entry is modified. The QueueEntryPriParams element provides the required parameters.

Structure of the QueueEntryPriParams Element
QueueEntryID, described in the table below, specifies the queue entry that has its priority modified.

Table 5-65 Contents of the QueueEntryPriParams element

Name	Data Type	Description
Priority	integer	Number from 0 to 100, where 0 = lowest priority and 100 = maximum priority.
QueueEntryID	string	ID of a queue entry. The ID is generated by the queue owner.

5.6.2.8 SubmitQueueEntry

Table 5-66 Contents of the SubmitQueueEntry message

Object Type	Element name	Description
CommandTypeObj	QueueSubmissionParams	Defines the job submission.
ResponseTypeObj	QueueEntry	Provides the queue entry of the submitted job.
	Queue	Describes the state of the queue after the command has been executed.
Definition of the QueueEntry and Queue elements, see Section 5.6.4.		

The SubmitQueueEntry message submits a job to a queue. The QueueSubmissionParams element provides the required parameters.

Structure of the QueueSubmissionParams Element
The job submission may contain queue-ordering attributes equivalent to those used by the SetQueueEntryPriority and SetQueueEntryPosition messages. The *URL* attribute specifies the location where the JDF file to be submitted can be retrieved by the queue controller. The location type in the *URL* attribute (such as *File*, *http* or *CID*) defines the submission method. The optional *ReturnURL* attribute specifies the location where the modified JDF should be sent after the job is completed or aborted.

Table 5-67 Contents of the QueueSubmissionParams element

Name	Data Type	Description
Hold ?	boolean	If *true*, the entry is submitted as held. Default = *false*

Name	Data Type	Description
NextQueueEntryID ?	string	ID of the queue entry that should be ordered directly behind the entry.
PrevQueueEntryID ?	string	ID of the queue entry that should be ordered directly in front of the entry.
Priority ?	integer	Number from 0 to 100, where 0 = lowest priority and 100 = maximum priority. Default = 1
ReturnURL ?	URL	URL where the JDF file should be sent when the job is completed or aborted. If not specified, the JDF should be placed in the default output hot folder of the queue controller.
URL	URL	Location of the JDF to be submitted. In the case of MIME/Multipart/Related, the location may be either a URL or a CID.
WatchURL ?	URL	URL of the controller that should be notified when the status of the **QueueEntry** changes. Specifying this URL is the equivalent of sending a **QueueEntryStatus** query with a persistent channel and *ChangeAttribute* = "*" to this URL.

File Submission
If the URL defines a file, the controller may retrieve the file at the location specified in the *URL* attribute.

The following example declares a file on the network:

```
<Command Type="SubmitQueueEntry" >
  <QueueSubmissionParams URL="File:///c:/AnyDirectory/job1.jdf"/>
</Command>
```

HTTP External JDF Submission
The following example declares an intranet or Internet location. In this example, the queue controller can retrieve the file with a standard HTTP get command. Note that the job itself may be a MIME/Multipart entity. It may also be dynamically generated by a CGI script or another such tool.

```
<Command Type="SubmitQueueEntry" >
  <QueueSubmissionParams  URL="http://JobServer.JDF.COM?job1"/>
</Command>
```

HTTP MIME/Multipart/Related Submission
If a message controller is capable of decoding MIME, it is legal to submit a MIME/Multipart/Related message. The first section of the multipart MIME document must be the JMF submission command. Internal links are defined using the Content-ID (CID) label in MIME. The second section must be the JDF job. Subsequent sections are the linked entities, such as the preview images shown in the following example:

```
MIME-Version: 1.0
Content-Type: multipart/Related; boundary=unique-boundary

--unique-boundary
Content-type: text/xml
…
<JMF TimeStamp="2000-06-12T08:56+02:00" SenderID="JobCreator P_01">
<Command ID="Cmd-0234" Type="SubmitQueueEntry"">
<QueueSubmissionParams URL="CID:JDF1/>
</Command>
</JMF>
…

--unique-boundary
Content-type: text/xml
Content-ID: JDF1

<JDF … >

--unique-boundary
```

```
Content-type: image/png
Content-ID: Yellow-PNG-Page1

png image of a separation may be here

--unique-boundary--
```

5.6.3 Global Queue Handling

Whereas the commands in the preceding section change the state of an individual queue entry, the commands in this section modify the state of an entire queue. Note that entries that are executing in a device are not affected by the global queue-handling commands and must be accessed individually. An individual queue can be selected by specifying the target device/queue in the *DeviceID* attribute of the JMF root. If no *DeviceID* is specified, the commands or queries are applied to all devices/queues that are controlled by the controller that received the message. The following individual messages are defined:

Table 5-68 Global queue-handling commands

Message type	Family	Description
CloseQueue	CR	The queue is closed. No jobs may be accepted by the queue.
FlushQueue	CR	All entries in the queue are removed.
HoldQueue	CR	The queue is held. No jobs within the queue may be executed.
OpenQueue	CR	The queue is opened. Jobs may be accepted.
QueueEntryStatus	QRS	Returns a QueueEntry element.
QueueStatus	QRS	Returns the Queue elements that describe a queue or set of queues.
ResumeQueue	CR	The queue is activated and queue entries may be executed.
SubmissionMethods	QR	Queries a list of supported submission methods to the queue.

5.6.3.1 CloseQueue

Table 5-69 Contents of the CloseQueue message

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The queue is closed. No further queue entries are accepted by the queue. The status of entries that are already in the queue remains unchanged and prior entries may be executed.

5.6.3.2 FlushQueue

Table 5-70 Contents of the FlushQueue message

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

All queue entries in the queue are removed. Only pending queue entries may be removed.

5.6.3.3 HoldQueue

Table 5-71 Contents of the HoldQueue message

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The queue is held. No entries may be executed. Note that the status of a held entry prior to HoldQueue is retained so that held jobs should remain held after a ResumeQueue. New entries may, however, still be submitted to a held queue.

5.6.3.4 OpenQueue

Table 5-72 Contents of the OpenQueue message

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The queue is opened and new queue entries may be accepted by the queue. A held queue remains held. The OpenQueue command is the opposite of a CloseQueue command.

5.6.3.5 QueueEntryStatus

Table 5-73 Contents of the QueueEntryStatus message

Object Type	Element name	Description
QueryTypeObj Modified in JDF 1.1A	QueueEntryDefList	Defines the addressed queue entries. Note that this element was QueueEntryDef * prior to JDF1.1A.
ResponseTypeObj	QueueEntry *	Describes the status of the queried queue entries.
For the definition of the elements above see Section 5.6.4.		

The QueueEntryStatus message returns queue entry descriptions. The QueueEntryDef elements specify the queue entries to be queried. If no QueueEntryDef element is specified, the query returns a list of QueueEntry elements, one for each entry in the queue. If no QueueEntryDef is specified and the query defines a persistent channel, a Signal is emitted for any entry whose status changes. This includes changes as a result of modifications of the queue status, such as hold or resume.

Structure of the QueueEntryDefList Element
New in JDF 1.1A

The QueryTypeObj of QueueEntryStatus has been modified from QueueEntryDef* to QueueEntryDefList because of a type collision in the XML Schema. QueueEntryDef had been used both as a QueryTypeObj and as a. CommandTypeObj.

Table 5-74 Contents of the QueueEntryDefList element

Name	Data Type	Description
QueueEntryDef *	element	Defines the addressed queue entries.

5.6.3.6 QueueStatus

Table 5-75 Contents of the QueueStatus message

Object Type	Element name	Description
QueryTypeObj	-	

Object Type	Element name	Description
ResponseTypeObj	Queue	Describes the status of the queue.
For the definition of the Queue element, see Section 5.6.4.		

Returns a queue description.

5.6.3.7 ResumeQueue

Table 5-76 Contents of the ResumeQueue message

Object Type	Element name	Description
CommandTypeObj	-	
ResponseTypeObj	Queue	Describes the state of the queue after the command has been executed.
For the definition of the Queue element, see Section 5.6.4.		

The queue is activated and queue entries may be executed. The ResumeQueue command is the opposite of a HoldQueue command.

5.6.3.8 SubmissionMethods

Table 5-77 Contents of the SubmissionMethods message

Object Type	Element name	Description
QueryTypeObj	-	
ResponseTypeObj	SubmissionMethods ?	Describes the submission methods supported by the queue.

The SubmissionMethods message returns the submission methods that are supported by a queue controller.

Structure of the SubmissionMethods Element

The response element may contain multiple attributes, as defined below. If an attribute is not specified, the corresponding submission method is not supported.

Table 5-78 Contents of the SubmissionMethods element

Name	Data Type	Description
File ?	boolean	Can retrieve a JDF from a File specified in the URL Default = *false*
HotFolder ?	URL	URL specification of a hot folder location. Default = *no hot folder*
HttpGet ?	boolean	Can retrieve a JDF via HTTP get commands. Default = *false*
MIME ?	boolean	Accepts MIME/Multipart/Related submission messages via a message post. Default = *false*

The following is an example of a response to a SubmissionMethods query:

```
<Response ID="M1" refID="Q1" Type="SubmissionMethods"/>
  <SubmissionMethods File="true"
    HotFolder="File://MyDevice/HotFolder" HttpGet="true" MIME="false"/>
</Response>
```

5.6.4 Queue-Handling Elements

In this section elements used by queue-handling commands are defined. The following table shows the resulting status of a queue in dependence on global queue commands CloseQueue/OpenQueue and HoldQueue/ResumeQueue as well as the load of queue and its processor. The first command pair determines the logical state of the first column "Closed" and the second of the column "Held". The queue is held if the queue manager doesn't send existing entries to the queue's processor.

Table 5-79 Definition of the Queue Status Attribute values

Closed	Held	Queue Full	Processor Full	Status
Yes	Yes	Any	Any	*Blocked*
Yes	No	Any	Any	*Closed*
No	Yes	Any	Any	*Held*
No	No	Any	No	*Waiting*
No	No	No	Yes	*Running*
No	No	Yes	Yes	*Full*
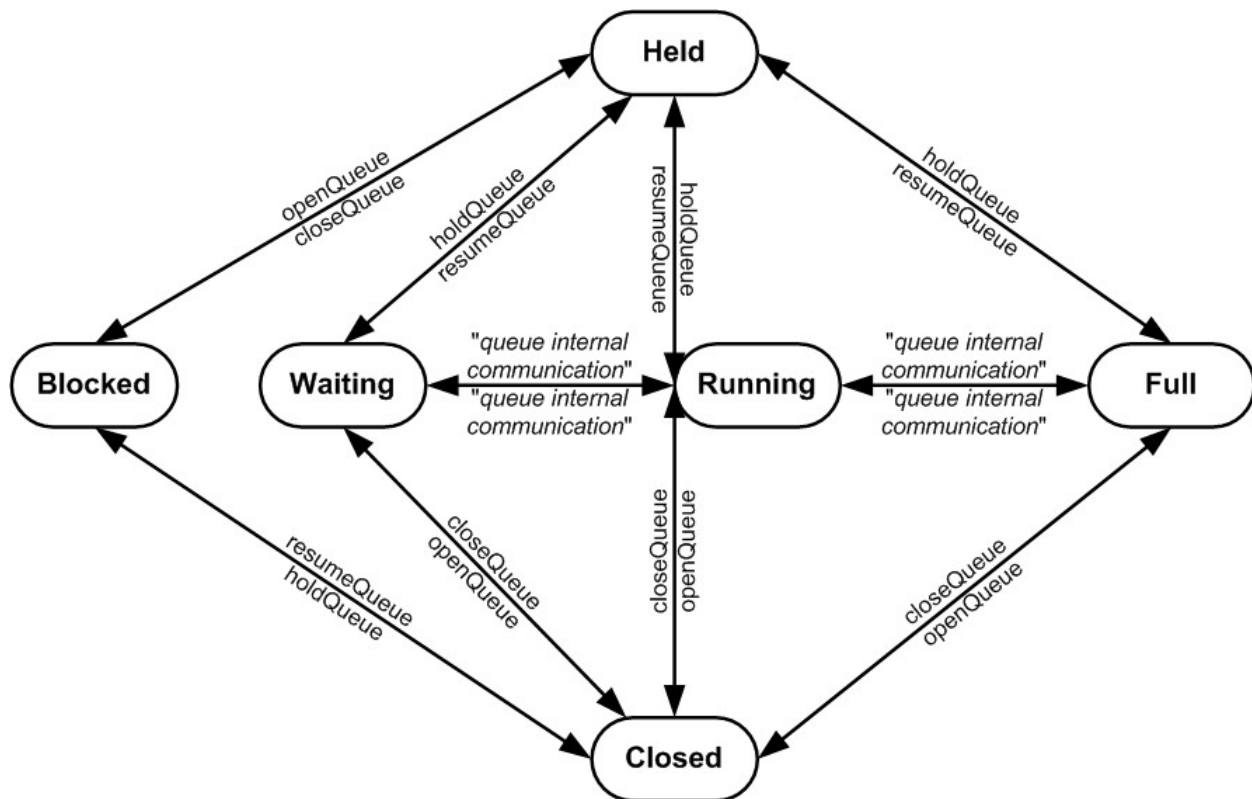
Figure 5.6 Effects of the global queue messages on the queue Status

Structure of the Queue Element

The attributes in the following table are defined for Queue message elements. Queue elements represent the queue of a device including QueueEntry elements that represent both pending and running queue entries.

Table 5-80 Contents of the Queue element

Name	Data Type	Description
Status	Enumeration	Status of the queue. Possible values are:
		Blocked – Queue is completely inactive. No entries may be added and no entries are executed. The queue is closed and held. The queue requires an interaction like OpenQueue or ResumeQueue to reactivate it.
		Closed – Queue entries that are in the queue are executed, but no new entries may be submitted. The lock must be removed explicitly by the OpenQueue command.
		Full – Queue entries that are in the queue are executed but no new entries may be submitted. The lock is removed by the queue controller as soon as it is able to do so.
		Running – A process is executing. Entries may be submitted and will be executed when they reach their turn in the queue.
		Waiting – Queue accepts new entries and has free resources to immediately commence processing.
		Held – Entries may be submitted but will not be executed until the queue is resumed by the ResumeQueue command.
DeviceID	String	Identifies the queue/device.
Device *	Element	The devices that execute entries in this queue.
QueueEntry *	element	Queue entry elements (see Table 5-81 , below). The entries are ordered in the sequence they will be executed, beginning with the running entries.

Example of a Queue message element:

```
<Queue Status="Running" DeviceID="Q12345">
  <QueueEntry QueueEntryId="111-1" Priority="1" Status="Running" JobId="111" JobPar-
tId="1"/>
  <QueueEntry QueueEntryId="111-2" Priority="1" Status="Waiting" JobId="111" JobPar-
tId="2"/>
  <QueueEntry QueueEntryId="112-1" Priority="55" Status="Held" JobId="112" JobPar-
tId="1"/>
</Queue>
```

Structure of the QueueEntry Element

Table 5-81 Contents of the QueueEntry element

Name	Data Type	Description
JobID ? Modified in JDF 1.1	string	The Job ID of the JDF process.
JobPartID ?	string	The JobPartID of the JDF process.
Priority ?	integer	Priority of the QueueEntry. Values are 0-100. 0 = lowest priority, while 100 = highest priority. Default = 1
QueueEntryID	string	ID of a QueueEntry. This ID is generated by the queue owner.
StartTime ? New in JDF 1.1	dateTime	Time when the job has been started.
Status Modified in JDF 1.1A	enumeration	Status of the individual entry. Possible values are: *Running* – The queue entry is running. *Waiting* – The queue entry is waiting and will be executed when resources are available. *Held* – The queue entry is held and will not execute until resumed. *Removed* – The queue entry has been removed. This status can only be sent when a persistent channel watches a queue and the

Name	Data Type	Description
		queue entry is removed.
SubmissionTime ?	dateTime	Time when the entry was submitted to the queue.

Structure of the QueueEntryDef Element

The element specifies a queue entry and is used to refer to a certain queue entry.

Table 5-82 Contents of the QueueEntryDef element

Name	Data Type	Description
QueueEntryID	string	ID of the queue entry. The ID is generated by the queue owner.

5.7 Extending Messages

This specification defines a set of predefined messages for general usage. Extensions to existing messages and additional message types may be defined using the standard extension rules described in JDF Extensibility. Note, the generic content of Section 3.1.1 Generic Contents of JDF Elements is also valid for JMF elements. It is not allowed to define message extensions which duplicate the functionality of messaging types, messaging elements, or message attributes that are already defined in this specification.

For example the content of the *Type* attribute may be specified with a prefix that identifies the organization that defined the extension. The prefix and name should be separated by a single colon (':'). Any additional attributes and elements are allowed, and internal elements may be declared with explicit namespaces. The official namespace of JMF elements is xmlns="http://www.CIP4.org/JDFSchema_1_1". This namespace is identical to that defined for JDF in JDF Extensibility. An example is provided:

```
<JMF … xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:Circus="Circus Schema URI">
  <Query Type="Circus:IsClownHappy" ID="Q1">
    <Circus:ClownParams Gender="male"/>
  </Query>
</JMF>
```

The response will also have the "Circus:" namespace identifier. All Circus elements are explicitly declared.

```
<JMF … xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:Circus="Circus Schema URI">
  <Response ID="M1" refID="Q1" Type="Circus:IsClownHappy">
    <Circus:Clown name="Joe" happy="true">
    <Circus:Clown name="John" happy="false">
  </Response>
</JMF>
```

5.7.1 IfraTrack Support

The extending mechanism can be used to implement compatibility with other XML-based messaging standards, for example version 3.0 of IfraTrack. The *Type* attribute is set to the appropriate namespace, and

More on IfraTrack

IfraTrack is a specification for the interchange of status and management information between local and global production management systems in newspaper production. For more information on IfraTrack, including a case study paper, please see http://www.ifra.com/WebSite/news.nsf/(StructuredSearchAll)?OpenAgent&IFRATRACK

the foreign message is included, as demonstrated in the following example:

```
<JMF … xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:IFRA="IfraTrack URI">
  <Query ID="Q1" Type="IFRA:IMF">
    <IMF xmlns="IfraTrack URI">
      Whatever you want (may be multiple top level elements)
    </IMF>
  </Query>
</JMF>
```

The legal response would be:

```
<JMF … xmlns="http://www.CIP4.org/JDFSchema_1_1" xmlns:IFRA="IfraTrack URI">
  <Response ID="M1" refID="Q1" Type="IFRA:IMF">
    <IMF xmlns="IfraTrack URI">
      The appropriate IFRA response(s)
    </IMF>
  </Response>
</JMF>
```

Note that the application is free to select the appropriate response types in order to fulfill its local (IfraTrack) proto-
col requirements if it uses its own namespace. In the examples above the default namespace associated with the
IMF query and response elements has been overwritten by the Ifra-namespace. Additional information on using
IfraTrack and JDF is in Appendix E Modeling IfraTrack in JDF.

Chapter 6 Processes

The following chapter describes the processes that are defined in detail for JDF.

6.1 Process Template

Processes are defined by their input and output resources, therefore, all relevant resource information is provided in tables for each process. Furthermore, although they are not listed for each process, additional, optional input resources as defined in the following table as well as any implementation resources are implied for all processes defined in this chapter.

> **The JDF Cookbook**
>
> Chapter 6 and Chapter 7 is "the list of ingredients" in the JDF "cookbook." The following processes and resources are fairly exhaustive. You can choose to use only what fits your workflow.

Input Resources

Name	Description
Resource	Represents any input resource. If an optional resource is not specified in a JDF instance, the JDF Consumer may make its own assumption regarding attributes and subelements of the resource. Specification defined attribute defaults cannot be guaranteed.
Res1 (usage1)	A resource of type Res1 with the *ProcessUsage* attribute *usage1*
Res1 (usage2)	A resource of type Res1 with the *ProcessUsage* attribute *usage2*
ApprovalSuccess *	Any number of **ApprovalSuccess** resources may be appended to processes in order to model proofing and verification requirements. This is implied and not specified explicitly in the tables in the following section. For more information on the *Approval* process, see Section 6.2.1.
Implementation *	Abstract resource that is a placeholder for any implementation resource (examples are **Employee, Device** or **Tool**) that is associated with processing this node.
Preview * Added in JDF1.1A	Any number of previews may be associated with a process and used for display purposes.

Output Resources

Name	Description
Resource	Represents any output resource.

6.2 General Processes

6.2.1 Approval

The *Approval* process can take place at various steps in a workflow. For example, a resource, such as a printed sheet or a finished book, is used as the input to be approved, and an **ApprovalSuccess** (given, for example, by a customer or foreman) is produced. Combining the *Approval* process with any other process can be used to represent a request for a receipt.

Input Resources

Name	Description
ApprovalParams	Details of the approval process.
Resource *	The resources to be proofed. The input will most often be a resource of class *Handling* or *Quantity*.

Output Resources

Name	Description
ApprovalSuccess	Result of any proofing process given, for example, by a customer or foreman. Note that **ApprovalSuccess** resources are only available on success.
Resource * (Accepted)	Represents the input resources that have been accepted for further processing by the approval process as output resources. This is typically used to transfer the resource *Status* of *Draft* to *Available* (see also Formal Iterative Processing).
Resource * (Rejected)	Represents the input resources that have been rejected for further processing by the approval process as output resources. This may be used to define additional processing for rejected resources.

6.2.2 Buffer
New in JDF 1.1

The **Buffer** process is used to buffer a resource for a certain time period. This can be buffering of a complete resource or of a partial resource, e.g., in a pipe. The quantity of the input and output of resources should be equal. Waiting for printed material to dry before finishing is an example of the **Buffer** process.

Input Resources

Name	Description
BufferParams	The parameters, e.g. times and locations of the **Buffer** process.
Resource	The physical resources to be buffered. These may be any resource whose class is Consumable, Handling or *Quantity*.

Output Resources

Name	Description
Resource	The same resource after buffering. The resource must have a class of Consumable, Handling, or Quantity.

6.2.3 Combine
The **Combine** process is used to combine multiple physical resources or logical resources, e.g., **RunList**s of the same content to form one resource. The quantity of the input and output of resources should be equal. The ordering of the input ResourceLinks must be honored.

Input Resources

Name	Description
Resource +	The resources to be combined.

Output Resources

Name	Description
Resource	Result of combining. The resource formed as a result of the Combine process.

6.2.4 Delivery
This process can be used to describe the delivery of a physical resource to or from a location. This delivery may be internal—meaning within the company—or to an external company or customer. The CustomerInfo element of the JDF node can also be used if the delivery to is to be made to only one customer. Note that a delivery receipt can be requested by combining the **Delivery** process with an **Approval** process.

Input Resources

Name	Description
DeliveryParams	Necessary information about the item or items to be delivered is stored here.
Resource	Any resource delivered to a location. This can be a physical resource or a Parameter resource that is delivered electronically.

Output Resources

Name	Description
Resource	Any resource picked up from a location. This can be a physical resource or a Parameter resource that is delivered electronically.

6.2.5 ManualLabor

New in JDF 1.1

This process can be used to describe any process where resources are handled manually. The *ManualLabor* process is designed to monitor any type of non-automated labor from an MIS system.

Input Resources

Name	Description
Resource *	Resources that are required to create the output Resource.
ManualLaborParams	Details on the *ManualLabor* process.

Output Resources

Name	Description
Resource	The resource that was created by manual work. In general these will be components, but handling resources may also be created manually.

6.2.6 Ordering

This process can be used to describe the *Ordering* (requisition) of a Resource element. Orders can be placed internally, i.e., within the company, or externally.

Input Resources

Name	Description
OrderingParams	Necessary information about the items to be ordered, such as the supplier address, item quantity, or unit type.

Output Resources

Name	Description
Resource + Modified in JDF 1.1	All kinds of physical resources can be ordered.

6.2.7 Packing

Deprecated in JDF 1.1

This process can be used to describe the *Packing* of a PhysicalResource element for transport purposes. The *Packing* process has been deprecated in version 1.1 and beyond. It is replaced by the individual processes defined in Section 6.5.45.5 Packaging Processes.

Input Resources

Name	Description
PackingParams	Necessary information about the packing process.
PhysicalResource	All kinds of physical resources can be packed.

Output Resources

Name	Description
PhysicalResource	The packaged physical resources. Note that *Amount* attributes referring to this resource still refer to individual products and not to boxes, cartons or pallets.

6.2.8 ResourceDefinition

This process can be used to describe the interactive or automated process of defining resources such as set-up information. This process creates output resources or modifies input resources of the same type as the output resources. The *ResourceDefinition* process is designed to monitor interactive work such as creating imposition templates. It can also be used to model a hot folder process that accepts resources from outside of a JDF based workflow.

Input Resources

Name	Description
Resource * Modified in JDF 1.1	Any type of resource. Generally these will be templates.
ResourceDefinitionParams ?	Details on how to handle defaults.

Output Resources

Name	Description
Resource + Modified in JDF 1.1	The same type of resource as the input.

6.2.9 Split

This process is used for splitting one physical or logical resource into multiple physical or logical resources containing the same content as the original. The quantity of the input and output of resources should be equal.

Input Resources

Name	Description
Resource	The resource to be split.

Output Resources

Name	Description
Resource +	The resources formed as a result of splitting.

6.2.10 Verification

The *Verification* process is used to confirm that a process has been completely executed. In the case of variable data printing, in which every document is unique and must be validated individually, database access is required. Verification in this situation may involve scanning the physical sheet and interpreting a bar code or alphanumeric characters. The decoded data may then be either recorded in a database to be later cross referenced with a verification list, or cross referenced and validated immediately in real time.

Input Resources

Name	Description
DBSchema ?	Schema description of the cross-reference database.

Name	Description
DBSelection ?	Database link that defines the database that contains cross-reference data.
IdentificationField *	Identifies the position and type of data for an automated, OCR-based verification process.
VerificationParams	Controls the verification requirements.

Output Resources

Name	Description
ApprovalSuccess ?	Signature file that defines verification success.
DBSelection ?	Database link where the verification data should be recorded.

6.3 Prepress Processes

6.3.1 ColorCorrection

ColorCorrection is the process of modifying the specification of colors in documents to achieve some desired visual result. The process may be performed to ensure consistent colors across multiple files of a job or to achieve a specific design intent, e.g., "Brighten the image up a little".

   ColorCorrection is distinct from **ColorSpaceConversion**, which is the process of changing how the colors specified in the job will be produced on paper. Rather, **ColorCorrection** is the process of modifying the desired result, whatever the specified colorspace might be.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies the assumed color model for the job.
ColorCorrectionParams New in JDF 1.1	Parameters of the **ColorCorrection** process
RunList	List of content elements that are to be operated on.

Output Resources

Name	Description
RunList	List of color-corrected pages.

6.3.2 ColorSpaceConversion

ColorSpaceConversion, as the name implies, is the process of converting all colors used in the job to a known colorspace. There are two ways in which a controller can use this process to accomplish the color conversion. It can simply order the colors to be converted by the device assigned to the task, or it can request that the process simply tag the input data for eventual conversion. Additionally, the process may remove all tags from the content.

   The parameters of this resource provide the ability to selectively control the conversion or tagging of graphical objects based on object class and/or incoming color space.

   Like all other color manipulation supported in JDF, the color conversion controls are based on the use of ICC profiles. While the assumed characterization of input data can take many forms, each can internally be represented as an ICC profile. In order to perform the transformations, input profiles must be paired with the identified final target device profile to create the transformation.

In order to avoid the loss of black color fidelity resulting from the transformation from a four-component CMYK to a three-component interchange space, the agent may select a DeviceLink[1] profile as the assumed color space characterization. In these instances, the final target profile is ignored. Since there is no algorithmic way to determine that the output characterization in a device link profile is equivalent to another profile, some of the responsibility to select a sensible combination falls on the agent or end user.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies the assumed color model for the job.
ColorSpaceConversionParams	Parameters that define how colorspaces will be converted in the file.
RunList	List of pages on which to perform the selected operation.

Output Resources

Name	Description
ColorantControl ?	Identifies the assumed color model for the job. The ColorantControl resource may be modified by a ***ColorSpaceConversion*** Process.
RunList	List of pages on which the selected operation has been performed.

6.3.3 ContactCopying

New in JDF 1.1

ContactCopying is the process of making an analog copy of a film onto a another film or plate. It includes ***Film-ToPlateCopying*** as defined in JDF 1.1.

Input Resources

Name	Description
ContactCopyParams	The settings of the exposure task.
DevelopingParams ?	Controls the physical and chemical specifics of the media development process.
ExposedMedia +	The film or films to be copied onto the plate.
Media	The unexposed plate.
TransferCurvePool?	Area coverage correction and coordinate transformations of the device.

Output Resources

Name	Description
ExposedMedia	The resulting exposed contact copy.

6.3.4 ContoneCalibration

This process specifies the process of contone calibration. It consumes contone raster data, such as that output from an interpreting and rendering process. It produces contone raster data which has been calibrated to a press using a well defined screening process.

---

[1] DeviceLink profiles are ICC profiles that map directly from one device color space to another device color space. Therefore, it represents a one-way link or connection between devices. Examples for DeviceLink profiles are CMYK to CMYK print process conversions or RGB to CMYK color separations.

Input Resources

Name	Description
RunList	Ordered list of rasterized **ByteMaps** representing pages or surfaces.
ScreeningParams ? Modified in JDF 1.1	Parameters specifying which halftoning mechanism is to be applied and with what specific controls.
TransferFunctionControl ? Modified in JDF 1.1	Specifies which calibration to apply.

Output Resources

Name	Description
RunList	Ordered list of rasterized **ByteMaps** representing pages or surfaces.

6.3.5 DBDocTemplateLayout

This process specifies the creation of a master document template that is used as an input resource for the *DBTemplateMerging* process. It is similar to the *LayoutElementProduction* process except that the output is a set of document templates. Document template are represented in JDF as **LayoutElement** resources with *Template* = *true*.

Input Resources

Name	Description
LayoutElement *	Page elements without links to a database.
DBRules	Description of the rules that should be applied to database records in order to generate graphic output.
DBSchema	Database schema that describe the structure of data in the database.

Output Resources

Name	Description
LayoutElement *	The document template is a **LayoutElement** with links to a database. These links are proprietary to the linking application and are not described in JDF. The *Template* attribute must be *true*.

6.3.6 DBTemplateMerging

This process specifies the creation of personalized PDL instance documents by combining a document template and instance data records from a database. The resulting instance documents will generally be consumed by an *Imposition*, a *RIP'ing*, and ultimately by a *DigitalPrinting* process.

Input Resources

Name	Description
DBMergeParams	Parameters of the merge process.
DBSelection	Instance database records to be merged into the document.
LayoutElement *	Document template page element with internal links to a database.

Output Resources

Name	Description
RunList	Page element without links to a database. This element usually contains a printable **LayoutElement** resource such as PPML, PDF or even plain ASCII.

6.3.7 FilmToPlateCopying

Deprecated in JDF 1.1

FilmToPlateCopying has been replaced by the more generic *ContactCopying.*
FilmToPlateCopying is the process of making an analog copy of a film onto a printing plate.

Input Resources

Name	Description
DevelopingParams ?	Controls the physical and chemical specifics of the media development process.
ExposedMedia	The film or films to be copied onto the plate.
Media	The unexposed plate.
PlateCopyParams	The settings of the exposure task.

Output Resources

Name	Description
ExposedMedia	The resulting exposed plate.

6.3.8   FormatConversion

New in JDF 1.1
The *FormatConversion* process controls the conversion from one document type to another, for instance TIFF to BMP.

Input Resources

Name	Description
FormatConversionParams	Set of parameters required to control the *FormatConversion* process.
RunList	List of documents and/or pages to be converted.

Output Resources

Name	Description
RunList	List of documents and pages that have been converted.

6.3.9   ImageReplacement

This process provides a mechanism for manipulating documents that contain referenced image data. It allows for the "fattening" of files that simply contain a reference to external data or contain a low resolution proxy. Additionally, the **ImageReplacementParams** resource can be specified so that this process generates proxy images from referenced data. *ImageReplacement* is intentionally neutral of the conventions used to identify the externally referenced image data.

Input Resources

Name	Description
ImageCompressionParams ? New in JDF 1.1	This resource provides a set of controls that determines how images will be compressed in the resulting "fat" PDF pages.
ImageReplacementParams	Describes the controls selected for the manipulation of images.
RunList	List of page contents on which to perform the selected operation.

Output Resources

Name	Description
RunList	List of page contents with images that have been manipulated as indicated by the **ImageReplacementParams** resource.

6.3.10 ImageSetting

The image recording process is executed by an imagesetter or platesetter that images a bitmap onto the film or plate media.

Input Resources

Name	Description
DevelopingParams ? New in JDF 1.1	Controls the physical and chemical specifics of the media development process.
ImageSetterParams ? Modified in JDF 1.1	Controls the device specific features of the imagesetter.
Media	The unexposed media.
RunList	Identifies the set of bitmaps to image. May contain bytemaps or images.
TransferCurvePool ? New in JDF 1.1	Area coverage correction and coordinate transformations of the device.

Output Resources

Name	Description
ExposedMedia	The exposed media resource.

6.3.11 Imposition

The *Imposition* process is responsible for combining several pages of input graphical content on to a single surface whose dimensions are reflective of the physical output media. Printer's marks can be added to the surface in order to facilitate various aspects of the production process. Among other things, these marks are used for press alignment, color calibration, job identification, and as guides for cutting and folding.

Note that the *Imposition* process specifies the task of combining pages and marks on sheets. The task of setting up the parameters needed for *Imposition*, e.g., **Layout,** is defined either by *LayoutPreparation* or by the generic *ResourceDefinition* process.

There are two mechanisms provided for controlling the flow of page images onto **Media**. The default mechanism, which provides the functionality of **Layout** in PJTF, explicitly identifies all page content for each Sheet imaged and references these pages by means of the Documents and/or MarkDocuments array. Setting the *Automated* attribute of the **Layout** resource to *true* activates a template approach to printing and relies upon the full Documents hierarchy to specify the page content to image. Automated impositioning is equivalent to the PrintLayout functionality in PJTF.

In JDF, there is a single **Layout** resource definition. Its structure is broad enough to encompass the needs of both fully specified and template-driven imposition. When described fully, the **Layout** resources include an array of Signatures. Each Signature in turn specifies an array of Sheets, and each Sheet can have up to two Surfaces (*Front* and *Back*), on which the page images and any marks are to be placed using PlacedObjects. A Sheet that specifies no Surface content will be blank. Pages that are to be printed must be placed onto Surfaces using ContentObject subelements which explicitly identify the page (via the *Ord* attribute which specifies an index into the document **RunList**). Thus, the **Layout** hierarchy specifies explicitly which pages will be imaged.

When describing automated imposition, **Layout** resources specify a single Signature of Sheet(s) where page contents are imaged. The (virtual) sequence of pages which is to be imaged via automated layout is defined by the Document **RunList**. Pages are drawn in order from this sequence to satisfy the ContentObjects in the Surfaces for the Signature in the **Layout**, and the Signature is repeated until all pages of the sequence are consumed. Each time the Signature is repeated, pages are consumed in "chunks" whose size are determined by the value of *MaxOrd* + 1 (if present in the **Layout**), or by the largest *Ord* value or calculated *OrdExpression* value for any ContentObject in the Signature (if *MaxOrd* is absent).

Attributes of the **Media** are given for each Sheet used in printing. Because the same Signature is repeated until all pages are consumed, the **Layout** hierarchy can provide hints or preferences about special needs for sets of page content via InsertSheet elements. Inserting media is a way to separate sections of the document content. Thus alternate content is printed only as necessary to fill areas which would normally have page content because new media has been added or to designate where a document section will begin as specified by the odd or even position of the Signature.

In a JDF model, impositioning is defined separately from other processes, which may precede or follow it. A *Combined* node may combine **Imposition** with other processes (such as **Separation** or **Interpreting**) to describe a device that happens to perform both in a single execution module.

Input Resources

Name	Description
Layout	A **Layout** resource that indicates how the content pages from the Document **RunList** and marks from the Marks **RunList** (see below) are combined onto imposed surfaces.
RunList (Document)	Structured list of incoming page contents which is transformed to produce the imposed surface images.
RunList ? (Marks)	Structured list of incoming marks. These are typically printer's marks such as fold marks, cut marks, punch marks, or color bars.

Output Resources

Name	Description
RunList	Structured list of imposed surfaces. The *Type* of the **LayoutElement**s must all be *Surface*. Typically the output **RunList** will be partitioned by *PartIDKeys* = "*SheetName Side Separation*". If the **Imposition** process is executed before RIPping, this **RunList** will generally be consumed by an **Interpreting** process. In the case of post-RIP **Imposition** it will be consumed by **DigitalPrinting** or **ImageSetting**.

6.3.12 InkZoneCalculation

The **InkZoneCalculation** process takes place in order to preset the ink zones before printing. The **Preview** data are used to calculate a coverage profile that represents the ink distribution along and perpendicular to the ink zones within the printable area of the preview. The **InkZoneProfile** can be combined with additional, vendor-specific data in order to preset the ink zones and the oscillating rollers of an offset printing press.

Input Resources

Name	Description
InkZoneCalculationParams	Specific information about the printing press geometry(such as the number of zones) to calculate the **InkZoneProfile**.
Layout ? New in JDF 1.1	Specific information about the **Media** (including type and color) and about the **Sheet** (placement coordinates on the printing cylinder).
Preview	A low resolution bitmap file representing the content to be printed.
Sheet ? Deprecated in JDF 1.1	Specific information about the **Media** (including type and color) and about the **Sheet** (placement coordinates on the printing cylinder). Replaced by **Layout** in JDF 1.1.
TransferCurvePool ?	Function to apply **ContactCopying**, **DigitalPrinting**, and **ConventionalPrinting** process characteristics such as press, climate, and substrate under certain standardized circumstances. This function can be used to generate an accurate **InkZoneProfile**.

Output Resources

Name	Description
InkZoneProfile	Contains information about ink coverage along and perpendicular to the ink zones for a specific press geometry.

6.3.13 Interpreting

The interpreting device consumes page descriptions and instructions for controlling the printing device. The parsing of graphical content in the page descriptions produces a canonical display list of the elements to be drawn on each page.

The interpreter may encounter, and must act upon, device control instructions that affect the physical functioning of the printing device, such as media selection and page delivery. **Media** selection determines which type of medium is used for printing and where that medium can be obtained. Page delivery controls the location, orientation, and quantity of physical output.

The interpreter is also responsible for resolving all system resource references. This includes handling font substitutions and dealing with resource aliases. However, the interpreter specifically does not get involved with any functions of the device that could be considered finishing features, such as stapling, duplexing, and collating.

Input Resources

Name	Description
ColorantControl ? Modified in JDF 1.1	Identifies the color model used by the job.
FontPolicy ?	Describes the behavior of the font machinery in absence of requested fonts.
InterpretingParams	Provides the parameters needed to interpret the PDL pages specified in the **RunList** resource.
PDLResourceAlias *	These resources allow a JDF to reference resources which are defined in a Page Description Language (PDL). For example, a **PDLResourceAlias** resource could refer to a font embedded in a PostScript file.
RunList	This resource identifies a set of PDL pages or surfaces which will be interpreted.

Output Resources

Name	Description
InterpretedPDLData	Pipe of streamed data which represents the results of *Interpreting* the pages in the **RunList**. The format and detail of these data is implementation specific. In particular, it is assumed that the *Interpreting* and *Rendering* processes are tightly coupled and that there is no value in attempting to develop a general specification for the format of this data.

6.3.14 LayoutElementProduction

This process describes the creation of page elements. It also explains how to create a layout that can put together all of the necessary page elements, including text, bitmap images, vector graphics, PDL, or application files such as Adobe InDesign®, Adobe PageMaker®, and Quark XPress®. The elements might be produced using any of a number of various software tools. This process is often performed several times in a row before the final **LayoutElement**, representing a final layout file, is produced.

Input Resources

Name	Description
LayoutElement *	URL of the PDL or application file, bitmap image file, text file, vector graphics file, etc. Additional information (e.g., the page number or X, Y-coordinates) might be stored in the Comment element of the **LayoutElement** resource. Customer information such as the file templates, manuscripts, and sketches are handled via URL.

Output Resources

Name	Description
LayoutElement ?	A URL of the PDL or application file is produced by this process if no **RunList** is produced. Additional information, e.g., page number or X, Y-coordinates, might be stored in the Comment of the **LayoutElement**.
RunList ?	A **RunList** of **LayoutElement** resources of *ElementType Page* or *Document* is produced if this **LayoutElementProduction** task is the last process of type **LayoutElementProduction**.

6.3.15 LayoutPreparation

New in JDF 1.1

The **LayoutPreparation** process specifies the process of defining the **Layout** resource for the Imposition process. Note that it is possible to create a **Combined** process that includes both **LayoutPreparation** and **Imposition.** In this case, the **Layout** and **RunList (Marks)** resource would not be explicitly defined, since they are exchange resources between the two processes.

Input Resources

Name	Description
LayoutPreparationParams	Set of parameters required to control the **LayoutPreparation** process.
RunList (Document)	List of documents and/or pages that will be input into the layout. Note that this Runlist is for information only and not modified by the **LayoutPreparation** process.
RunList ? (Marks)	List of marks that will be input into the layout. These are typically printer's marks such as fold marks, cut marks, punch marks, or color bars.

Output Resources

Name	Description
Layout	The layout of the document to be imposed.
RunList (Marks) ?	List of marks that may be used as input of the following **Imposition** process.
TransferCurvePool ?	Definition of the transfer curves and coordinate systems of the devices.

6.3.16 PDFToPSConversion

The **PDFToPSConversion** process controls the generation of PostScript from a single PDF document. This process may be used at any time in a host-based PDF workflow to exit to PostScript for use of tools that consume such data. Additionally, it may be used to actively control the physical printing of data to a device that consumes PostScript data. The JDF model of this may include a **PDFToPSConversion** process in a *Combined* node with a **PSToPDFConversion** process.

Input Resources

Name	Description
PDFToPSConversionParams	Set of parameters required to control the generation of PostScript.
RunList	List of documents and pages to be converted to PostScript.

Output Resources

Name	Description
RunList	Stream or streams of resulting PostScript code. This PostScript code may end up physically stored in a file or be piped to another process. The *GeneratePageStreams* attribute of the **PDFToPSConversionParams** resource determines whether there is a single stream generated for all pages in the **RunList** or whether each page is generated in to a separate consecutive stream.

6.3.17 Preflight

Preflighting is the process of examining the components of a print job to ensure that the job will print successfully and with the expected results. *Preflight* checks may be performed on each PDL document identified within the associated **RunList** resource.

Preflighting a file is generally a three-step process. First, the pages are inventoried against a preflight profile, detailing the expected or hoped-for results. The resulting inventory identifies the significant characteristics of all the pages in the job. Next, the characteristics are tested against a set of criteria specified by a series of preflight constraint resources. Finally, results and discrepancies are reported in a **PreflightAnalysis** hierarchy log as analysis.

Agents record the instructions for, and devices record the results of, preflight operations in JDF jobs, using hierarchies headed by three types of resources: Inventory, Profile, and Results. The Inventory hierarchy may be used to record all the information gathered in the first step, although devices need not record this information. The Profile hierarchy is used to record the criteria used to test the file in the second step. And the Results hierarchy is used to record the results of the tests. In all three hierarchies, information is grouped into categories. There are six predefined categories in JDF—Colors, Document, Fonts, FileType, Images and Pages, but applications may define other categories if needed.

In a profile hierarchy, each category is populated with PreflightConstraint elements. Each PreflightConstraint element specifies a test that the application will perform when analyzing the file. In the Inventory and Results hierarchies, each category is populated with two kinds of subelements that record information about specific characteristics of the file: PreflightInstance and PreflightDetail. Such information is recorded in the following two ways:

1. Information that is specific to one instance of some file object is recorded via PreflightInstance subelements that occur in each of the results pools such as FontResultsPool and ImageResultsPool). Within each PreflightInstance element, PreflightInstanceDetail subelements provide detailed information about that instance. For example, to record information about each font used in the file, the FontResultsPool contains one PreflightInstance subelement, which groups a set of PreflightInstanceDetail subelements. Each of these subelements records one specific characteristic of the font.

2. Information that applies to the file as a whole is recorded via PreflightDetail subelements, which occur in the various results pools. For example, to record all the page sizes used in the file, the PagesResultsPool would contain several PreflightDetail subelements, one for each page size used in the file.

An Inventory hierarchy may be used to record all information about a file. Preflight tools are not required to create an Inventory hierarchy as part of the preflight information they record. However, tools may find it useful to record this information, allowing them to avoid reparsing the entire file in order to perform a new Analysis.

Profile hierarchies specify the constraints against which the file is tested. Each Analysis hierarchy reflects the results of evaluating the file characteristics, which may be recorded in an Inventory hierarchy, against a set of tests recorded in a Profile hierarchy.

PreflightConstraint elements record the specific details for the constraints specified in the **PreflightProfile** resource. PreflightDetail and PreflightInstanceDetail elements record results, while PreflightInstance elements group PreflightInstanceDetail subelements for instances of file objects. The details recorded are PDL-specific.

Applications can define constraints within any of the defined constraint categories for any file type. In addition, applications may add to the set of defined constraints and constraint categories, defining both the new category and the constraint within the category.

Whether constraints are specified for predefined or new constraint categories, the eventual values for those constraints are always expressed as PreflightConstraint elements which are part of a **PreflightProfile**. Furthermore, the results are always expressed as either PreflightDetail elements or PreflightInstance elements, which group PreflightInstanceDetail subelements for Analysis results.

Note that the resources for Preflight are under development and subject to major changes in a future release of this specification.

Input Resources

Name	Description
PreflightInventory ?	Provides an exhaustive list of all items already resolved in a previous preflight.
PreflightProfile	A specified list of constraints against which pages may be tested.
RunList	The list of pages to be preflighted.

Output Resources

Name	Description
PreflightAnalysis ?	Describes the results of a preflight operation. Provides analytical information for the constraints against which the file was tested.
PreflightInventory ?	Provides an exhaustive list of all items considered in preflight.
RunList ?	A list of pages that may or may not have been modified as a result of a fix-up operation.

6.3.18 PreviewGeneration

The *PreviewGeneration* process produces a low resolution **Preview** of each separation that will be printed. The **Preview** can be used in later processes such as *InkZoneCalculation*. The *PreviewGeneration* process typically takes place after *Imposition* or *RIPping*.

The **PreviewGeneration** can be performed in one of the following two ways: 1.) the imaged printing plate is scanned by a conventional plate scanner or 2.) medium to high resolution digital data are used to generate the **Preview** for the separation(s). The extent of the PDL coordinate system (as specified by the *MediaBox* attribute, the resolution of the preview image, and width and height of the image) must fulfill the following requirements:

$$MediaBox \text{ length} / 72 * \text{x-resolution} = \text{width} \pm 1$$
$$MediaBox \text{ height} / 72 * \text{y-resolution} = \text{height} \pm 1$$

A gray value of 0 represents full ink, while a value of 255 represents no ink (see the DeviceGray color model in chapter 4.8.2. of the *PostScript Language Reference Manual*).

Rules for the Generation of the Preview Image

To be useful for the ink consumption calculation, the preview data must be generated with an appropriate resolution. This means not only spatial resolution, but also color or tonal resolution. Spatial resolution is important for thin lines, while tonal resolution becomes important with large areas filled with a certain tonal value. The maximum error caused by limited spatial and tonal resolution should be less than 1 %.

Spatial Resolution

Since some pixel of the preview image might fall on the border between two zones, their tonal values must be split up. In a worst case scenario, the pixels fall just in the middle between a totally white and a totally black zone. In this case, the tonal value is 50%, but only 25% contributes to the black zone. With the resolution of the preview image and the zone width as variables, the maximum error can be calculated using the following equation:

$$error[\%] = \frac{100}{4 * resolution[L/mm] * zone\_width[mm]}$$

For zone width broader than 25 mm, a resolution of 2 lines per mm will always result in an error less than 0.5 %. Therefore, a resolution of 2 lines per mm (equal to 50.8 dpi) is suggested.
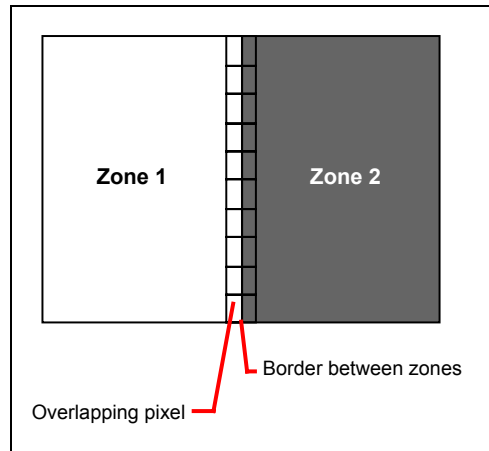
Figure 6.1 Worst case scenario for area coverage calculation

Tonal Resolution
The kind of error caused by color quantization depends on the number of shades available. If the real tonal value is rounded to the closest (lower or higher) available shade, the error can be calculated using the following equation:

$$error[\%] = \frac{100}{2*number\_of\_shades}$$

Therefore, at least 64 shades should be used.

Line Art Resolution
When rasterizing line art elements, the minimal line width is 1 pixel, which means 1/resolution. Therefore, the relationship between the printing resolution and the (spatial) resolution of the preview image is important for these kind of elements. In addition, a specific characteristic of PostScript RIPs adds another error: within PostScript, each pixel that is touched by a line is set. Tests with different PostScript jobs have shown that a line art resolution of more than 300 dpi is normally sufficient for ink-consumption calculation.

Conclusion
There are quite a few different ways to meet the requirements listed above. The following list includes several examples:

- The job can be Ripped with 406.4 dpi monochrome.

- With anti-aliasing, the image data can be filtered down by a factor of 8 in both directions. This results in an image of 50.8 dpi with 65 color shades.

- High resolution data can also be filtered using anti-aliasing. First, the Ripped data, at 2540 dpi monochrome, is taken and filtered down by a factor of 50 in both directions. This produces an image of 50.8 dpi with 2501 color shades. Finally those shades are mapped to 256 shades, without affecting the spatial resolution.

Rasterizing a job with 50.8 dpi and 256 shades of gray is not sufficient. The problem in this case is the rendering of thin lines (see Line Art Resolution).

Recommendations for Implementation
The following three guidelines are strongly recommended:
- The resolution of RIPped line art must be at least 300 dpi.
- The spatial resolution of the preview image must be approximately 20 pixel/cm (= 50.8 dpi).
- The tonal resolution of the preview image must be at least 64 shades.

Input Resources

Name	Description
ColorantControl ? New in JDF 1.1	The **ColorantControl** resources that define the ordering and usage of inks in print modules. Needed for generating thumbnails.
ExposedMedia ?	The *PreviewGeneration* process can use an exposed printing plate to produce a **Preview** resource. This task is performed using an analog plate-scanner. Only one of **ExposedMedia, Preview,** or **RunList** may be specified in any *PreviewGeneration* process.
Preview ? New in JDF 1.1	Medium or low resolution bitmap file that can be used for calculation of overviews and thumbnails. Only one of **ExposedMedia, Preview,** or **RunList** may be specified in any *PreviewGeneration* process.
PreviewGenerationParams	Parameters specifying the size and the type of the preview.
RunList ?	High resolution bitmap data is consumed by the *PreviewGeneration* process. These data represent the content of a separation that is recorded on a printing plate or other such item. Only one of **ExposedMedia, Preview,** or **RunList** may be specified in any *PreviewGeneration* process.
TransferCurvePool? New in JDF 1.1	Area coverage correction and coordinate transformations of the device.

Output Resources

Name	Description
Preview	The **Preview** data are comprised of low resolution bitmap files representing, for example, the content of a separation that is recorded on a printing plate or other such item.

6.3.19 Proofing

The *Proofing* process results in the creation of a physical proof, represented by an **ExposedMedia** resource. Proofs can be used to check an imposition or the expected colors for a job. The inputs of this process are a **RunList**, which identifies the pages to proof; the **ProofingParams** resource, which describes the type of proof to be created; and a **Media** resource to describe the physical media that will be used.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies the color model used by the job.
ColorSpaceConversionParams ?	This resource provides information needed to convert colorspaces in the pages for proofing. Generally present if a color proof is desired, unless the pages in the **RunList** have already been operated on by a previous colorspace conversion process.
Layout ?	Required if an imposition proof is desired.
Media	This resource characterizes the output media for the proof.
ProofingParams	This resource provides the parameters needed to produce the desired proof.
RunList (Document)	Identifies the pages to be proofed. When the **Layout** resource is present in the **ProofingParams** resource, *Ord* values from ContentObject subelements refer to pages in this **RunList**.

Name	Description
RunList ? (Marks)	Structured list of incoming marks. These are typically printers marks, e.g., fold, cut or punch marks, or color bars.
	When the **Layout** resource is present in the **ProofingParams** resource, *Ord* values from MarkObject subelements refer to pages in this **RunList**.

Output Resources

Name	Description
ExposedMedia	The resulting physical proof.

6.3.20 PSToPDFConversion

This section defines the controls required to invoke a device that accepts a PostScript stream and produces a set of PDF pages as output.

Input Resources

Name	Description
FontParams ?	These parameters determine how the conversion process will handle font errors encountered in the PostScript stream.
ImageCompressionParams ?	This resource provides a set of controls that determines how images will be compressed in the resulting PDF pages.
PSToPDFConversionParams ?	These parameters control the operation of the process that interprets the PostScript stream and produces the resulting PDF pages.
RunList	This resource specifies where the PostScript stream is to be found.

Output Resources

Name	Description
RunList	This resource identifies the location of the resulting PDF pages.

6.3.21 Rendering

The *Rendering* process consumes the display list of graphical elements generated by an interpreter. It color manages and scans/converts the graphical elements according to the geometric and graphic state information contained within the display list. The controls governing the external rendering processes provide overrides and additional parameters for controlling the behavior of the process.

Input Resources

Name	Description
Media Deprecated in JDF 1.1	This resource provides a description of the physical media which will be marked. The physical characteristics of the media may affect decisions made during *Rendering*.
InterpretedPDLData	Pipe of streamed data that represents the results of *Interpreting* the pages in the **RunList**. The format and detail of these data is implementation specific. In particular, it is assumed that the *Interpreting* and *Rendering* processes are tightly coupled and that there is no value in attempting to develop a general specification for the format of this data.
RenderingParams ?	This resource describes the format of the **ByteMaps** to be created and other specifics of the *Rendering* process.

Output Resources

Name	Description
RunList	Ordered list of rasterized **ByteMaps** representing pages

6.3.22 RIP'ing

RIP'ing is, in the context of a workflow, a *Combined* process that is an amalgamation of at least two processes. Most often it includes *Interpreting* and *Rendering*, but it may also include *SoftProofing*, *Trapping, Separation, Imposition*, and *Screening*. Thus a typical RIP node is of *Type Combined*, as shown in the following example:

```
<JDF Type="Combined" Types="Interpreting Rendering Screening" … />
```

The *RIPping* process consumes page descriptions and instructions for producing the graphical output. It parses the graphical contents in the page descriptions, renders the contents, and produces a rasterized image of the page. This raster may contain contone data and be represented upon output as a **ByteMap**. Alternatively, the *RIPping* process may also perform halftone screening, in which case the output is in the form of a bitmap. It is also responsible for resolving all system resource references that include font handling and resource aliasing.

Instructions read by the RIP include information about the media, halftoning, color transformations, colorant controls and other items that affect that rasterized output. They do not, however, represent any specific controls for the physical output device, nor do they deal with any instructions intended for the finishing device.

When a *RIPping* process is comprised of only the *Interpreting* and *Rendering* processes, various intermediary steps are required before the output can be run through a *ConventionalPrinting* process. In theory, however, a workflow could include no intermediary steps between a *RIPping* process and a *DigitalPrinting* process. The following workflow scenarios represent possible process chains in each circumstance:

- RIP→Screening→ImageSetting→ContactCopying→ConventionalPrinting
- RIP→(Screening)→DigitalPrinting

Since RIP'ing never stands alone as a process, see the processes that contribute to the RIP for input and output resources.

6.3.23 Scanning

The *Scanning* process creates bitmaps from analog images using a scanner.

Input Resources

Name	Description
ExposedMedia	Description of the media to be scanned. The **ExposedMedia** should be partitioned by *RunIndex*, in order to provide unique mapping from ExposedMedia to the output **RunList**.
ScanParams	High level scanner settings. These settings are specifically not intended as a replacement for low-level device interfaces such as TWAIN.

Output Resources

Name	Description
RunList	List of **ByteMap** resources or **LayoutElement** resources of *Type = Image*.

6.3.24 Screening

This process specifies the process of halftone screening. It consumes contone raster data, e.g., the output from an interpreting and rendering process. It produces monochrome which has been filtered through a halftone screen to identify which pixels are required to approximate the original shades of color in the document.

This process definition includes capabilities for post-RIP halftoning according to the PostScript definitions. Alternatively it allows for the selection of FM screening/error diffusion techniques. However, in these circumstances no

specific parameter sets are defined. In general, an actual screening process will be a **Combined** process of **Cali-bration** and **Screening**.

Input Resources

Name	Description
RunList	Ordered list of rasterized **ByteMaps** representing pages or surfaces.
ScreeningParams	Parameters specifying which halftone mechanism is to be applied and with what specific controls.

Output Resources

Name	Description
RunList	Ordered list of rasterized and screened output pages. Assumes that the resolution remains the same and that resulting data is one bit per component. Furthermore, the organization of planes within the data does not change.

6.3.25 Separation

The **Separation** process specifies the controls associated with the generation of color-separated data. It is designed to be flexible enough to allow a variety of possible methods for accomplishing this task. First of all, it sponsors host-based PDF separating operations, in which a **RunList** of preseparated PDF data is generated. It can also be combined with a RIP to allow control of In-RIP separations. In this scenario a **RunList** containing **ByteMaps** is generated as the output. Yet another anticipated combination is with the *ColorCorrection* process to deal with in-coming device-dependent data. And finally, it may be combined with an *ImageReplacement* process in order to do image substitution for omitted or proxy images.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies which colorants in the job are to be output.
RunList	List of pages that are to be operated on.
SeparationControlParams	Controls for the separation process.

Output Resources

Name	Description
RunList	List of separated pages or separated raster bytemaps.

6.3.26 SoftProofing

SoftProofing is the process of reviewing final-form output on a monitor rather than in paper form. The inputs are a **RunList**, which identifies the pages to proof; the **ProofingParams** resource, which describes the type of proof to be created.

Within the **ProofingParams** resource, the proof device parameter specifies the characterization the monitor on which the proof will be viewed. This processor must create and perform a transformation from the final target device to the proof device colors before displaying the document contents.

The soft proofing parameters allow sufficient control to determine whether any images are displayed in the proof. If so, the ability to select low resolution proxies or full resolution images is provided. The mechanism for approving proofs requires the generation of a PDF file containing the proofing parameters and a digital signature noting the acceptance of them. The approval PDF file need not contain any graphical data.

Like all other color manipulation supported in JDF, the color conversion controls are based on the use of ICC profiles. While the assumed characterization of input data can take many forms, each can internally be represented as an ICC Profile. In order to perform the transformations, input profiles must be paired with the identified final target device profile to create the transformation.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies the color model used by the job.
ColorSpaceConversionParams ?	This resource provides information needed to convert colorspaces in the pages for proofing. Generally present if a color proof is desired, unless the pages in the **RunList** have already been operated on by a previous colorspace conversion process.
Layout ?	Required if an imposition proof is desired.
ProofingParams	Provides the parameters needed to produce the desired proof.
RunList (Document)	Identifies the pages to be proofed. When the **Layout** resource is present in the **ProofingParams** resource, *Ord* values from ContentObject subelements refer to pages in this **RunList**.
RunList ? (Marks)	Structured list of incoming marks. These are typically printer's marks, e.g., fold marks, cut marks, punch marks, or color bars. When the **Layout** resource is present in the **ProofingParams** resource, *Ord* values from MarkObject subelements refer to pages in this **RunList**.

Output Resources

None. The *SoftProofing* process is always combined with an *Approval* process.

6.3.27 Tiling

The *Tiling* process allows the contents of **Surfaces** to be imaged onto separate pieces of media. Note that many different workflows are possible. *Tiling* must always follow *Imposition*, but it can operate on imposed PDL page contents or on contone or halftone data. *Tiling* will generally be combined with other processes. For example, *Tiling* might be combined with *ImageSetting*. In that case, the input would be a **RunList** that contains **ByteMaps** for each **Surface**.

Input Resources

Name	Description
RunList (Surface)	Structured list of imposed page contents or **ByteMaps** that are to be decomposed to produce the images for each tile. The *Type* value of **LayoutElement** resources must all be *Surface*.
RunList ? (Marks)	Structured list of incoming marks. These are typically printer's marks that provide the information needed to combine the tiles.
Tile	A partitioned **Tile** resource that describes how the **Surface** contents are to be decomposed.

Output Resources

Name	Description
RunList	Structured list of portions of the decomposed surfaces. The value of the *Type* attribute of the **LayoutElement** resources must be *Tile*.

6.3.28 Trapping

Trapping is a prepress process that modifies PDL files to compensate for a type of error that occurs on presses. Specifically, when more than one colorant is applied to a piece of media using more than one inking station, the media may not stay in perfect alignment when moving between inking stations. Any misalignment will result in an error called misregistration. The visual effect of this error is either that inks are erroneously layered on top of one another, or, more seriously, that gaps occur between inks that should abut. In this second case, the color of the me-

dia is revealed in the gap and is frequently quite noticeable. **Trapping**, in short, is the process of modifying PDL files so that abutting colorant edges intentionally overlap slightly, in order to reduce the risk of gaps.

The **Trapping** process specifies that a set of document pages should be modified to reduce or (ideally) eliminate visible misregistration errors in the final printed output. The process may be combined with **RIPping** or specified as a stand-alone process.

Input Resources

Name	Description
ColorantControl ? Modified in JDF1.1A	Identifies color model used by the job.
FontPolicy ? New in JDF 1.1	Describes the behavior of the font machinery in absence of requested fonts.
RunList	Structured list of incoming page contents that are to be trapped.
TrappingDetails	Describes the general setting needed to perform trapping.

Output Resources

Name	Description
RunList	Structured list of the modified page contents after **Trapping** has been executed.

6.4 Press Processes

Press processes are various technological procedures involving the transfer of ink to a substrate. From a technical standpoint they are often classified in impact and non-impact printing technologies. The impact printing class can be further subdivided into relief, intaglio, planograph, or screen technologies, which in turn can be divided in further subparts. Because of the way a workflow is constructed in JDF, however, a different approach to classification was used. All of the various printing technologies are gathered into three categories: 1.) **ConventionalPrinting**, which involves printing from a physical master, 2.) **DigitalPrinting**, which involves generic commercial printing from a digital master. A third process, 3.) **IDPrinting**, which stands for integrated digital printing and involves simple digital printing as specified in the IPP protocol was defined in JDF 1.0 but is deprecated in JDF 1.1. A Combined process including **DigitalPrinting** should be implemented instead.

The most prominent physical, planographic printing technologies are offset lithography and electrophotography. They are also the printing processes with the highest adoption in today's graphic arts industry. Consequently, the **ConventionalPrinting** process in JDF takes them as models. That does not mean, however, that other printing techniques can not make use of the **ConventionalPrinting** process and its resources. The extensibility features of JDF may be used to fill other requirements related to printing technology.

6.4.1 ConventionalPrinting

This process covers several conventional printing tasks, including sheetfed printing, web printing, web/ribbon coating, converting, and varnishing. Typically, each takes place after prepress and before postpress processes. Press machinery often includes postpress processes, e.g., **Folding**, **Numbering**, and **Cutting**, as in-line finishing operations. The **ConventionalPrinting** process itself does not cover these postpress tasks. Using a conventional printing press for producing a pressproof can be performed in the following two ways:

- A proof of type **Component** is produced with a **ConventionalPrinting** process. The result of this process is then sent to the **Approval** process, which in turn produces an **ApprovalSuccess** resource. That resource is then passed on to a second **ConventionalPrinting** process, which requires that the press be set up a second time.

- The *DirectProof* attribute of the **ConventionalPrintingParams** can be used to specify the proof if it is produced during the **ConventionalPrinting** process. In this case, the press need only be set up once.

Note, the definition and ordering of separations is specified by the *DeviceColorantOrder* attribute of the appropriate **ColorantControl** resource.

Input Resources

Name	Description
ColorantControl ?	The **ColorantControl** resources that define the ordering and usage of inks in print modules.
Component ? (Input)	Various components in the form of preprints can be used in *ConventionalPrinting* in lieu of **Media**. Examples include waste or a set of preprinted sheets.
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier *ConventionalPrinting* process.
ConventionalPrintingParams	Specific parameters to set up the press.
ExposedMedia ? (Proof)	A Proof is used to compare color and content during *ConventionalPrinting*. This Proof is produced by a prepress proofing device.
ExposedMedia (Plate)	The printing plate and information about it (such as *Thickness* and *RegisterPunch*) is used to set up the press.
Ink ? Modified in JDF 1.1	Information (brand, type, clone) about the ink is useful to set up the press.
InkZoneProfile ?	The **InkZoneProfile** contains information about how much ink is needed along the printing cylinder of a specific printing press. It is only useful for Offset Lithography presses with ink key adjustment functions.
Layout ? New in JDF 1.1	Sheet and Surface elements from the **Layout** tree such as **CIELABMeasuringField**, **DensityMeasuringField**, or **ColorControlStrip** can be used for quality control at the press. The quality control field value and position can be of interest for automatic quality control systems. **RegisterMark** can be used to line up the printing plates for the press run, and its position can in turn be used to position items such as a camera.
Media ?	The physical substrate, e.g., paper or foil, and information about the **Media**, e.g., such as thickness, type, and size, are useful in setting up paper travel in the press. This resource must be present if no preprinted **Component** (Input) resource is used.
Sheet ? Deprecated in JDF 1.1	Specific information about the **Media** (including type and color) and about the **Sheet** (placement coordinates on the printing cylinder). Replaced by **Layout** in JDF 1.1.
TransferCurvePool? New in JDF 1.1	Area coverage correction and coordinate transformations of the device.

Output Resources

Name	Description
Component (Good)	Describes the printed sheets or ribbons which may be used by another printing process or postpress processes. Note that the *Amount* attribute of the ResourceLink to this resource indicates the number of copies of the entire job which will be produced.
Component ? (Waste)	Produced waste of printed sheets or ribbons.

6.4.2 DigitalPrinting

DigitalPrinting is a direct printing process that, like *ConventionalPrinting*, occurs after prepress processes but before postpress processes. In *DigitalPrinting*, the data to be printed are not stored on an extra medium (such as a printing plate or a printing foil), but instead are stored digitally. The printed image is generated for every output using the digital data. Electrophotography, inkjet, and other technologies are used for transferring ink (both liquid ink and dry toner) onto the substrate. Furthermore, both sheet and web presses can be used as machinery for *DigitalPrinting*.

DigitalPrinting is often used to image a small area on preprinted **Component**s to perform actions such as addressing or numbering another **Component**. This kind of process can be executed by imaging with an inkjet printer during press, postpress, or packaging operations. Therefore, *DigitalPrinting* is not only a press or prepress operation but sometimes also a postpress process.

Digital printing devices which provide some degree of finishing capabilities, such as collating and stapling, as well as some automated layout capabilities, such as N-up and duplex printing may be modeled as a combined process which includes *DigitalPrinting*. Such a combined process may also include other processes, e.g., *ContoneCalibration, Cutting, Folding, HoleMaking, Imposition, Interpreting, LayoutPreparation, Perforating, Rendering, Screening, Stacking, Stitching, Trapping, or Trimming*.

Controls for *DigitalPrinting* are provided in the **DigitalPrintingParams** resource. The set of input resources of a combined process which includes *DigitalPrinting* may be used to represent an Internet Printing Protocol (IPP) job or a PPML job. See Application Notes for IPP and Variable Data printing.

Note: Putting a label on a product or DropItem is not *DigitalPrinting* but *Inserting*.

Input Resources

Name	Description
ColorantControl ?	The **ColorantControl** resources that define the ordering and usage of inks in print modules.
Component * (Input)	Various components can be used in *DigitalPrinting* instead of **Media**. Examples include preprinted covers, waste, precut **Media**, or a set of preprinted sheets or webs. If multiple **Component** * (Input) resources are linked to one process, the mapping of media to content is defined in the partitions of **DigitalPrintingParams.**
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier *ConventionalPrinting* process (see description in Section 6.4.1).
DigitalPrintingParams	Specific parameters to set up the machinery.
ExposedMedia ?	A **Proof** is useful for comparisons (completeness, color accuracy) with the print out of the *DigitalPrinting* process.
Ink ?	Ink or toner and information that is needed for *DigitalPrinting*.
Layout ? New in JDF 1.1	Sheet and Surface elements from a Layout such as the **CIELABMeasuringField**, **DensityMeasuringField**, or **ColorControlStrip** can be used for quality control at the press. The value and position of the quality can be of interest for automatic quality control systems. **RegisterMark**s can be used to line up the printing registration during press run, and its position can in turn be used to position an item such as a camera.
Media *	The physical **Media** and information about the **Media**, such as thickness, type, and size, is used to set up paper travel in the press. This has to be present if no preprinted **Component** (input) resource is present. Unprinted **Media** used for covers are also defined as **Media**. Note: Printing a job on more than one web or sheet at the same time is parallel processing.
RunList	Rendered data in **ByteMap**s that will be printed on the digital press is needed for *DigitalPrinting*. The **RunList** contains only **ByteMap**s.
Sheet ? Deprecated in JDF 1.1	Specific information about the **Media** (including type and color) and about the **Sheet** (placement coordinates on the printing cylinder). Replaced by **Layout** in JDF 1.1.
TransferCurvePool? New in JDF 1.1	Area coverage correction and coordinate transformations of the device.

Output Resources

Name	Description
Component (Good)	Components are produced for other printing processes or postpress processes. Note that the *Amount* attribute of the ResourceLink to this resource indicates the number of copies of the entire job which will be produced.
Component ? (Waste)	Produced waste, may be used by other processes.

6.4.3   IDPrinting

Deprecated in JDF 1.1

IDPrinting, which stands for Integrated Digital Printing, is a specific form of digital printing. It combines functionality that might be represented by the *Interpreting*, *Rendering*, *Screening*, and *DigitalPrinting* processes in a single process. In addition, devices which support *IDPrinting* frequently provide some degree of finishing capabilities, such as collating and stapling, as well as some automated layout capabilities, such as N-up and duplex printing.

   Controls for *IDPrinting* are provided in the **IDPrintingParams** resource. These controls are intended to be somewhat limited in their scope. If greater control over various aspects of the printing process is required, *IDPrinting* should not be used. Ultimately, the controls specified for *IDPrinting* can be used to generate an Internet Printing Protocol (IPP) job. See JDF/1.0 Appendix F for a mapping between JDF IDPrinting and IPP. *IDPrinting* may be combined with other processes, such as *Trapping* or *ColorSpaceConversion*.

Input Resources

Name	Description
ColorantControl ?	The **ColorantControl** resources that define the ordering and usage of inks in print modules.
Component ? (Cover)	A finished cover may be combined with the pages that will be output by this process.
Component ? (Input)	Various components can be used in *IDPrinting* instead of **Media**. Examples include waste, precut **Media**, or a set of preprinted sheets or webs.
Component ? (Proof)	A Proof component is used if a proof was produced during an earlier *ConventionalPrinting* process.
ExposedMedia ?	A **Proof** is useful for comparisons (completeness, color accuracy) with the print out of the *IDPrinting* process.
FontPolicy ?	Describes the behavior of the font machinery in absence of requested fonts.
Ink ?	Ink or toner and information about it is needed for *IDPrinting*.
InterpretingParams *	A set of resources that specify how the device should interpret the PDL files which are referenced by the **RunList** for the process. Note that **InterpretingParams** is an abstract resource. Instances are PDL-specific.
IDPrintingParams ?	Specific parameters to set up the machinery.
Media ?	The physical **Media** and information about the **Media**, such as thickness, type, and size, are used to set up paper travel in the press. This has to be present if no preprinted **Component** (input) resource is present. Note: Printing a job on more than one web or sheet at the same time is parallel processing.
RenderingParams ?	This resource describes the format of the **ByteMaps** to be created.
RunList	The set of pages to be printed.
ScreeningParams ?	Parameters specifying which halftone mechanism is to be applied and with what specific controls.

Name	Description
TransferFunctionControl ?	Controls whether the device performs transfer functions and what values are used when doing so.

Output Resources

Name	Description
Component (Good)	Components are produced for other printing processes or postpress processes. Note that the *Amount* attribute of the ResourceLink to this resource indicates the number of copies which will be produced.
Component ? (Waste)	Produced waste, may be used by other processes.

6.5 Postpress Processes

In this specification, the postpress processes are presented in two parts: an alphabetical list of processes that is then followed by a Postpress Processes Structure section that divides these processes into subchapters for structuring purposes. This structuring is useful to find specific processes. Please note that processes, in some cases can be used to describe operations that go beyond the scope of a specific chapter. Therefore, it is a good idea not only to look at certain processes within a subchapter but also to find out what functionality other processes offer if a specific task needs to be addressed.

6.5.1 AdhesiveBinding

Deprecated in JDF 1.1

The *AdhesiveBinding* has been split into the following individual processes:

- *CoverApplication,*
- *Gluing*
- *SpinePreparation,*
- *SpineTaping.*

Note that the parameters of the *GlueApplication* ABOperations have been moved into **CoverApplicationParams** and **SpineTapingParams** as GlueApplication refelements. The generic *GlueApplication* ABOperation is now described by the *Gluing* process.

6.5.2 BlockPreparation

New in JDF 1.1

As there are many options for a hardcover book, the block preparation is more complex than what has already been described for other types of binding above. Those options are the ribbon band (numbers of bands, materials and colors), gauze (material and glue), headband (material and colors), kraft paper (material and glue), and tightbacking (different geometry and measurements).

Input Resources

Name	Description
Component	The **BlockPreparation** process consumes one **Component** and creates a book block.
BlockPreparationParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the prepared book block. Its *ProductType* = "*BookBlock*"

6.5.3 BoxPacking

New in JDF 1.1

A pile, stack or bundle of products can be packed into a box or cartoon.

Input Resources

Name	Description
Component	The *BoxPacking* process puts a set of Components into the box **Component**.
BoxPackingParams	Specific parameters to set up the machinery.
Component (Box) ?	Details of the box or carton.

Output Resources

Name	Description
Component	One **Component** is produced: the boxed **Component**.

6.5.4 CaseMaking

New in JDF 1.1

Case making is the process where a hard case is produced. As there are many different kinds of hardcover cases, they will be described in a later version of the JDF specification.

Input Resources

Name	Description
Component (CoverMaterial) ?	The cover material which may be either a preprinted and processed sheet of paper. If no **Component** is specified, a Media (**CoverMaterial**) must be specified.
CaseMakingParams	Specific parameters to set up the machinery.
Media (CoverMaterial)?	The *CaseMaking* process may also consume unprocessed **Media** as cover material. Only one of Media (**CoverMaterial**) or Component **(CoverMaterial)** must be specified.
Media (CoverBoard) Modified in JDF 1.1A	The cardboard Media used for the cover board.
Media (SpineBoard)?	The cardboard Media used for the spine board. If not specified, the same media as used for Media **(CoverBoard)** is used.

Output Resources

Name	Description
Component	One **Component** is produced: the produced book case. Its *ProductType* = "*BookCase*"

6.5.5 CasingIn

New in JDF 1.1

The hard cover book case and the book block are joined in the *CasingIn* process.

Input Resources

Name	Description
Component	The prepared book block.
Component (Case)	The hard cover book case.
CasingInParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the hard cover book.
Component	One **Component** is produced: the thread-sewn components forming an item such as a raw book.

6.5.6 ChannelBinding

Various sizes of metal clamps can be used in *ChannelBinding*. The process can be executed in two ways. In the first, a pile of single sheets—sometimes together with a front and back cover—is inserted into a U-shaped clamp and crimped in special machinery. In the second, a preassembled cover that includes the open U-shaped clamp is used instead of the U-shaped clamp alone. The thickness of the pile of sheets determines in both cases the width of the U-shaped clamp to be used for forming the fixed document, which is not meant to be reopened later.

Input Resources

Name	Description
Component (BookBlock)	The operation requires one component: the block of sheets to be bound.
Component ? (Cover)	The empty cover with the U-shaped clamp that might, for example, have been printed before it is used during the *ChannelBinding* process.
ChannelBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the channel-bound component forming an item such as a bro-chure.

6.5.7 CoilBinding

CoilBinding is a technique that creates bindings not meant to be reopened later. Another name for *CoilBinding* is *spiral binding*. Metal wire, wire with plastic, or pure plastic is used to fasten prepunched sheets of paper, card-board, or other such materials. First, automated machinery forms a spiral of proper diameter and length. The ends of the spiral are then "tucked-in". Finally, the content is permanently fixed. Note that every time a coil-bound book is opened, a vertical shift occurs as a result of the coil action. This is a characteristic of the process.

Input Resources

Name	Description
Component	The operation requires one component: the pile of prepunched sheets of-ten including a top and button cover.
CoilBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the coil-bound component forming an item such as a calendar.

6.5.8 Collecting

This process collects folded sheets or partial products, some of which may have been cut. The first **Component** to enter the workflow lies at the bottom of the pile col-lected on a saddle, and the sequence of the input components that follows depends upon the produced component. The figure to the right shows a typical collected pile.

The operation coordinate system is defined as follows: The y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge, i.e., the product front edge.

Input Resources

Name	Description
CollectingParams ?	Specific parameters to set up the machinery.
Component +	Variable amount of sheets to be collected.
DBRules *	Database input that describes which sheets should be collected for a particular instance component. In this version the schema is only human readable text. One rule is applied for each individual component.
DBSelection ?	Database input that describes which sheets should be collected for a particular instance component.
IdentificationField ?	Information about identification marks on the component.

Output Resources

Name	Description
Component	A block of collected sheets is produced. This **Component** can be joined in further postpress processes.

6.5.9 CoverApplication

New in JDF 1.1

CoverApplication describes the process of applying a soft cover to a book block.

Input Resources

Name	Description
CoverApplicationParams	Specific parameters to set up the machinery.
Component	The book block on which the cover is applied
Component (Cover)	The soft cover that is applied.

Output Resources

Name	Description
Component	The book block with the applied soft cover.

6.5.10 Creasing

New in JDF 1.1

Sheets are creased or grooved to enable folding or to create even, finished page delimiters.

Input Resources

Name	Description
Component ?	This process consumes one **Component**: the printed sheets.
CreasingParams	Details of the ***Creasing*** process.

Output Resources

Name	Description
Component	One creased **Component** is produced.

6.5.11 Cutting

Sheets are cut using a guillotine *Cutting* machine. Before *Cutting*, the sheets might be jogged and buffered.
CutBlocks and or **CutMark**s can be used for positioning the knife. After the *Cutting* process is performed, the blocks are often again buffered on a pallet.

Since *Cutting* is described here in a way that is machine independent as much as possible, the **CutBlock** elements specified do not directly imply a certain cutting sequence. Therefore, a sequence must be determined by a specialized agent.

Input Resources

Name	Description
Component ?	This process consumes one **Component**: the printed sheets.
CutBlock * Deprecated in JDF 1.1	One or several **CutBlock**s can be used to find the *Cutting* sequence. Only one of **CutBlock** or **Cut** may be specified.
CutMark * Deprecated in JDF 1.1	**CutMark** resources can be used to adapt the theoretical cut positions to the real positions of the corresponding blocks on the **Component** to be cut.
CuttingParams New in JDF 1.1	Details of the *Cutting* process.
Media ?	Cutting can be performed to unprinted **Media** in order to adjust size or shape.

Output Resources

Name	Description
Component +	One or several blocks of cut components are produced. When **Media** are cut, the output **Component**s can be input resources for processes such as *ConventionalPrinting*.

6.5.12 Dividing

Deprecated in JDF 1.1.

Dividing has been replaced by *Cutting.* In-line finishing of web presses often includes equipment for cutting the ribbon(s) in cross direction. This operation can be described with the *Dividing* process. *Dividing* in cross direction is likely to happen after former folding, which is a *LongitudinalRibbonOperations* process. It may affect one or more ribbons at the same time that are all part of one **Component**.

Input Resources

Name	Description
Component	The *Dividing* process consumes one **Component**: the web(s) or ribbon(s) entering the crosscutting machinery. The substrate might have been treated with *LongitudinalRibbonOperations* and may be folded with a former fold.
DividingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: either the divided web or ribbon.

6.5.13 Embossing

New in JDF 1.1

The *Embossing* process is performed after printing to stamp a raised or depressed image (artwork or typography) into the surface of paper, using engraved metal embossing dies, extreme pressure, and heat. Embossing styles include blind, deboss, and foil-embossed.

Input Resources

Name	Description
Component	This process consumes one **Component**:
EmbossingParams	Parameters to setup the machinery.
Media ?	If foil stamping or foil embossing, the stamping foil material is required.
Tool ?	The embossing stamp or calendar.

Output Resources

Name	Description
Component	One **Component** is created.

6.5.14 EndSheetGluing

EndSheetGluing finalizes the folded **Sheet** or book block in preparation for case binding. It requires three **Component**s—the back-end sheet, the book block, and the front-end sheet—and information about how they are merged together. Back-end sheets and front-end sheets are in most cases sheets folded once before *EndSheetGluing* takes place. The end sheets serve as connections between the book block and the cover boards.

Input Resources

Name	Description
Component (BackEndSheet)	A back-end sheet to be mounted on the book block.
Component (BookBlock)	A back-end sheet and a front-end sheet are glued onto the book block.
Component (FrontEndSheet)	A front-end sheet to be mounted on the book block.
EndSheetGluingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	A book block is produced that includes the end sheets.

6.5.15 Folding

Buckle folders or knife folders are used for *Folding* sheets. One or more sheets can be folded at the same time. Web presses often provide in-line *Folding* equipment. Longitudinal *Folding* is often performed using a former, a plow folder, or a belt, while jaw folding, chopper folding, or drum folding equipment is used for folding the sheets that have been divided.

The JDF *Folding* process covers both operations done in stand-alone *Folding* machinery—typically found for processing sheet fed printed materials—and in-line equipment of web printing presses. Creasing and/or slot perforating are sometimes necessary parts of the *Folding* operation that guarantee exact process execution. They depend on the folder used, the **Media**, and the folding layout. These operations are specified in the *Creasing* and *Perforating* processes respectively.

Input Resources

Name	Description
Component	**Component**s, including a printed sheet or a pile of sheets, are used in the *Folding* process.
FoldingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component Modified in JDF 1.1	The process produces a **Component**, which in most cases is a folded **Sheet.**

6.5.16 Gathering

In the *Gathering* process, ribbons, sheets, or other **Components** are accumulated on a pile that will eventually be stitched or glued in some way to create an individual **Component**. The input **Components** may be output resources of a web-printing machine used in *Collecting* or of any machine that executes a *ConventionalPrinting* or *DigitalPrinting* process. In sheet applications, a moving gathering channel is used to transport the pile. But no matter what the inception of the *Gathering* process, the sequence of the input components dictates the produced component. The figure on the right shows typical gathered piles.
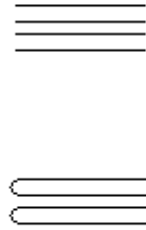
Input Resources

Name	Description
Component +	Variable amount of components including single sheets or folded sheets are used in the *Gathering* process.
GatheringParams	Specific parameters to set up the machinery.
DBRules *	Database input that describes which sheets should be gathered for a particular instance component. The schema are only in the form of human-readable text. One rule is applied for each individual component.
DBSelection ?	Database input that describes which sheets should be gathered for a particular instance component.
IdentificationField ?	Information about identification marks on the component.

Output Resources

Name	Description
Component	Components gathered together, such as a pile of folded sheets.

6.5.17 Gluing

New in JDF 1.1

Gluing describes arbitrary methods of applying glue to a **Component**.

Input Resources

Name	Description
Component	This process consumes one **Component**: the printed sheets.
GluingParams	Details of the *Gluing* process.

Output Resources

Name	Description
Component	One **Component** is produced.

6.5.18 HeadBandApplication

New in JDF 1.1

Head bands are applied to the hard cover book block.

Input Resources

Name	Description
Component	The prepared book block.
HeadBandApplicationParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the hard cover block with head bands.

6.5.19 HoleMaking

A variety of machines, such as those responsible for stamping and drilling, can perform the *HoleMaking* process. This postpress process is needed for different binding techniques, such as spiral binding. One or several holes with different shapes can be made that are later on used for binding the book block together.

Input Resources

Name	Description
Component	One **Component,** such as a printed sheet or a pile of sheets, are modified in the *HoleMaking* process.
HoleMakingParams	Specific parameters, including hole diameter, and positions, used to set up the machinery.

Output Resources

Name	Description
Component	A **Component** with holes, such as a book block or a single sheet, is produced for further postpress processes.

6.5.20 Inserting

This process can be performed at several stages in postpress. The process can be used to describe the labeling of products, of packages, or the gluing-in of a **Component** (such as a card, sheet, or CD-ROM). Two **Components** are required for the *Inserting* process: the "mother" **Component** and the "child " **Component**. Inserting can be a selective process by means of inserting different "*child*" **Component**s. Information about the placement is needed to perform the process.

Input Resources

Name	Description
Component (Mother)	Designates where to insert the child **Component**.
Component (Child)	The **Component** to be inserted in the mother **Component**.
InsertingParams	Specific parameters, such as placement, to set up the machinery.
DBRules ?	Database input that describes whether the child should be inserted for a particular instance **Component**. In this version the schema is only human readable text.
DBSelection ?	Database input that describes whether the child should be inserted for a particular instance **Component**.
IdentificationField ?	Information about identification marks on the **Component**.

Output Resources

Name	Description
Component	A mother **Component** is produced containing the inserted child **Component**.

6.5.21 Jacketing

New in JDF 1.1

The jacketing is the process where the book is wrapped by a jacket that needs to be folded twice. As long as the book is specified and the jacket dimensions are known, there are just a few important details. If the jacketing device also creases the jacket, this can be described with a *Combined* process of *Jacketing* and *Creasing*.

Input Resources

Name	Description
JacketingParams	Specific parameters to set up the machinery.
Component (Book)	The book that the jacket is wrapped around.
Component (Jacket)	The description of the jacket.

Output Resources

Name	Description
Component	The jacketed book.

6.5.22 Labeling

New in JDF 1.1

A label can be attached to a bundle. The label can contain information on the addressee, the product, the product quantities, etc., which can be different for each bundle.

Input Resources

Name	Description
Component	The *Labeling* process labels one **Component** with a set of labels.
Component(Label) ?	The label to be attached to the **Component.**
LabelingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the labeled **Component**.

6.5.23 Laminating

In the *Laminating* process, a plastic film is bonded to one or both sides of a **Component's** media, and adhered (under pressure) with either a thermal setting or pressure sensitive adhesive.

Input Resources

Name	Description
Component	A Component is required for *Laminating*.
LaminatingParams	Specific parameters to set up the machinery.
Media ?	The laminating foil material.

Output Resources

Name	Description
Component	One **Component** is produced: the laminated component.

6.5.24 LongitudinalRibbonOperations

Deprecated in JDF 1.1.

In-line finishing within web printing presses can include folding, perforating, or applying a line of glue on the ribbon while it is traveling in longitudinal direction. In version 1.1.of JDF and beyond, in-line finishing is described using the "standard" finishing processes, e.g., *Creasing*, *Cutting,* or *Folding* in a combined node with *ConventionalPrinting*.

Input Resources

Name	Description
Component	The **Component** can consist of more than one web or ribbon that has been combined with the *Gathering* process.
LongitudinalRibbonOperation-Params	Specific parameters to set up the machinery tools for the *Longitudinal-RibbonOperations* process.

Output Resources

Name	Description
Component +	A ribbon is produced that is used in other postpress processes. If the *LongitudinalRibbonOperations* process was slitting, more than one **Component** is produced.

6.5.25 Numbering

Numbering is the process of stamping or applying variable marks in order to produce unique components, for items such as lottery notes or currency. No database access is required, and the counters automatically increase incrementally. *Numbering* is also used for alphanumeric, automatic, and unique marking.

Input Resources

Name	Description
Component	One **Component**, such as a printed sheet or a pile of sheets, are modified in the *Numbering* process.
NumberingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the numbered sheet.

6.5.26 Palletizing

New in JDF 1.1

Bundles, stacks, piles or boxes can be loaded onto a palette.

Input Resources

Name	Description
Component	The *Palletizing* process describes placing the bundle that is represented by the **Component** onto a palette.
PalletizingParams	Specific parameters to set up the machinery.
Pallet	The palette.

Output Resources

Name	Description
Component	One **Component** is produced: the loaded palette.

6.5.27 Perforating

New in JDF 1.1

Perforating describes any process where a **Component** is perforated.

Input Resources

Name	Description
Component	This process consumes one **Component**: the printed sheets.
PerforatingParams	Details of the *Perforating* process.

Output Resources

Name	Description
Component	One **Component** is produced.

6.5.28 PlasticCombBinding

In the *PlasticCombBinding* process, a plastic insert wraps through prepunched holes in the substrate. Most often, these holes are rectangular and elongated. After the plastic comb is opened with a special tool, the prepunched block of sheets—often together with a top and button cover—is inserted onto the "teeth" of the plastic comb. When released from the machine, the teeth return to their original cylindrical positions with the points tucked into the backside of the spine area. Special machinery can be used to reopen the plastic comb binding.

Input Resources

Name	Description
Component	The operation requires one component: the pile of sheets often including a top and button cover.
PlasticCombBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the plastic-comb-bound component forming an item such as a calendar.

6.5.29 RingBinding

In this process, prepunched sheets are placed in a ring binder. Ring binders have different numbers of rings that are fixed to a metal backbone. In most cases, two, three, or four metal rings hold the sheets together as long as the binding is closed. Depending on the amount of sheets to be bound together, ring binders of different thickness must be used.

Input Resources

Name	Description
Component (BookBlock)	The operation requires one component: the pile of prepunched sheets to be inserted into the ring binder.
Component ? (RingBinder)	The empty ring binder that might have been printed, for example, before it is used during the *RingBinding* process.
RingBindingParams	Specific parameters to set up the process/machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the ring-bound component forming an item such as a calendar.

6.5.30 SaddleStitching

Deprecated in JDF 1.1

In *SaddleStitching*, signatures are collected so that all sections have a common spine, and then stitched with staples through the spine. *SaddleStitching* has been replaced by *Stitching* in JDF 1.1.

Input Resources

Name	Description
Component	The only required **Component** is the collected pile.
SaddleStitchingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	The stitched-together components.

6.5.31 ShapeCutting

New in JDF 1.1

The *ShapeCutting* process can be performed using tools such as hollow form punching, perforating, or die-cutting equipment.

Input Resources

Name	Description
Component	This process consumes one **Component**: the printed sheets.
ShapeCuttingParams	Details of the *ShapeCutting* process.
Tool ?	The cut die

Output Resources

Name	Description
Component	One **Component** is produced.

6.5.32 Shrinking

New in JDF 1.1

Shrink-wrap must be treated in order to shrink.

Input Resources

Name	Description
Component	The *Wrapping* process wraps a bundle that is represented by a **Component**.
ShrinkingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the loaded palette.

6.5.33 SideSewing

Deprecated in JDF 1.1 Replaced by *ThreadSewing*.

This is a binding technique resulting in robust products that have a significant loss of inner margin space and poor handling characteristics. For these reasons, other binding techniques are used more often. In **SideSewing**, the first step is to create the holes in the book block and inject the glue (see Section 6.5.45.2 HoleMaking). Then the entire book is sewn at once with a *ThreadMaterial* such as *Cotton or Polyester*. If the book block is rather thick, a *Stitching* process using wire might be performed before **SideSewing**.

Input Resources

Name	Description
Component	The only required **Component** is the gathered sheets.
SideSewingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	The **Component** is produced.

6.5.34 SpinePreparation

New in JDF 1.1

The **SpinePreparation** process describes the preparation of the spine of book blocks for hard and soft cover book production, e.g., milling and notching.

Input Resources

Name	Description
SpinePreparationParams	Specific parameters to set up the machinery.
Component	The raw book block.

Output Resources

Name	Description
Component	The book block with a processed spine.

6.5.35 SpineTaping

New in JDF 1.1

SpineTaping describes the process of applying a tape strip to the spine of a book block. It also describes the process of applying kraft paper to a hard cover book block.

Input Resources

Name	Description
SpineTapingParams	Specific parameters to set up the machinery.
Component	The book block that the spine is taped to.

Output Resources

Name	Description
Component	The book block with the spine.

6.5.36 Stacking

New in JDF 1.1

The stacking process collects physical resources (products) and produces a pile, stack or bundle for delivery. In a standard production each bundle consists of the same amount of identical products, possibly followed by one or more odd-

count bundles. In a production with variable data (e.g., newspaper dispatch, demographic production or individual ad-dressed products), each bundle has a variable amount of products, and, in the worst case, each product can be different from the others. The input components are single products; the output components are stacks of this product.

Input Resources

Name	Description
Component	The *Stacking* process consumes one **Component** and stacks it onto a stack.
StackingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the stack of input Components.

6.5.37 Stitching

Gathered or collected sheets or signatures are stitched together with a cover.

Input Resources

Name	Description
Component	The only required **Component** is the pile of gathered sheets, including the cover.
StitchingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the gathered or collected sheets includ-ing the cover stitched together.

6.5.38 Strapping

New in JDF 1.1

A bundle can be strapped. There are different kinds of strapping, e.g., single (one strap around the bundle), double (two parallel straps), and cross (two crossed straps).

Input Resources

Name	Description
Component	The *Strapping* process puts straps around a bundle that is represented by a **Component**.
StrappingParams	Specific parameters to set up the machinery.
Strap ?	The straps used.

Output Resources

Name	Description
Component	One **Component** is produced: the strapped **Component**.

6.5.39 StripBinding

New in JDF 1.1

Hard plastic strips are held together by plastic pins, which in turn are bound to the strips with heat. The sheets to be bound must be prepunched so that the top strip with multiple pins fits through the assembled material. It is then connected to the bottom strip with matching holes for the pins. The binding edge is often compressed in a special

machine before the excess pin length is cut off. The backstrip is permanently fixed with plastic clamping bars and cannot be removed without a special tool.

Input Resources

Name	Description
Component	The operation requires one component: the block of sheets to be bound.
StripBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the Velobound component forming an item such as a book.

6.5.40 ThreadSealing

New in JDF 1.1

Similar to Smythe sewing, *ThreadSealing* involves sewing the signatures at the spine of the book. After the signatures are sewn they are gathered and run through the perfect binder. The perfect binder however does not grind the spine. Instead the binding adhesive (which attaches the cover) envelops the thread that holds the book together. This special thread holds to the glue to create a sewn book with most of the same properties as Smythe sewing.

Input Resources

Name	Description
Component	This process consumes one **Component**: the printed sheets.
ThreadSealingParams	Details of the *ThreadSealing* process.

Output Resources

Name	Description
Component	One **Component** is produced.

6.5.41 ThreadSewing

This process may include a gluing application, which would be used principally between the first and the second or the last and the last sheet but one. Gluing may also be necessary if different types of paper are used.

Input Resources

Name	Description
Component	The operation requires one component: the gathered sheets.
ThreadSewingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the thread-sewn components forming an item such as a raw book block.

6.5.42 Trimming

The *Trimming* process is performed to adjust a book block or sheet to its final size. In most cases, it follows a block joining process, and the process is often executed as an in-line operation of a production chain. For example, the binding station may deliver the book blocks to the trimmer. A *Combined* operation in the trimming machinery would then execute a cut at the front, head, and tail in a cycle of two operations. Closed edges of folded signatures would then be opened while the book block is trimmed to its predetermined dimensions.

Some trimming machines, such as magazine production systems, can produce N-ups. In every case, however, the additional trimming cuts that divide the N-ups result in separated book blocks. Sometimes a stripe is trimmed out between the book blocks. To describe these operations, multiple *Trimming* processes must be defined in JDF.

Input Resources

Name	Description
Component	A bound book block is required for *Trimming*.
TrimmingParams	Specific parameters, e.g., trim size, to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the trimmed component.

6.5.43 WireCombBinding

The *WireCombBinding* is a technique that creates bindings not meant to be reopened later. *WireCombBinding* is often named *Wire-O®-binding*. Metal wire, wire with plastic, or pure plastic is used to fasten prepunched sheets of paper, cardboard, or other such materials. The wire—often formed as a double wire—is inserted into the holes, then curled to create a circular enclosure.

Input Resources

Name	Description
Component	The operation requires one component: the pile of preprinted sheets often including a front and back cover.
WireCombBindingParams	Specific parameters to set up the machinery.

Output Resources

Name	Description
Component	One **Component** is produced: the wire-comb bound component forming an item such as a calendar.

6.5.44 Wrapping

New in JDF 1.1

Single products, bundles or pallets can be wrapped by film or paper.

Input Resources

Name	Description
Component	The *Wrapping* process wraps a bundle that is represented by a **Component**.
WrappingParams	Specific parameters to set up the machinery.
Media ?	The wrapping material.

Output Resources

Name	Description
Component	One **Component** is produced: the wrapped **Component**.

6.5.45 Postpress Processes Structure

6.5.45.1 Block Production
This subcategory of the postpress processes merges together all the processes for making a book block. First the block is compiled using the Collecting and Gathering processes. After that, it is combined using one or several of the block

joining processes, including *CoverApplication, SaddleStitching*, *SideSewing*, *SpineTaping*, *Stitching*, and *Thread-Sewing*. The workflow using these processes eventually produces a **Component** that can be trimmed.

6.5.45.1.1 Block Compiling

The *Gathering* and *Collecting* processes are used to position unfolded sheets and/or folded sheets in a planned order. These operations set a fixed page sequence in preparation for three-side trimming and binding. Block compiling includes:

- Collecting
- Gathering

6.5.45.1.2 Block Joining

The block joining processes can be grouped into two major subcategories: conventional binding methods, which includes the processes of *Stitching*, *SaddleStitching*, *CoverApplication, SpinePreparation, SpineTaping*, *ThreadSewing*, and *SideSewing*; and single-leaf binding methods, which are listed in Section Single-Leaf Binding Methods. Together they form a subcategory of block-production processes. All of these processes, which are known as block-joining processes, unite sheets and/or folded sheets lying loose on top of each other.

There are numerous possible binding methods. The most prominent ones are modeled by the processes described in the following sections. Many of them can be part of a combined production chain being performed as in-line tasks. Block Joining includes:

- AdhesiveBinding
- CoverApplication
- SaddleStitching
- SideSewing
- SpinePreparation
- SpineTaping
- Stitching
- ThreadSewing

6.5.45.1.2.1 Single-Leaf Binding Methods

Besides the conventional binding methods, there is a multifaceted group of binding methods for single-leaf bindings. This group can again be subdivided into two subtypes: loose-leaf binding and mechanical binding, each of which is described in the sections that follow.

6.5.45.1.2.1.1 Loose-Leaf Binding Method

This binding techniques allow contents to be changed, inserted, or removed at will. There are two essential groups of loose-leaf binding systems: those that require the paper to be punched or drilled and those that do not. The *RingBinding* method, described in the next section, is the most prominent binding in the loose-leaf binding category. Loose-Leaf Binding Methods include:

- RingBinding

6.5.45.1.2.1.2 Mechanical Binding Methods

Single leafs are fastened into what is essentially a permanent system that is not meant to be reopened. However, special machinery can be used to reopen some of the mechanical binding systems described below.

In mechanical binding, printing and folding can be done in a conventional manner. The gathered sheets, however, often require the back to be trimmed, as well as the other three sides. Mechanical bindings are often used for short-run jobs such as ones that have been printed digitally. The most prominent mechanical binding processes are described in the sections that follow. Mechanical Binding Methods include:

- ChannelBinding
- CoilBinding
- PlasticCombBinding
- RingBinding

- StripBinding
- WireCombBinding

6.5.45.2 HoleMaking
See HoleMaking.

6.5.45.3 Laminating
See Laminating.

6.5.45.4 Numbering
See Numbering.

6.5.45.5 Packaging Processes
The individual processes defined in this section replace the deprecated *Packing* process. Packaging processes include:

- BoxPacking
- Labeling
- Palletizing
- Shrinking
- Stacking
- Strapping
- Wrapping

Each of these processes share a common coordinate system as depicted below:
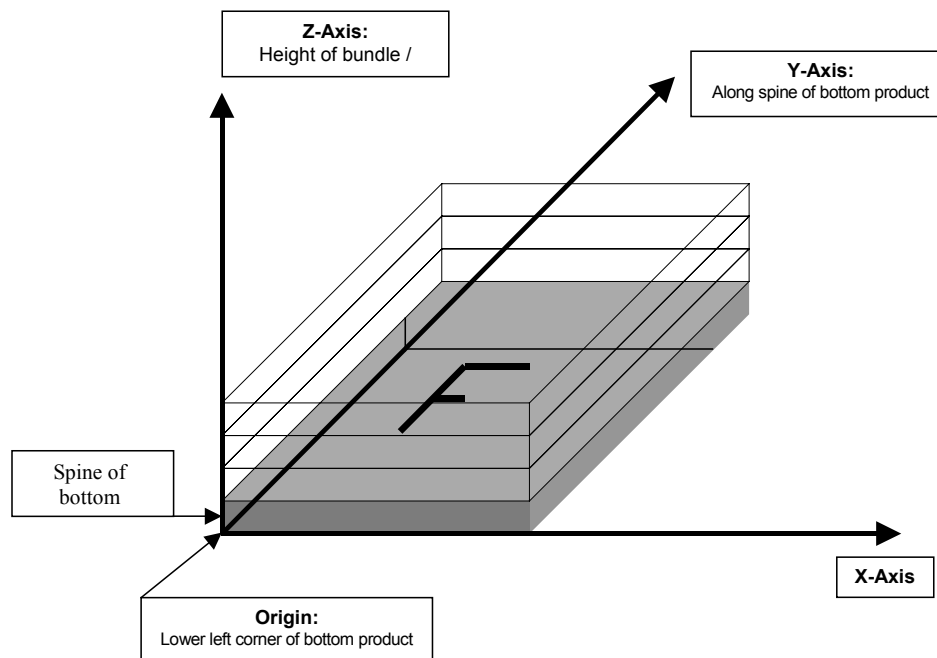
Figure 6.2 Packaging Process Coordinate System

6.5.45.6 Processes in Hardcover Book Production
The following processes refer to the production of hard cover books. As there are several processes which are needed to produce a hardcover, some of them are optional, others are essential. The processes described are in detail:

CaseMaking: Production of hard cover book cases.

BlockPreparation: The optional hardcover design elements like rounding and backing, ribbon band, headband, side gluing, and tightbacking are described here. Application of kraft paper to the book block is described in the ***SpineTaping*** process.

CasingIn: In this process, the case and the prepared book block are brought together.

Jacketing: In the jacketing process, the jacket is wrapped around the hardcover book.

Processes in Hardcover Book Production include:

- BlockPreparation
- CaseMaking
- CasingIn
- HeadBandApplication
- Jacketing

6.5.45.7 Sheet Processes

Many printing processes produce sheets that must be processed further in finishing operations. The web processes presented in the preceding sections result in sheets that are treated in much the same way as sheets produced by sheetfed printing presses. The following processes describe these sheet finishing operations. Sheet processes include:

- Creasing
- Cutting
- Embossing
- Folding
- Gluing
- Perforating
- ShapeCutting
- ThreadSealing

6.5.45.8 Tip-on/in

The following processes (***EndSheetGluing***, ***Inserting***) are part of the postpress operations. They can be grouped together as the tip-on/in processes. Both processes can be performed by hand, tip-on/in machine, or by a press. Tip-on/in includes:

- EndSheetGluing
- Inserting

6.5.45.9 Trimming

See Trimming.

6.5.45.10   Web Processes

This subchapter of the postpress processes is dedicated to web and ribbon operations, i.e., operations that require a web or a ribbon to execute. In essence, a ribbon is a web that has been slit or cross-cut. More specifically, a web is a continuous strip of **Media** to be used for printing, e.g., paper or foil. This substrate is called "web" while it is threaded through the printing machinery, but once it has run through the ***Dividing*** process and been slit, the web no longer exists. In its place are ribbons or sheets.

A ribbon, then, is the part of the web that enters the folder. If the web is never slit, however, the web and the ribbon are identical. Slitting and salvage-trim operations on a web can result in one or more ribbons. A ribbon can be further subdivided after it has been slit. After the ***Dividing*** process, sheets are treated further. The ***Gathering*** process and ***Folding*** process also handle web and ribbon applications.

Chapter 7 Resources

Introduction

Resources represent inputs and outputs, the 'things" that are produced, modified, consumed, or in any way used by nodes. A more thorough description was provided in Section 3.7 Resources. The resources in this chapter are divided into two sections. The first section documents all of the resources of class *Intent*. The second section documents the rest of the resources that have been defined for JDF.

7.1 Intent Resources

As was described in Section 4.1.1 Product Intent Constructs, intent resources are designed to narrow down the available options when defining a JDF job. Many of the elements in intent resources are optional. If an optional element of an intent resource is omitted, and no additional information is specified in the description, the value defaults to "don't care".

    All intent resources share a set of subelements that allow a Request for Quote to describe a range of acceptable values for various aspects of the product. These elements, taken together, allow an administrator to provide a specific value for the quote. Section 7.1.1, below, describes these elements.

    Each of the following sections begins with a brief narrative description of the resource. Following that is a list containing details about the properties of the resource, as shown below. The first item in the list provides the class of the resource, which, in this section is always *Intent*. For more information on resource class, see Section 3.7.1 Resource Classes. A template of this list is shown below.

    After the list describing the resource properties, each section contains tables that outline the structure of each resource and, when applicable, the abstract or subelement information that pertains to the resource structure. The first column contains the name of the attribute or element. In some cases, a resource will contain an element with more than one value associated with it. If this is the case, the element name is listed as often as it appears, and a term in parentheses that identifies the kind of element is included in the column. A template of these tables is also provided below.

Resource Properties Template

Resource class:	Defines the resource class.
Resource referenced by:	List of parent resources that contain elements of this type. Only valid for elements.
Example Partition:	List of valid partitioning boundaries: *BlockName, DocIndex, DocRunIndex, DocSheetIndex, FountainNumber, LayerIDs, Location, Option, PageNumber, PartVersion, PreviewType, RibbonName, Run, RunIndex, RunTag, RunPage, Separation, SetIndex, SheetIndex, SheetName, Side, SignatureName, TileID*, *WebName* If a partition is specified, the resource may contain nested elements of its own type. Note that resources may also be partitioned by keys that are not included in the list, e.g., *Option*, which is valid for any resource.
Input of processes:	List of node types that use the resource as an input resource.
Output of processes:	List of node types that create the resource as an output resource.

Resource Structure Template

Name	Data Type	Description
Name of attribute	data type of attribute	Usage of the attribute.
Name of element	element	Subelements that must be defined locally within the resource.
Name of element	refelement	Elements that are based on other atomic resources or resource elements. These may either be in-line elements or instances of **ResourceRef** elements (see Section 3.8.6). In case of **ResourceRef** elements, a "Ref" must be appended to the name specified in the table column entitled "Name".

7.1.1   Intent Resource Span Subelements

Intent resources contain subelements that allow spans of values to be specified. These subelements also provide mechanisms to select a set of values from the provided range and map them to a set of quotes. These subelements are called span elements. The abstract span element to be used is determined by the data type of the values to be recorded. All possible span elements are listed in the following table.

Each span element contains further attributes or subelements. The contents shared by all span elements are listed in the Section 7.1.1.1 Structure of Abstract Span Subelement, below, and the contents particular to each span element type are described in the sections that follow.

Span Element Types	Data Type	Description
DurationSpan New in JDF 1.1	element	Describes a set of duration values.
EnumerationSpan	element	Describes a set of enumeration values.
IntegerSpan	element	Describes a numerical range of integer values.
NameSpan	element	Describes a set of NMTOKEN values.
NumberSpan	element	Describes a numerical range of values.
OptionSpan	element	Describes an intent in which the principal information is that a specific option is requested.
ShapeSpan New in JDF 1.1	element	Describes a set of shape values.
StringSpan	element	Describes a set of string values.
TimeSpan	element	Describes a set of dateTime values.
XYPairSpan	element	Describes a set of XYPair values.

7.1.1.1   Structure of Abstract Span Subelement

Abstract span elements of intent resources have a common set of attributes and elements that define the priority, data type, and requested identity of the element. These attributes are described in the following table. In addition, abstract Span elements have 3 attributes that define the aspects of the span. The data type of these values depends on the data type of the span and is defined in the following sections:

Actual: The accepted actual value

Preferred: A preferred value

Range: A proposed range of values

Name	Data Type	Description	
DataType	enumeration	Describes the data type of the span element within an intent resource. This attribute is provided for applications that do not have access to schema validation. Possible values are:	
		DurationSpan	*OptionSpan*
		EnumerationSpan	*ShapeSpan*
		IntegerSpan	*StringSpan*
		NameSpan	*TimeSpan*
		NumberSpan	*XYPairSpan*

Name	Data Type	Description
Priority ?	enumeration	Indicates the importance of the specific intent. The following values have prescribed meanings: *None* – Default value. *Suggested* – The customer will accept a value of **Actual** that is different than the value of **Preferred** or outside of **Range**. *Required* – **Actual** must be equal to **Preferred** or within **Range**. Note that the attribute **Preferred** is available in the data types which inherit from this abstract type.

The following table describes the allowed values defined by the combination of *Range*, *Preferred,* and *Priority* in Span resources.

Priority	Preferred Exists	Range Exists	Suggested Value defined by:	Required Value defined by:
None	any	any	-	-
Suggested	yes	no	*Preferred*	-
Suggested	yes	yes	*Preferred*	-
Suggested	no	yes	*Range*	-
Required	yes	no	-	*Preferred*
Required	yes	yes	*Preferred*	*Range*
Required	no	yes	-	*Range*

7.1.1.2 Structure of the DurationSpan Subelement

New in JDF 1.1

This span subelement is used to describe a selection of instances in time. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	duration	The actual value selected for the quote.
Preferred ?	duration	Provides a value specified by the person submitting the request, indicating what that person prefers. **Preferred** must fall within the range of values specified in **Range**.
Range ?	DurationRange	Range provides a valid range of time durations. Default = **Preferred.**

7.1.1.3 Structure of the EnumerationSpan Subelement

This span subelement is used to describe ranges of enumerative values. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement. It is identical to the NameSpan element except for the fact that it describes a closed list of enumeration values.

Name	Data Type	Description
Actual ?	enumeration	The actual value selected for the quote.
Preferred ?	enumeration	Provides a value specified by the person submitting the request, indicating what that person prefers. **Preferred** must fall within the range of values specified in **Range**.
Range ?	enumerations	Provides a set of discreet enumeration values. Default = **Preferred.**

7.1.1.4   Structure of the IntegerSpan Subelement

This span subelement is used to describe ranges of integer values. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	integer	The actual value selected for the quote.
Preferred ?	integer	Provides a value specified by the person submitting the request, indicating what that person prefers. The value of *Preferred* must fall within the range of values specified in *Range*.
Range ?	IntegerRangeList	Provides either a set of discreet values, a range of values, or a combination of the two that comprise all allowed values for the span. Default = *Preferred.*

7.1.1.5   Structure of the NameSpan Subelement

This span subelement is used to describe name ranges. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement. It is identical to the EnumerationSpan element except for the fact that it describes an extensible list of NMTOKEN values.

Name	Data Type	Description
Actual ?	NMTOKEN	The actual value selected for the quote.
Preferred ?	NMTOKEN	Provides a value specified by the person submitting the request, indicating what that person prefers. *Preferred* must fall within the range of values specified in *Range*.
Range ?	NMTOKENS	Provides a set of discreet values. Default = *Preferred.*

7.1.1.5.1   Specifying New Values in a NameSpan Subelement

NameSpan elements generally define an open list of predefined values. If a value that is not included in the list must be specified, a comment that defines that value can be included in the NameSpan using the new name as a *Name* attribute of the comment, as demonstrated in the following example:

```
<HoleType DataType="NameSpan" Range="36Hole 42Hole">
<Comment Name="36Hole">6 equidistant holes on each side of a hexagonal piece of paper
</Comment>
<Comment Name="42Hole">7 equidistant holes on each side of a hexagonal piece of paper
</Comment>
</HoleType>
```

7.1.1.6   Structure of the NumberSpan Subelement

This span subelement is used to describe a numerical range of values. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	number	The actual value selected for the quote.
Preferred ?	number	Provides a value specified by the person submitting the request, indicating what that person prefers. *Preferred* must fall within the range of values specified in *Range*.
Range ?	NumberRangeList	Provides either a set of discreet values, a range of values, or a combination of the two. Default = *Preferred.*

7.1.1.7   Structure of the OptionSpan Subelement

This span subelement is used to describe a range of options or boolean values. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	boolean	The actual value selected for the quote. If the option is included = *true*.

Name	Data Type	Description
Preferred ?	boolean	Provides a value specified by the person submitting the request, indicating what that person prefers.
Detail ?	string	*Detail* provides information about the option.

7.1.1.8 Structure of the ShapeSpan Subelement

New in JDF 1.1

This span subelement is used to describe ranges of numerical value pairs. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	shape	The actual value selected for the quote.
Preferred ?	shape	Provides a value specified by the person submitting the request, indicating what that person prefers. The value of *Preferred* must fall within the range of values specified in *Range*.
Range ?	ShapeRangeList	Provides either a set of discreet values, a range of values, or a combination of the two that comprise all allowed values for the span. Default = *Preferred.*

7.1.1.9 Structure of the StringSpan Subelement

This span subelement is used to describe string ranges. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	string	The actual value selected for the quote.
Preferred ?	string	Provides a value specified by the person submitting the request, indicating what that person prefers. *Preferred* must fall within the range of values specified in *Range*.
Range *	telem	Provides a set of discreet string values. Default = *Preferred.*

7.1.1.10 Structure of the TimeSpan Subelement

This span subelement is used to describe a selection of instances in time. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	dateTime	The actual value selected for the quote.
Preferred ?	dateTime	Provides a value specified by the person submitting the request, indicating what that person prefers. *Preferred* must fall within the range of values specified in *Range*.
Range ?	TimeRange	Range provides a valid time period. Default = *Preferred.*

7.1.1.11 Structure of the XYPairSpan Subelement

This span subelement is used to describe ranges of numerical value pairs. It inherits from the abstract span element described in Section 7.1.1.1 Structure of Abstract Span Subelement.

Name	Data Type	Description
Actual ?	XYPair	The actual value selected for the quote.
Preferred ?	XYPair	Provides a value specified by the person submitting the request, indicating what that person prefers. The value of *Preferred* must fall within the range of values specified in *Range*.

Name	Data Type	Description
Range ?	XYPairRangeList	Provides either a set of discreet values, a range of values, or a combination of the two that comprise all allowed values for the span. Default = *Preferred.*

7.1.2 ArtDeliveryIntent

This resource specifies the prepress art delivery intent for a JDF job and maps the items to the appropriate reader pages and separations. Art delivery refers to any physical or electronic asset that is required for processing the job.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Example Partition:	*Option*
Input of processes:	Any product node
Output of processes:	-

Resource Structure

Name	Data Type	Description
ArtDeliveryDate ? New in JDF 1.1	TimeSpan	Specifies the latest time by which the transfer of the artwork will be made.
ArtDeliveryDuration ? New in JDF 1.1	DurationSpan	Specifies the latest time by which the transfer will be made relative to the date of the purchase order. Within an RFQ or a Quote only one of either *ArtDeliveryDate* or *ArtDeliveryDuration* may be specified. Within a purchase order only *ArtDeliveryDate* is allowed.
ArtHandling ? New in JDF 1.1	Enumeration-Span	Describes what should happen to the artwork after usage. The return or pickup address must be specified by a Contact with *ContactTypes* including "*ArtReturn*". Possible values are: *ReturnWithProof* – the artwork is delivered back to the customer together with the proof, if there is any. *ReturnWithProduct* – the artwork is delivered back to the customer together with the final product. The default. *Return* – the artwork is delivered back independently directly after usage. *Pickup* – the customer picks up the artwork *Destroy* – the printer must destroy the artwork *PrinterOwns* – the artwork belongs to the printer *Store* – the printer has to store the artwork for future purposes
DeliveryCharge ? New in JDF 1.1	Enumeration-Span	Specifies who pays for a delivery being made by a third party. Possible values are: *Printer* *Buyer:* The default.

Name	Data Type	Description
Method ?	NameSpan	Identifies a required delivery method, may be a generic item, e.g.: *EMail* *ExpressMail* *InterofficeMail* *OvernightService* *Courier* *CompanyTruck* May also be a delivery service brand, e.g.: *UPS* *DHL* *FedEx*
PreflightStatus ? New in JDF 1.1	enumeration	Information about a preflight process probably applied to the artworks before being submitted. Possible values are: *NotPerformed* – No preflighting was applied. The default. *WithErrors* – Preflighting resulted in error and warning messages. *WithoutErrors* – Preflighting was successful.
ReturnList ? New in JDF 1.1	NMTOKENS	Type of printer created intermediate materials that should be sent to the customer after usage. Possible values include: *DigitalMedia* – Digital data on media such as a CD. *DigitalNetwork* – Digital data via network. *ExposedPlate* – Preexposed press plates, usually used for a rerun. *ImposedFilm* – Film of the imposed surfaces. *LooseFilm* – Film of individual pages or sections. *OriginalPhysicalArt* – Analog artwork, e.g,. reflective or transparencies *Tool* – Tools required for processing the job, e.g., a die for die cutting or embossing stamp. *None* – No intermediate materials should be returned to the customer. The default.
ReturnMethod ? New in JDF 1.1	NameSpan	Identifies a required delivery method for returning the artwork, if *ArtHandling* = *Return* and for the printer created materials listed in *ReturnList*. The predefined values are the same as the list specified in *Method*.
Transfer ? New in JDF 1.1	Enumeration-Span	Describes the responsibility of the transfer. Possible values are: *BuyerToPrinterDeliver* – the buyer delivers the artwork to the printer. The printer may specify in the quote a special **Contact** with *ContactTypes* including *Delivery*, where the buyer should send the artwork. *BuyerToPrinterPickup* – the printer picks up the artwork. The **Contact** with *ContactTypes* including *pickup* describes, where the printer has to pick up the artwork.
ArtDelivery + Modified in JDF 1.1	element	Individual delivery.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the art delivery. This must only be specified if the printer is expected to pick up the art delivery at this address. Defaults to an empty element, i.e., the art is delivered to the printer.

Name	Data Type	Description
Contact * New in JDF 1.1	refelement	Address and further information about the transfer of the artwork. The actual delivery address is specified as the Address of the Contact with *ContactTypes* including *Delivery*. Only one Contact with *ContactTypes* including *Delivery* may be specified. The actual pickup address is specified as the Address of the Contact with *ContactTypes* including *Pickup*. Only one Contact with *ContactTypes* including *Pickup* may be specified.

Structure of ArtDelivery Elements

Each ArtDelivery element defines a set of existing products that are required to create the specified product. Attributes that are specified in an ArtDelivery element overwrite those that are specified in their parent ArtDeliveryIntent element. If optional attributes are not specified, their values default to the values specified in ArtDeliveryIntent.

Name	Data Type	Description
Amount ?	integer	Number of physical objects to be delivered. Only valid if no detailed resource description, i.e. ExposedMedia, RunList, ScanParams or Tool is specified.
ArtDeliveryDate ? New in JDF 1.1	TimeSpan	Specifies the latest time by which the transfer of the artwork will be made.
ArtDeliveryDuration ? New in JDF 1.1	DurationSpan	Specifies the latest time by which the transfer will be made relative to the date of the purchase order. Within an RFQ or a Quote only one of either *ArtDeliveryDate* or *ArtDeliveryDuration* may be specified. Within a purchase order only the *ArtDeliveryDate* is allowed.
ArtDeliveryType New in JDF 1.1	NMTOKEN	Type of artwork supplied. Possible values include: *DigitalMedia* – Digital data on media such as a CD. *DigitalNetwork* – Digital data via network. *ExposedPlate* – preexposed press plates, usually used for a rerun. *ImposedFilm* – Film of the imposed surfaces. *LooseFilm* – Film of individual pages or sections. *OriginalPhysicalArt* – analog artwork, e.g. reflective or transparencies *Proof* – physical proof delivered with digital scan or separated film asset. *Tool* – Tools required for processing the job, e.g. a Die for Die cutting or embossing stamp. *None* – No artwork exists and it must be created

Name	Data Type	Description
ArtHandling ? New in JDF 1.1	Enumeration-Span	Describes what should happen to the artwork after usage. The return or pickup address must be specified by a Contact with *ContactTypes* including "*ArtReturn*". Possible values are: *ReturnWithProof* – the artwork is delivered back to the customer together with the proof, if there is any. *ReturnWithProduct* – the artwork is delivered back to the customer together with the final product. *Return* – the artwork is delivered back independently directly after usage. *Pickup* – the customer picks up the artwork *Destroy* – the printer must destroy the artwork *PrinterOwns* – the artwork belongs to the printer *Store* – the printer has to store the artwork for future purposes Defaults to the value of *ArtHandling* in **ArtDeliveryIntent**.
DeliveryCharge ? New in JDF 1.1	Enumera-tionSpan	Specifies who pays for a delivery being made by a 3rd party. Possible values are: *Printer* *Buyer* Defaults to the value of *DeliveryCharge* in **ArtDeliveryIntent**
HasBleeds ?	boolean	If *true*, the file has bleeds. Default = *false*.
IsTrapped ?	boolean	If *true*, the file has been trapped. Default = *false*.
Method ?	NameSpan	Identifies a required delivery method, may be either a generic item from the following list: *EMail*, *ExpressMail*, *InterofficeMail,* *OvernightService,* *Courier,* *CompanyTruck*. May also be a delivery service brand. For example: *UPS* *DHL* *FedEx* Defaults to the value of *Method* in **ArtDeliveryIntent**.
PageList ?	Integer-RangeList	Set of pages of the output **Component** that are filled by this **ArtDelivery**. This maps the pages in the **ArtDelivery** to the Pages in the product that is produced. For example if PageList = "3~5", page 0 of the ArtDelivery (e.g., RunList) is page 3 in the product, page 1 is page 4, etc. Default = "0~-1", i.e., all pages in reader order.
PreflightOutput ? New in JDF 1.1	URL	Pointer to the output information created by the preflight tool, if *PreflightStatus* is either *WithoutErrors* or *WithErrors*.

Name	Data Type	Description
PreflightStatus ? New in JDF 1.1	enumeration	Information about a preflight process. The values are identical to those of *PreflightStatus* in **ArtDeliveryIntent**. Defaults to the value of *PreflightStatus* in **ArtDeliveryIntent**.
ReturnMethod ? New in JDF 1.1	NameSpan	Identifies a required delivery method for returning the artwork, if *ArtHandling* = *Return*. Defaults to the value of *ReturnMethod* in **ArtDeliveryIntent**.
Transfer ? New in JDF 1.1	Enumeration-tionSpan	Describes the responsibility of the transfer. The values are identical to those of *Transfer* in **ArtDeliveryIntent**. Defaults to the value of *Transfer* in **ArtDeliveryIntent**.
Company ? Deprecated in JDF 1.1	refelement	Address and further information about the art delivery. This must only be specified if the printer is expected to pick up the art delivery at this address. Defaults to the value of Company specified in the parent **ArtDeliveryIntent**.
Component ? Deprecated in JDF 1.1	refelement	Description of a physical component, e.g., physical artwork. If neither Component, ExposedMedia nor RunList are specified, no details of the ArtDelivery except the *ArtDeliveryType* and *Amount* are known.
Contact * New in JDF 1.1	refelement	Address and further information about the art transfer. Defaults to the value of Contact specified in the parent **ArtDeliveryIntent**.
ExposedMedia ?	refelement	Description of exposed media, e.g., film, plate or proof. If neither ExposedMedia, RunList, nor Tool are specified, no details of the ArtDelivery except the *ArtDeliveryType* and *Amount* are known.
RunList ?	refelement	Link to digital artwork. If neither ExposedMedia, RunList, nor Tool are specified, no details of the ArtDelivery except the *ArtDeliveryType* and *Amount* are known.
ScanParams ?	refelement	Description of a ScanParams that defines scanning details for the exposed media defined by ExposedMedia.
Tool ? New in JDF 1.1	refelement	Details of the Tool if *ArtDeliveryType* = *"Tool"*. If neither ExposedMedia, RunList, nor Tool are specified, no details of the ArtDelivery except the *ArtDeliveryType* and *Amount* are known.

7.1.3 BindingIntent

This resource specifies the binding intent for a JDF job using information that identifies the type of binding required and which side is to be bound. The input components that are used as a cover should have a *ProcessUsage* of *Cover*. The input components that are used as a hard cover jacket should have a *ProcessUsage* of *Jacket*. All other input components are bound in the order of their appearance in the ResourceLinkPool of the JDF node that contains the **BindingIntent**.

Resource Properties
Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
BackCoverColor ? New in JDF 1.1	Enumeration-Span	Defines the color of the back cover material of the binding. Allowed values are defined in Appendix A.2.8 NamedColor. If not specified, it defaults to the values defined in *CoverColor.*

Name	Data Type	Description
BindingOrder ? New in JDF 1.1	enumeration	Specifies whether the child **Components** should be collected or gathered if multiple child **Components** are combined. One of: *Collecting*: The child **Components** are collected on a spine and placed within one another. The first **Component** is on the outside. *Gathering*: The child **Components** are gathered on a pile and placed on top of one another. The first **Component** is on the top. The default. *List*: More complex ordering of child **Components** is specified using the BindList in this intent resource for this product.
BindingColor ?	Enumeration-Span	Defines the color of the spine material of the binding. Allowed values are defined in Appendix A.2.8 NamedColor. Default = don't care, i.e., system specified.
BindingLength ?	Enumeration-Span	Indicates which side should be bound when no content and, thus, no orientation is available but a quote for binding is required. *Long* – The default, if neither *BindingLength* nor *BindingSide* were specified. *Short*
BindingSide ?	Enumeration-Span	Indicates which side should be bound. Possible values are: *Top* *Bottom* *Right* *Left* Each of these values is intended to identify an edge of the job. These edges are defined relative to the orientation of the first page in the job with content on it. Default = *BindingLength* value, unless non-empty BindList was specified. If both *BindingSide* and *BindingLength* are specified, *BindingSide* has precedence.
BindingType Modified in JDF 1.1	Enumeration-Span	Describes the desired binding for the job. Possible values are: *Adhesive* – This type of binding can be handled with the **AdhesiveBinding** process. It includes perfect binding. Deprecated in JDF1.1 and replaced with *SoftCover* or *HardCover*. *ChannelBinding* – This type of binding can be handled with the **ChannelBinding** process. *CoilBinding* – This type of binding can be handled with the **CoilBinding** process. *EdgeGluing* – Gluing gathered sheets at one edge of the pile. This Type of Binding can be handled with the *Gluing* process. *HardCover* – This type of binding defines a hard cover bound book. *LooseBinding* – This type of binding defines a stack of pages with no additional binding. *PlasticComb* – This type of binding can be handled with the **PlasticCombBinding** process. *Ring* – This type of binding can be handled with the **RingBinding** process.

Name	Data Type	Description
		SaddleStitch – This type of binding can be handled with the **Stitching** process.
		Sewn – This type of binding can be handled with the **Thread-Sewing** process.
		SideSewn – This type of binding can be handled with the **ThreadSewing** process.
		SideStitch – This type of binding can be handled with the **Stitching** process.
		SoftCover – This type of binding defines a soft cover bound book. It includes perfect binding.
		StripBind – This type of binding can be handled with the **StripBinding** process.
		Tape – This type of binding is an inexpensive version of the *SoftCover*.
		ThreadSealing – This type of binding can be handled with the **ThreadSealing** process.
		StripBind – This type of binding can be handled with the **StripBinding** process.
		WireComb – This type of binding can be handled with the **WireCombBinding** process.
CoverColor ?	Enumeration-Span	Defines the color of the cover material of the binding. Allowed values are defined in Appendix A.2.8 NamedColor. Default = don't care, i.e., system specified.
AdhesiveBinding *?* Deprecated in JDF 1.1	element	Details of AdhesiveBinding.
BindList ? New in JDF 1.1	element	Details of binding of individual child Components.
BookCase *?* Deprecated in JDF 1.1	element	Details of the book Case. Used in Combination with AdhesiveBinding ,ThreadSewing or ThreadSealing.
ChannelBinding *?*	element	Details of ChannelBinding. Default = ChannelBinding value.
CoilBinding *?*	element	Details of CoilBinding. Default = CoilBinding value.
EdgeGluing ? New in JDF 1.1	element	Details of EdgeGluing. Default = EdgeGluing value.
HardCoverBinding *?* New in JDF 1.1	element	Details of HardCoverBinding. Default = HardCoverBinding value.
PlasticCombBinding *?*	element	Details of PlasticCombBinding. Default = PlasticCombBinding value.
RingBinding ?	element	Details of RingBinding. Default = RingBinding value.
SaddleStitching ?	element	Details of SaddleStitching. Default = SaddleStitching value.
SideSewing ?	element	Details of SideSewing. Default = SideSewing value.
SideStitching ?	element	Details of SideSewing. Default = SideSewing value.
SoftCoverBinding *?* New in JDF 1.1	element	Details of SoftCoverBinding. Default = SoftCoverBinding value.

Name	Data Type	Description
Tape ? New in JDF 1.1	element	Details of Tape binding. Default = Tape value.
Tabs ?	element	Details of Tabs. Default = no tabs
ThreadSealing ?	element	Details of ThreadSealing. Default = ThreadSealing value.
ThreadSewing?	element	Details of ThreadSewing. Default = ThreadSewing value.
StripBinding ? New in JDF 1.1	element	Details of StripBinding. Default = StripBinding value.
VeloBinding ? Deprecated in JDF 1.1	element	Details of VeloBinding. Renamed to StripBinding in JDF 1.1.
WireCombBinding ?	element	Details of WireCombBinding. Default = WireCombBinding value.

Structure of BindList Subelement
New in JDF 1.1

Name	Data Type	Description
BindItem *	element	Individual bind item description. Default = BindingIntent::BindingSide value if empty, i.e., as if the BindList element weren't there.

Structure of BindItem Subelement
New in JDF 1.1

Name	Data Type	Description
BindingType ?	EnumerationSpan	Describes the desired binding for the individual BindItem. The list of possible values is defined in **BindingIntent**:*BindingType.* Defaults to the value specified in the parent **BindingIntent**.
ChildFolio ?	XYPair	Definition of the fold between two pages in the BindItem component that is bound to the cover. The two numbers in the *ChildFolio* attribute are the page numbers of the two outer pages of the child **Component**, which touch the cover or another child **Component**. The pages are counted in the order, which is described in *FolioCount* of the child product. Defaults to the spine of the child.
ParentFolio	XYPair	Definition of the fold between two pages in the Cover **Component** that receive the BindItem. The two numbers in the *ParentFolio* attribute are the page numbers in the Cover **Component**, which touch the child **Component**. The pages are counted in the order, which is described in *FolioCount* of the cover product.
Transformation ?	matrix	Rotation and offset between the **Component** to be inserted and the "parent" **Component**. For details on transformations, see How and Where Coordinates and Transformations Are Used/Defined in JDF.

Name	Data Type	Description
WrapPages ?	IntegerRangeList	List of pages of the Cover that wrap around a BindItem after all folds are correctly positioned. It is sufficient to specify the pages of the *Front* surface of the cover. Note that this key must only be specified if the folding is ambiguous. Default = empty list.
BookCase *?*	element	Details of the hard cover book Case. Used in Combination with HardCoverBinding.
ChannelBinding *?*	element	Details of ChannelBinding.
CoilBinding *?*	element	Details of CoilBinding.
EdgeGluing ?	element	Details of EdgeGluing.
HardCoverBinding *?*	element	Details of HardCoverBinding.
PlasticCombBinding *?*	element	Details of PlasticCombBinding.
RingBinding ?	element	Details of RingBinding.
SaddleStitching ?	element	Details of SaddleStitching.
SideSewing ?	element	Details of SideSewing.
SideStitching ?	element	Details of SideStitching.
SoftCoverBinding *?*	element	Details of SoftCoverBinding.
Tape ?	element	Details of Tape binding.
Tabs ?	element	Details of Tabs.
ThreadSealing ?	element	Details of ThreadSealing.
ThreadSewing?	element	Details of ThreadSewing.
StripBinding ?	element	Details of StripBinding.
WireCombBinding ?	element	Details of WireCombBinding.

Structure of the AdhesiveBinding Subelement.

Deprecated in JDF 1.1

Name	Data Type	Description
Scoring ?	EnumerationSpan	Scoring option for **AdhesiveBinding**. Possible values are: *TwiceScored* *QuadScored* *None* Values are based on viewing the cover in its flat prebinding state.
SpineGlue ?	EnumerationSpan	Glue type used to define **AdhesiveBinding** procedures. Possible values are: *ColdGlue* *Hotmelt* *PUR* – Polyurethane Rubber
TapeBinding ?	OptionSpan	If *true*, a cloth tape which has been preglued with hot-melt adhesive is used in **AdhesiveBinding** the unmilled block., e.g., FastBack or DocuTech binding. Default = *false*

Structure of the BookCase Subelement.

Deprecated in JDF 1.1

This subelements contains details of the book case for hard cover book binding. The actual binding parameters are set in the appropriate AdhesiveBinding, ThreadSewing or ThreadSealing elements.

Name	Data Type	Description
HeadBands ?	OptionSpan	The following **CaseBinding** choice specifies the use of headbands on a case bound book. If *true*, headbands are inserted both top and bottom. Default = *false*.
Shape ?	EnumerationSpan	Indicates the shape of the "back" or spine of a Casebound book. Possible values are: *RoundedBack* *SquareBack*
Thickness ?	NumberSpan	Specifies thickness of board which is wrapped as front and back covers of a case bound book, in points.

Structure of the ChannelBinding Subelement.

Name	Data Type	Description
Cover ?	OptionSpan	If *true*, the clamp used in **ChannelBinding** includes a preassembled cover. Default = *false*
Thickness ?	NumberSpan	Specifies thickness of board which is wrapped as front and back covers of a Case bound book, in points. Default = system specified.

Structure of the CoilBinding Subelement.

Name	Data Type	Description
CoilMaterial ?	EnumerationSpan	The coil materials available for **CoilBinding**. Possible values are: *Steel* – plain steel *ColorCoatedSteel* – coated steel *Plastic* – plastic Default = system specified

Structure of the EdgeGluing Subelement.

New in JDF 1.1

Name	Data Type	Description
EdgeGlue ?	EnumerationSpan	Glue type used to glue the edge of the gathered sheets. Possible values are: *ColdGlue* *Hotmelt* *PUR* – Polyurethane Rubber Default = system specified

Structure of the HardCoverBinding Subelement. New in JDF 1.1

Name	Data Type	Description
BlockThreadSewing ?	OptionSpan	Option if the block is also thread sewn. Default = *false*

Name	Data Type	Description
EndSheets ?	OptionSpan	Option if end sheets are applied. Default = *true*
StripMaterial ?	EnumerationSpan	SpineTaping strip material. Possible values are: *Calico* *Cardboard* *CrepePaper* *Gauze* *Paper* *PaperlinedMules* *Tape*
HeadBands ?	OptionSpan	The following **CaseBinding** choice specifies the use of headbands on a case bound book. If *true*, headbands are inserted both top and bottom. Default = *false*.
HeadBandColor ?	EnumerationSpan	Defines the color of the headband. Allowed values are defined in Appendix A.2.8 NamedColor.
Jacket ?	EnumerationSpan	Specifies whether a hard cover jacket is needed and how it is attached. If specified, details of the jacket are described in the **Component** with **ProcessUsage** of *Jacket.* Possible values: *None*: No jacket is required. *Loose*: The jacket is loosely wrapped. *Glue*: Jacket is glued to the spine Default = *None*
JapanBind ?	OptionSpan	Bind the book block at the open edge, so that the folds are visible on the outside. Default = *false*.
SpineBrushing ?	OptionSpan	Brushing option for **SpinePreparation.**
SpineFiberRoughing ?	OptionSpan	FiberRoughing option for **SpinePreparation.**
SpineGlue ?	EnumerationSpan	Glue type used to glue the book block to the cover. Possible values are: *ColdGlue* *Hotmelt* *PUR* – Polyurethane Rubber
SpineLevelling ?	OptionSpan	Leveling option for **SpinePreparation.**
SpineMilling ?	OptionSpan	Milling option for **SpinePreparation.**
SpineNotching ?	OptionSpan	Notching option for **SpinePreparation.**
SpineSanding ?	OptionSpan	Sanding option for **SpinePreparation.**
SpineShredding ?	OptionSpan	Shredding option for **SpinePreparation.**
Thickness ?	NumberSpan	Specifies thickness of board which is wrapped as front and back covers of a case bound book, in points.

Name	Data Type	Description
TightBacking ?	EnumerationSpan	Definition of the geometry of the back of the book block. This can be one of: *Flat:* The default *Round*: rounding way, *FlatBacked*: backing way, *RoundBacked*, rounding way, backing way.
RegisterRibbon*	refelement	Number, materials, colors and details of register ribbons.

Structure of the PlasticCombBinding Subelement.

Name	Data Type	Description
PlasticCombType ? Modified in JDF 1.1	NameSpan	The distance between the "teeth" in ***PlasticCombBinding*** and the distance between the holes of the prepunched sheets must be the same. The following values from the hole type catalog in Appendix L exist: *P12m-rect-02*: Distance = 12 mm; Holes = 7 mm x 3 mm *P16_9i-rect-0t*: Distance = 14.28 mm; Holes = 8 mm x 3 mm The following values are deprecated in JDF 1.1. *Euro* – Distance = 12 mm; Holes = 7 mm x 3 mm *USA1* – Distance = 14.28 mm; Holes = 8 mm x 3 mm Default = system specified

Structure of the RingBinding Subelement.

Name	Data Type	Description
BinderMaterial ?	NameSpan	The following describe ***RingBinding*** binder materials used. Values include: *Cardboard* – Cardboard with no covering. *ClothCovered* – Cardboard with cloth covering. *Plastic* – Binder cover fabricated from solid plastic sheet material, e.g., PVC sheet. *VinylCovered* – Cardboard with colored vinyl covering. Default = system specified
HoleType ? New in JDF 1.1	Enumeration-Span	Predefined hole pattern for the ring system. Multiple hole patterns are not allowed, e.g., 3-hole ring binding and 4-hole ring binding holes on one piece of media. For details of the hole types, refer to Appendix L JDF/CIP4 Hole Pattern Catalog. Allowed values include:

Name	Data Type	Description	
		R2-generic	*R5-generic*
		R2m-DIN	*R5i-US-a*
		R2m-ISO	*R5i-US-b*
		R2i-US-a	*R5i-US-c*
		R2i-US-b	*R6-generic*
		R3-generic.	*R6m-4h2s*
		R3i-US	*R6m-DIN-A5*
		R4-generic	*R7-generic*
		R4m-DIN-A4	*R7i-US-a*
		R4m-DIN-A5	*R7i-US-b*
		R4m-swedish	*R7i-US-c*
		R4i-US	*R11m-7h4s*
RingDiameter ?	NumberSpan	Size of the rings in points. Default = system specified, but suitable for specified *HoleType* (s). Note: In ring shapes other than round, this size is specified by industry-standard method.	
RingMechanic ?	OptionSpan	The ring binder used includes a lever for opening and closing. Default = *false*	
RingShape ?	NameSpan	The following **RingBinding** shapes are used: *Round:* the default. *Oval* *D-shape* *SlantD*	
RingSystem Deprecated in JDF 1.1	NameSpan	The following values are deprecated from JDF 1.1 *2HoleEuro* *3HoleUS* *4HoleEuro* These have been replaced by *HoleType*.	
RivetsExposed ?	OptionSpan	The following **RingBinding** choice describes mounting of the ring mechanism in binder case. If *true*, the heads of the rivets are visible on the exterior of the binder. If *false*, the binder covering material covers the rivet heads. Default = *true*.	
ViewBinder ?	NameSpan	The following **RingBinding** clear vinyl outer wrap types are used on top of a colored base wrap: *Embedded* – Printed material is embedded by sealing between the colored and clear vinyl layers during the binder manufacturing. *Pocket* – Binder is designed so that Printed material may be inserted between the color and clear vinyl layers after the binder is manufactured. Default = *Pocket*	

Structure of the SaddleStitching Subelement.

Name	Data Type	Description
StitchNumber ? `New in JDF 1.1`	IntegerSpan	Number of stitches used for saddle stitching. Default = system specified.

Structure of the SideSewing Subelement.

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description

Structure of the SideStitching Subelement.

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description

Structure of the SoftCoverBinding Subelement.

`New in JDF 1.1`

Name	Data Type	Description
BlockThreadSewing ?	OptionSpan	Specifies whether the block is also thread sewn. Default = *false*
GlueProcedure ?	Enumeration-Span	Glue procedure used to glue the book block to the cover. Possible values are: *Spine:* *SideOnly:* Glued at the side/endsheets but not the spine. *SingleSide:* Swiss Brochure *SideSpine*: Both side gluing and SpineGluing. The default.
Scoring ?	Enumeration-Span	Scoring option for **SoftCoverBinding**. Possible values are: *TwiceScored* *QuadScored* *None* Values are based on viewing the cover in its flat prebinding state.
SpineBrushing ?	OptionSpan	Brushing option for **SpinePreparation.**
SpineFiberRoughing ?	OptionSpan	FiberRoughing option for **SpinePreparation.**
SpineGlue ?	Enumeration-Span	Glue type used to glue the book block to the cover. Possible values are: *ColdGlue* *Hotmelt* *PUR* – Polyurethane Rubber
SpineLevelling ?	OptionSpan	Leveling option for **SpinePreparation.**
SpineMilling ?	OptionSpan	Milling option for **SpinePreparation.**
SpineNotching ?	OptionSpan	Notching option for **SpinePreparation.**
SpineSanding ?	OptionSpan	Sanding option for **SpinePreparation.**
SpineShredding ?	OptionSpan	Shredding option for **SpinePreparation.**

Structure of the Tape Subelement.

`New in JDF 1.1`

Name	Data Type	Description
TapeColor ?	Enumeration-Span	Defines the color of the tape material of the binding. Allowed values are defined in Appendix A.2.8 NamedColor. Default = don't care, i.e., system specified

Structure of the Tabs Subelement.

Specifies tabs.

Name	Data Type	Description
TabBanks ?	Integer	Number of rows of tabs on the face of the book. Default = 1
TabsPerBank ?	Integer	Number of equal-sized tabs in a single bank, if all positions were filled. Default = don't care, i.e., system specified. Note: Banks may have tabs only in some of the possible positions
TabExtensionDistance ?	NumberSpan	Distance tab extends beyond the body of the book block, in points. Default = system specified
TabExtensionMylar ?	OptionSpan	If *true*, the tab extension will be mylar reinforced Default = *false*
TabBindMylar ?	OptionSpan	If *true*, the tab bind edge will be mylar reinforced Default = *false*
TabBodyCopy ?	OptionSpan	If *true*, Color will be applied not only on tab extension, but also on tab body. Note: Lack of body copy allows all tabs within a bank to be printed on a single sheet. Default = *false*
TabMylarColor ?	Enumeration-Span	Specifies the color of the mylar used to reinforce the tab extension. This is conditional on TabExtensionMylar being *true*. Allowed values are defined in Appendix A.2.8 NamedColor. Default = don't care, i.e., system specified

Structure of the ThreadSealing Subelement.

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description

Structure of the ThreadSewing Subelement.

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description
Sealing ?	OptionSpan	If *true*, thermo-sealing is required in **ThreadSewing**.

Structure of the StripBinding Subelement.

New in JDF 1.1

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description

Structure of the VeloBinding Subelement.

Deprecated in JDF 1.1

This is a placeholder that may be filled with private or future data.

Name	Data Type	Description

Structure of the WireCombBinding Subelement.

Name	Data Type	Description
WireCombMaterial ?	Enumeration-Span	The material used for forming the **WireCombBinding**. Possible values are: *Steel*-Silver – The default if **BindingColor** is specified as *silver*, otherwise **ColorCoatedSteel**. *ColorCoatedSteel*

Name	Data Type	Description
WireCombShape ?	Enumeration-Span	The shape of the **WireCombBinding.** Possible values are: *Single* – Each "tooth" is made with one wire *Twin* – The shape of each "tooth" is made with a double wire, e.g., Wire-O. Default = system specified

7.1.4 ColorIntent

This resource specifies the type of ink to be used. Typically, the parameters consist of a manufacturer name and additional identifying information. The resource also specifies any coatings and colors to be used, including the process color model and any spot colors.

Resource Properties

Resource class: Intent
Example Partition: *Option, PageNumber, Side*
Resource referenced by: -
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
Coatings ? Modified in JDF 1.1	StringSpan	Material usually applied to a full surface on press as a protective or gloss enhancing layer over ink. Possible values include: *DullVarnish* *GlossVarnish* *UV* *Aqueous* *Silicone* The individual strings within **Coatings** are of type NMTOKENS and may contain multiple entries from the above list.
ColorStandard ?	NameSpan	The color process standard requested for the job. Possible values include: *CMYK – G*eneric four color process. *FIRST* – Flexographic Image Reproduction Specifications & Tolerances. *GRACOL* – General Requirements for Applications in Commercial Offset Lithography *Hexachrome* – 6 Colors CMYK+Orange and Green. *HIFI* – 7 Colors CMYK+Red, Green and Blue. *ISO12647* – ISO offset standard. *Monochrome* – Black and white. *None* – No marks. Used to define one-sided printing. *SNAP* – Specifications for Newsprint Advertising Production *SWOP* – Specifications for Web Offset Publications

Name	Data Type	Description
Coverage ?	NumberSpan	Cumulative colorant coverage percentage. For example, a full sheet of 100% deep black in CMYK has *Coverage = "400".* Typical coverages based on one color plane are: *Light = 1-9%* *Medium= 10-35%* *Heavy= 36+%*
InkManufacturer ?	NameSpan	Name of the manufacturer of the ink requested, e.g.: *Toyo* *Sun* *Gans*
ColorPool ? New in JDF 1.1	refelement	Additional details about the colors used.
ColorsUsed ?	element	Array of color separation names that are requested. If not specified, the values are implied from *ColorStandard*. If additional information about the colors is required, it can be specified in the referenced **ColorPool** resource.

Structure of the ColorsUsed Subelement

Name	Data Type	Description
SeparationSpec*	element	These can be process colors, generic spot colors or specific colors. In addition, partial coating is specified by adding a Separation-Spec with anything from *Coatings* as *Name*: *DullVarnish* *GlossVarnish* *UV* *Aqueous* *Bronzing* *Silicone*

7.1.5 DeliveryIntent

Summarizes the options that describe pickup or delivery time and location options of a job. It also defines the number of copies that are requested for a specific job or delivery. This includes delivery of both final products and of proofs.

Resource Properties
Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
Accepted ?	boolean	The quote that is specified by this **DeliveryIntent** has been accepted. Default = *false*
BuyerAccount ?	string	Account ID of the buyer with the delivery service.

Name	Data Type	Description
DeliveryCharge ? New in JDF 1.1	Enumeration-Span	Specifies who pays for a delivery being made by a 3rd party. Possible values are: Printer Default = *Buyer*
Earliest ?	TimeSpan	Specifies the earliest time after which the transfer may be made.
EarliestDuration ?	DurationSpan	Specifies the earliest time by which the transfer must be made relative to the date of the purchase order. Within an RFQ or a Quote, only one of either *Earliest* or *EarliestDuration* may be specified. Within a purchase order only the *Earliest* is allowed.
Method ?	NameSpan	Identifies a required delivery method, may be a generic item from the following list: *BestWay* – The sender decides how to deliver. *CompanyTruck* *Courier* *Email* *ExpressMail* *InterofficeMail* *Storage* – The product must be stored by the supplier. *OvernightService* Unknown May also be a delivery service brand. For example: *UPS* *DHL* *FedEx*
Ownership ?	enumeration	If *Origin* (default), then ownership of goods is transferred upon leaving point of origin. If *Destination*, ownership is transferred upon receipt at destination.
Overage ?	NumberSpan	Percentage value that defines the acceptable upwards variation of *Amount*. Defaults to the trade custom defaults as defined by PIA, BVD etc.
Pickup ? Deprecated in JDF 1.1	boolean	Specifies whether the delivery brings or picks up the merchandise. Default = *false*, which means that the drop is delivered to the address specified in **Company**. If *Pickup = true*, the **DeliveryIntent** describes an input to the job, e.g., a CD for inserting, a preprinted cover, etc. In this case **Company** describes the location where the merchandise is picked up.
Required ?	TimeSpan	Specifies the time by which the transfer must be made.
RequiredDuration ?	DurationSpan	Specifies the time by which the transfer must be made relative to the date of the purchase order. Within an RFQ or a Quote only one of either *Required* or *RequiredDuration* must be specified. Within a purchase order only *Required* is allowed.
ReturnMethod ? New in JDF 1.1	NameSpan	Identifies a required delivery method for returning the surplus material, if *SurplusHandling* = *Return*. The values may be of the same list as specified in *Method*.

Name	Data Type	Description
SurplusHandling ? New in JDF 1.1	Enumeration-Span	Describes what should happen with unused or redundant parts of the transfer specified with *Transfer* = *BuyerToPrinterDeliver* or *BuyerToPrinterPickup* after the job. The return delivery or pickup address is specified in the Contact with *ContactTypes* including *SurplusReturn*. Possible values are: *ReturnWithProduct* – The surplus material is delivered back to the customer together with the final product. *Return* – The surplus material is delivered back independently directly after usage. *Pickup* – The customer picks up the surplus material *Destroy* – The printer must destroy the surplus material *PrinterOwns* – The surplus material belongs to the printer *Store* – The printer has to store the surplus material for future purposes Default = ReturnWithProduct.
Transfer ? New in JDF 1.1	EnumerationSpan	Describes the direction and responsibility of the transfer. Possible values are: *BuyerToPrinterDeliver* – The **DeliveryIntent** describes an input to the job, e.g., a CD for inserting, a preprinted cover, etc. In this case, the buyer delivers the merchandise to the printer. The printer may specify in the quote a special **Contact** with *ContactTypes* including "*Delivery*", where the buyer should send the merchandise to. *BuyerToPrinterPickup* – The **DeliveryIntent** describes an input to the job, e.g., a CD for inserting, a preprinted cover, etc. In this case, the printer picks up the merchandise. The **Contact** with *ContactTypes* including *pickup* describes, where the printer has to pick up the merchandise. *PrinterToBuyerDeliver* – The **DeliveryIntent** describes an output of the job. In this, case the printer delivers the merchandise to the buyer. The **Contact** that has *ContactTypes* including "*Delivery*" specifies where the printer should send the merchandise. *PrinterToBuyerPickup* – the **DeliveryIntent** describes an output of the job. In this case, the buyer picks up the merchandise. The printer may specify in the quote a special **Contact** that has *ContactTypes* including "Pickup", where the buyer should pick up the merchandise.
Underage ?	NumberSpan	Percentage value that defines the acceptable downwards variation of *Amount*. Defaults to the trade custom defaults as defined by PIA, BVD etc.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the addressee.

Name	Data Type	Description
Contact * New in JDF 1.1	refelement	Address and further information of the **Contact** responsible for the transfer. The actual delivery address is specified as the Address of the **Contact** with *ContactTypes* that includes "*Delivery*". The actual pickup address is specified as the Address of the **Contact** with *ContactTypes* that includes "*Pickup*". For each of the values "*Delivery*", "*Pickup*", and "*Billing*" only one **Contact** with *ContactTypes* including these values may be specified.
DropIntent +	element	Includes all locations where the product will be delivered. Note that multiple DropIntents specify multiple deliveries and not options for delivery.
Pricing ?	element	Pricing elements that define the pricing of the complete **DeliveryIntent** including any DropIntents or DropItemIntents that may contain further Pricing elements.

Structure of DeliveryIntent Elements

DropIntent

This element contains information about the intended individual drop of a delivery. Attributes that are specified in a DropIntent element overwrite those that are specified in their parent DeliveryIntent element. If optional values are not specified, they default to the values specified in the DeliveryIntent.

Name	Data Type	Description
Earliest ?	TimeSpan	Specifies the earliest time after which the transfer may be made.
EarliestDuration ?	DurationSpan	Specifies the earliest time by which the transfer must be made relative to the date of the purchase order. Within an RFQ or a Quote, only one of either *Earliest* or *EarliestDuration* may be specified. Within a purchase order only the *Earliest* is allowed.
Method ?	NameSpan	Identifies a required delivery method. The values are identical to those of *Method* in the DeliveryIntent root. Defaults to the value of *Method* in **DeliveryIntent**.
Pickup ? Deprecated in JDF 1.1	boolean	If *true*, the merchandise is picked up. If *false*, the merchandise is delivered. Default = *false*, which means that the **DropIntent** is delivered to the address specified in **Company**. If *Pickup = true*, the **DropIntent** describes an input to the job, e.g., a CD for inserting, a preprinted cover, etc. In this case, **Company** describes the location where the merchandise is picked up.
Required ?	TimeSpan	Specifies the time by which the delivery must be made.
RequiredDuration ?	DurationSpan	Specifies the time by which the delivery must be made relative to the date of the purchase order. Within an RFQ or a Quote only, one of either *Required* or *RequiredDuration* must be specified. Within a purchase order only *Required* is allowed.

Name	Data Type	Description
ReturnMethod ? New in JDF 1.1	NameSpan	Identifies a required delivery method for returning the surplus material, if *SurplusHandling = Return*. Defaults to the value of *ReturnMethod* in **DeliveryIntent**.
SurplusHandling ? New in JDF 1.1	Enumeration-Span	Describes what should happen with unused or redundant parts of the transfer. The values are identical to those of *SurplusHandling* in **DeliveryIntent**. Defaults to the value of *SurplusHandling* in **DeliveryIntent**.
Transfer ? New in JDF 1.1	Enumeration-Span	Describes the direction and responsibility of the transfer. The values are identical to those of *Transfer* in **DeliveryIntent**. Defaults to the value of *Transfer* in **DeliveryIntent**.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the addressee. Defaults to the Company specified in the parent resource.
Contact * New in JDF 1.1	refelement	Address and further information of the **Contact** responsible for the transfer. The actual delivery address is specified as the Address of the **Contact** with *ContactTypes* that includes "*Delivery*". The actual pickup address is specified as the Address of the **Contact** with *ContactTypes* that includes "*Pickup*". For each of the values "*Delivery*", "*Pickup*", and "*Billing*", only one Contact with *ContactTypes* including these values may be specified. Defaults to the **Contact** specified in the parent resource.
DropItemIntent +	element	A DropIntent may consist of multiple products, which are represented by their respective **Component** resources. Each DropItemIntent element describes a number of individual resources that is part of this DropIntent.
Pricing ?	element	Pricing element that defines the pricing of the DropIntent.

Structure of the DropItemIntent Subelement

Name	Data Type	Description
AdditionalAmount ?	integer	Number of components used to calculate the value of the *AdditionalPrice* attribute in the Pricing. Default = 1.
Amount ?	integer	Specifies the final number of components delivered. If not specified, defaults to the total amount of the resource that is referenced by Component or 1 if this DropItemIntent specifies a proof.
OrderedAmount ?	integer	Specifies the original number of components ordered. If not specified, Default = *Amount*.
Proof ? New in JDF 1.1	string	This DropItem refers to a proof that is specified in a ProofItem of the **ProofingIntent** of this product node. The *ProofName* attribute of a ProofItem must match *Proof*. One of either Component or *Proof* must be specified.
Unit ?	string	Unit of measurement for the *Amount* specified in the *Component* attribute. Defaults to the value of *Unit* defined in the resource described by the Component.
PhysicalResource? Modified in JDF 1.1	refelement	Description of the individual item that is delivered. One of either PhysicalResource or *Proof* must be specified. Note that PhysicalResource is an abstract resource and that the element must be an instance of PhysicalResource.
Pricing ?	element	Pricing element that defines the pricing of the DropItemIntent.

Contents of the Pricing Subelement

Name	Data Type	Description
AdditionalPrice ?	number	Price for ordering the number of copies specified in the *AdditionalAmount* attribute as specified in the parent element of the Pricing.
Currency ?	NMTOKEN	Three digit currency definition according to ISO 4217. It defaults to the currency defined in the parent quote.
HasPrice ?	boolean	Specifies whether the line item defined by this quote has a price. If *false*, the line item is not included in the parent quote, and the price is unknown and must be added. Default = *true*, i.e., the line item is included in the parent quote.
Item ?	string	Name of the item that this particular quote element describes. Default = everything
Price ?	number	Price for ordering the number of copies specified in the *Amount* attribute as specified in the parent element of the Pricing. If not specified, it defaults to the sum of prices of the direct child Pricing elements.
Payment ? New in JDF 1.1	element	Details of the payment method.
Pricing *	element	Individual items of the quote. Note that a parent quote defines the complete quote, i.e., including the values defined in the line items of any child quotes but excluding all line items with *HasPrice* = "*false*". The sum of line items need not be identical to the parent quote.

Contents of the Payment Subelement

New in JDF 1.1

Name	Data Type	Description
PayTerm ?	telem	Describes the payment terms & conditions.
CreditCard ?	element	Specifies credit card information

Contents of the CreditCard Subelement

New in JDF 1.1

Name	Data Type	Description
Authorization ?	String	Authorization code for this transaction.
AuthorizationExpires ?	gYearMonth	Expiration date of the *Authorization*.
Expires	gYearMonth	Expiration date of the credit card.
Number	NMTOKEN	Credit card number. The format is specified without blanks or any other separator characters.
Type	NMTOKEN	Credit card brand. Possible values include: Amex DinersClub Discovery *MasterCard*: This includes derived brands, e.g., EuroCard Visa

7.1.6 EmbossingIntent

New in JDF 1.1

This resource specifies the embossing and/or foil stamping intent for a JDF job using information that identifies whether or not the product is embossed or stamped and, if desired, the complexity of the affected area.

Resource Properties

Resource class: Intent
Resource referenced by: -
Example Partition: *Option, PageNumber, Side*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
EmbossingItem +	refelement	Each embossed image is described by one EmbossingItem.

Structure of the EmbossingItem Subelement

Name	Data Type	Description
Direction	EnumerationSpan	The direction of the image. Possible values are: *Both* – Both debossing and embossing in one stamp. *Depressed* – Debossing *Raised* – Embossing
EdgeAngle ?	NumberSpan	The angle of a beveled edge in degrees. Typical values are an angle of: 30, 40, 45, 50, or 60 degrees. For *EdgeAngle* to exist, *EdgeShape = Beveled* must be specified.
EdgeShape ?	EnumerationSpan	The transition between the embossed surface and the surrounding media may be rounded or beveled (angled). Possible values are: *Rounded* *Beveled*
EmbossingType	StringSpan	The strings defined in *EmbossingType* are whitespace separated combinations of the following tokens. Possible values for the tokens are: *BlindEmbossing – E*mbossed forms that are not inked or foiled. The color of the image is the same as the paper. *FoilEmbossing – C*ombines embossing with foil stamping in one single impression. *FoilStamping*-using a heated die to place a metallic or pigmented image from a coated foil on the paper. *RegisteredEmbossing – C*reates an embossed image that exactly registers to a printed image.
FoilColor ?	EnumerationSpan	Defines the color of the foil material which is used within the *FoilStamp* process. Allowed values are defined in the appendix A.2.8 NamedColor.
Height ?	NumberSpan	The height of the levels. This value specifies the *vertical* distance between the highest and lowest point of the stamp, regardless of the value of *Direction.*
ImageSize ?	XYPairSpan	The size of the bounding box of one single image.
Level ?	EnumerationSpan	The level of embossing. Possible values are: *SingleLevel*, *MultiLevel*, *Sculpted*
Position ?	XYPairSpan	Position of the center of the bounding box of the embossed image in the coordinate system of the Component.

7.1.7 FoldingIntent

This resource specifies the fold intent for a JDF job using information that identifies the number of folds, the height and width of the folds, and the folding catalog number. Note that the folding catalog is described in Section 7.2.55 and that the number of folds and the folding catalog are related.

Resource Properties

Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
FoldingCatalog	NameSpan	Description of the folding scheme as specified in the **FoldingParams** folding catalog attribute in the format "Fn-i". See JDF Folding Catalog descriptions in Figure 7.11 Fold Catalog part 1 and Figure 7.12 Fold Catalog part 2.
		Note: The folding scheme in this context refers to the folding of the finished product as seen after the cutting, not the folding, of the flat as seen in production.
Folds ? Deprecated in JDF 1.1	XYPair	Number of folds in x and in y direction. This attribute specifies the number of folds seen in the sheet after folding not the number of fold operations needed to achieve that result. If not specified, it must be inferred from *FoldingCatalog*. The product 2*(X+1)*(Y+1) of *Folds* must always match the n of "Fn-i" of *FoldingCatalog.*
Fold * New in JDF 1.1	element	This describes the details of folding operations in the sequence described by the value of *FoldingCatalog.* **Fold** must be specified if non-symmetrical folds are requested.

7.1.8 HoleMakingIntent

This resource specifies the holemaking intent for a JDF job, using information that identifies the type of HoleMaking operation or alternatively, an explicit list of holes.

Resource Properties

Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
HoleReferenceEdge ? New in JDF 1.1	enumeration	The edge of the media relative to where the holes should be punched. Use with HoleType. Possible values are:
		Left
		Right
		Top
		Bottom
		Pattern – Specifies that the reference edge implied by the value of *HoleType* in Appendix L JDF/CIP4 Hole Pattern Catalog is used. The default if *HoleType* is not *Explicit*, otherwise *Left*.

Name	Data Type	Description
HoleType Modified in JDF 1.1	StringSpan	Predefined hole pattern. Multiple hole patterns are specified as one NMTOKENS string, e.g., 3-hole ring binding and 4-hole ring binding holes on one piece of media. For details of the hole types, refer to Appendix L JDF/CIP4 Hole Pattern Catalog. Allowed values include: *R2-generic* *R6-generic* *R2m-DIN* *R6m-4h2s* *R2m-ISO* *R6m-DIN-A5* *R2i-US-a* *R7-generic* *R2i-US-b* *R7i-US-a* *R3-generic* *R7i-US-b* *R3i-US* *R7i-US-c* *R4-generic* *R11m-7h4s* *R4m-DIN-A4* *P12m-rect-0t* *R4m-DIN-A5* *P16_9i-rect-0t* *R4m-swedish* *W2_1i-round-0t* *R4i-US* *W2_1i-square-0t* *R5-generic* *W3_1i-square-0t* *R5i-US-a* *C9.5m-round-0t* *R5i-US-b* *Explicit* – Holes are defined in an array of **Hole** elements. *R5i-US-c* The following values are deprecated from JDF 1.0 *2HoleEuro* – Replace by either R2m-DIN or R2m-ISO. *3HoleUS* – Replace by R3I-US *4HoleEuro* – Replace by R4m-DIN-A4 or R4m-DIN-A5.
HoleList ?	element	Array of all **Hole** elements. Used when *HoleType* = *Explicit*. Default = no holes

Structure of the HoleList Subelement

Name	Data Type	Description
Hole * Modified in JDF 1.1	refelement	Description of individual holes. See 7.2.64 Hole.
HoleLine * New in JDF 1.1	refelement	Array of all **HoleLine** elements. See 7.2.65 HoleLine.

7.1.9 InsertingIntent

This resource specifies the placing or inserting of one component within another, using information that identifies page location, position and attachment method. The receiving component is defined by a *ProcessUsage* attribute of *"Parent"*. All other input components are mapped to the **Insert** elements by their ordering in the **ResourceLinkPool**.

Resource Properties
Resource class: Intent
Resource referenced by: -
Example Partition: *Option*

Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
GlueType ?	Enumeration-Span	Glue used to fasten the insert. Possible values are: *Permanent* *Removable* Default = system specified
InsertList	Element	List of individual inserts.
Method	Enumeration-Span	Possible values are: *BindIn* – Apply glue to fasten the insert *BlowIn* – Loose insert. The default.

Structure of InsertList Subelement

Name	Data Type	Description
Insert *	element	Individual insert description.

Structure of Insert Subelement

Name	Data Type	Description
Folio	IntegerRangeList	List of potential folios where the insert is to be placed. A **Folio** is defined by its first page in case **Method** = *BlowIn* and by the page that the glue is applied in case **Method** = *BindIn*. In general, a list of folios will only be supplied for **Method** = *BlowIn*. The pages are counted in the order, which is described in **FolioCount** of the parent **Component**.
GlueType ?	EnumerationSpan	Glue used to fasten the insert. Possible values are: *Removable* *Permanent* Defaults to the **GlueType** specified in the parent resource.
Method ?	EnumerationSpan	Inserting method. Possible values are: *BindIn* – Apply glue to fasten the insert *BlowIn* – Loose insert Defaults to the **Method** specified in the parent resource.
SheetOffset ? Deprecated in JDF 1.1	XYPair	Offset between the **Component** to be inserted and page in the parent **Component**.
Transformation ?	matrix	Rotation and offset between the **Component** to be inserted and the parent **Component**. For details on transformations, see How and Where Coordinates and Transformations Are Used/Defined in JDF. Default = *identity*
WrapPages ? New in JDF 1.1	IntegerRangeList	List of pages of the Cover that wrap around an Insert after all folds are correctly positioned. It is sufficient to specify the pages of the Front surface of the cover. Note that this key must only be specified if the folding is ambiguous. Default = empty list.

Name	Data Type	Description
GlueLine * New in JDF 1.1	element	Array of all GlueLine elements used to glue in the Insert. Must not be specified in conjunction with *GlueType*.

7.1.10 LaminatingIntent

This resource specifies the laminating intent for a JDF job using information that identifies whether or not the product is laminated and, if desired, the temperature and thickness of the laminant.

Resource Properties

Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
Laminated Deprecated in JDF 1.1	OptionSpan	If *true*, the product is laminated. Default = *false*
Temperature	Enumeration-Span	Temperature used in the lamination process. Possible values are: *Hot* *Cold*
Surface ?	Enumeration-Span	The surface to be laminated. One of: *Front* *Back* *Both* Default = system specified
Thickness ?	NumberSpan	Thickness of the laminating material. Measured in micron[μm]. Default = system specified

7.1.11 LayoutIntent

This resource records the size of the finished pages for the product component. It does not, however, specify the size of any intermediate results, such as press sheets. It also describes how the pages of the product component should be imaged onto the finished media. The size definition of the finished media describes the size of a sheet that is folded to create a product, not the size of a production sheet, e.g., in the press.

Resource Properties

Resource class: Intent
Example Partition: *Option*
Resource referenced by: -
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
Dimensions ? New in JDF 1.1	XYPairSpan	Specifies the width (X) and height (Y), respectively, of the media or product **Component** before folding in pts.
FinishedDimensions ? New in JDF 1.1	ShapeSpan	Specifies the height width (X), width height (Y) and depth (Z), respectively, of the finished product **Component** after folding in points. If the Z coordinate is 0, it is ignored.
FinishedPageOrientation ? Deprecated in JDF 1.1	enumeration	Indicates the desired orientation of the finished Media. Possible values are: *Portrait* – The short edges of the media are the top and bottom. *Landscape* – The long edges of the media are the top and bottom. Default = *Portrait*. In JDF 1.1, the page orientation is implied by the value of *Dimensions* and *FinishedDimensions.* If height (X) > width (Y), the product is portrait.
FolioCount ? New in JDF 1.1	enumeration	Defines the method used when counting page folios. The number of pages of one sheet of an individual component is given by the product $2*(X+1)*(Y+1)$, where x denotes the number of folds in x direction and y denotes the number of folds in y direction. One of: *Booklet*: Each sample of the component consists of two pages with no fold inside the page (the front side and the back side of one sample of the component). The pages are counted in reader order of the pages of the component in the product. The default. *Flat*: The pages are counted from the top left of the front side of the top media to the bottom right of the back side of the bottom media. *Flat* should be used for non-standard products where the reader order is ambiguous. The page breaks on a sheet are defined by the folds as specified by *FoldingCatalog* (see Figure 7.11 and Figure 7.12) in the **FoldingIntent** for the product. All sheets are counted, even if they are not included in the product, e.g., due to a **ShapeCuttingIntent**.
NumberUp ?	XYPair	Specifies a regular, multi-up grid of page cells into which content pages are mapped. The first value specifies the number of rows of page cells and the second value specifies the number of columns of page cells in the multi-up grid. Default = 1 1, i.e., 1 document page per side.
Pages ? New in JDF 1.1	IntegerSpan	Specifies the number of page sheets of the product component. *Pages* multiplied with *Dimensions* always defines the amount of paper that is used in the product. *Pages* describes the paper usage regardless of document layout. Default = 1
PageVariance ? New in JDF 1.1	IntegerSpan	Specifies the number of non-identical page sheets of the product component. If not specified, Default = value of *Pages.*

Name	Data Type	Description
Sides ?	enumeration	Indicates whether contents should be printed on one or both sides of the media. Possible values are:
		OneSided – Page contents will only be imaged on one side of the media.
		TwoSidedHeadToHead – Impose pages upon the front and back sides of media sheets so that the head (top) of page contents back up to each other.
		TwoSidedHeadToFoot – Impose pages upon the front and back sides of media sheets so that the head (top) of the front backs up to the foot (bottom) of the back.
Layout ? New in JDF 1.1	refelement	Specifies the details of a more complex **Layout**. Must not be specified together with *NumberUp*.

7.1.12 MediaIntent

This resource describes the media to be used for the product component. In some cases, the exact identity of the medium is known, while in other cases, the characteristics are described and a particular stock is matched to those characteristics.

Resource Properties

Resource class: Intent
Example Partition: *Option*
Resource referenced by: -
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
BackCoatings ?	Enumeration-Span	Identical to *FrontCoatings*, but applied to the back surface of the media.
		Default = value of *FrontCoatings*.
Brightness ?	NumberSpan	reflectance percentage.
BuyerSupplied ?	OptionSpan	Indicates whether the customer will supply the media.
Dimensions ?	XYPairSpan	Specifies the size of the media in points.
FrontCoatings ?	Enumeration-Span	What preprocess coating has been applied to the front surface of the media. Possible values are:
		None: the default.
		Glossy
		HighGloss
		Matte
		Satin
		Semigloss
Grade ?	IntegerSpan	The intended grade of the media on a scale of 1 through 5.
		Grade of paper material is defined in accordance with ISO 12647-2 ff. Offset printing grades:
		1 "Gloss-coated"
		2 "Matt-coated"
		3 "Gloss-coated, web"

Name	Data Type	Description
		4 "Uncoated, white"
		5 "Uncoated, yellowish"
HoleCount ? Deprecated in JDF 1.1	IntegerSpan	The intended number of holes that should be punched in the media (either pre- or post-punched). Default = 0. In JDF/1.1, use *HoleType* which includes the number of holes.
HoleType ? New in JDF 1.1	StringSpan	Predefined hole pattern that specifies the prepunched holes in the media. If custom holes are required, or the hole manufacturing method (prepunched or post-punched) is "don't care," this must be specified in HoleMakingIntent. Multiple hole patterns are specified as one NMTOKENS string, e.g, 3-hole ring binding and 4-hole ring binding holes on one piece of media. For details of the hole types, refer to Appendix L JDF/CIP4 Hole Pattern Catalog. Allowed values include: *None:* The default. / *R5i-US-c* *R2-generic* / *R6-generic* *R2m-DIN* / *R6m-4h2s* *R2m-ISO* / *R6m-DIN-A5* *R2i-US-a* / *R7-generic* *R2i-US-b* / *R7i-US-a* *R3-generic* / *R7i-US-b* *R3i-US* / *R7i-US-c* *R4-generic* / *R11m-7h4s* *R4m-DIN-A4* / *P12m-rect-0t* *R4m-DIN-A5* / *P16_9i-rect-0t* *R4m-swedish* / *W2_1i-round-0t* *R4i-US* / *W2_1i-square-0t* *R5-generic* / *W3_1i-square-0t* *R5i-US-a* / *C9.5m-round-0t* *R5i-US-b*
MediaColor ?	Enumeration-Span	Color of the media. Allowed values are defined in Appendix A.2.8 NamedColor.
MediaSetCount ?	integer	When the input media is grouped in sets, identifies the number of pieces of media in each set. For example, if the *UserMediaType* is "*PreCutTabs*", a *MediaSetCount* of 5 would indicate that each set includes 5 tab sheets.
MediaType ? New in JDF 1.1	Enumeration-Span	Describes the medium being employed. Possible values are: *Paper:* the default. *Transparency*
MediaUnit ?	Enumeration-Span	Describes the format of the media as it is delivered to the device. Possible values are: *Roll* *Sheet*
Opacity ?	Enumeration-Span	The opacity of the media. Possible values are: Opaque – the media is opaque. The default.

Name	Data Type	Description
		Transparent – the media is transparent
PrePrinted ?	boolean	Indicates whether the media is preprinted. Default = *false*
Recycled ?	OptionSpan	If *true*, recycled media is requested.
StockBrand ?	StringSpan	Strings providing available brand names. The customer may know exactly what paper is to be used. Example is "Lustro" or "Warren Lustro" even though the manufacturer name is included.
StockType ?	NameSpan	Strings describing the available stock. Examples include: *Bristol* *Cove,* *Bond* *Newsprint* *Index* *Offset* – This includes book stock. *Tag* *Text*
Texture ?	NameSpan	The intended texture of the media. Examples include: *Antique* – Rougher than vellum surface *Calendared* – Extra-smooth or polished uncoated paper *Linen* – Texture of coarse woven cloth *Smooth* *Stipple* – Fine pebble finish *Vellum* – Slightly rough surface
Thickness ? New in JDF 1.1	NumberSpan	The thickness of the chosen medium. Measured in micron [µm].
UserMediaType ?	NMTOKEN	A human-readable description of the type of media. The value can be used by an operator to select the correct media to load. The semantics of the values will be site-specific. Possible values include: *Continuous* – Continuously connected sheets of an opaque material. Which edge is connected is not specified. *ContinuousLong* – Continuously connected sheets of an opaque material connected along the long edge. *ContinuousShort* – Continuously connected sheets of an opaque material connected along the short edge. *Envelope* – Envelopes that can be used for conventional mailing purposes. *EnvelopePlain* – Envelopes that are not preprinted and have no windows. *EnvelopeWindow* – Envelopes that have windows for addressing purposes. *FullCutTabs* – Media with a tab that runs the full length of the medium so that only one tab is visible extending out beyond the edge of non-tabbed media. *Labels* – Label stock, e.g., a sheet of peel-off labels. *Letterhead* – Separately cut sheets of an opaque material includ-

Name	Data Type	Description
		ing a letterhead.
		MultiLayer – Form medium composed of multiple layers which are preattached to one another, e.g., for use with impact printers.
		MultiPartForm – Form medium composed of multiple layers not preattached to one another; each sheet may be drawn separately from an input source.
		Photographic – Separately cut sheets of an opaque material to produce photographic quality images.
		PreCutTabs – Media with tabs that are cut so that more than one tab is visible extending out beyond the edge of non-tabbed media.
		Stationery – Separately cut sheets of an opaque material.
		TabStock – Media with tabs (either precut or full-cut).
		Transparency – Separately cut sheets of a transparent material.
USWeight ?	NumberSpan	The intended weight of the media, measured in pounds per ream of basis size. Only one of **Weight** and **USWeight** may be specified. If known, **Weight** should be specified in g/m2.
Weight ?	NumberSpan	The intended weight of the media, measured in (g/m2). Only one of **Weight** and **USWeight** may be specified.

7.1.13 NumberingIntent

This resource describes the parameters of stamping or applying variable marks in order to produce unique components, for items such as lottery notes or currency.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: See Laminating.
Numbering
Output of processes: -

Resource Structure

Name	Data Type	Description
ColorName ?	EnumerationSpan	Defines the color of the numbering. Allowed values are defined in Appendix A.2.8 NamedColor. Default = don't care, i.e., system specified.
ColorPool ?	refelement	Additional details about the colors used.
NumberItem +	element	Individual position of the numbers on the finished page.

Structure of NumberItem Subelement

Name	Data Type	Description
ColorName ?	Enumeration-Span	Defines the color of the numbering. Allowed values are defined in Appendix A.2.8 NamedColor. If not specified, it defaults to the values defined in **NumberingIntent**.
StartValue ?	string	First value of the numbering machine. Default = 1, i.e., system specified.
XPosition ?	NumberSpan	Position of the numbering machine along the printer axis. Default = system specified.

Name	Data Type	Description
YPosition ?	NumberSpan	Position of the numbering machine across the printer axis. Default = system specified.
Orientation ?	NumberSpan	Rotation of the numbering machine in degrees. If *Orientation* = 0, the top of the numbers is along the leading edge. Default = 0
Step ?	integer	Number that specifies the difference between two subsequent numbers of the numbering machine. Default = 1
SeparationSpec?	element	Specifies the name of the **Color** in the **ColorPool** that is used for Numbering.

7.1.14 PackingIntent

This resource specifies the packaging intent for a JDF job, using information that identifies the type of package, the wrapping used, and the shape of the package. Note that this specifies packing for shipping only, not packing of items into custom boxes, etc. Boxes are convenience packaging and are not envisioned to be protection for shipping. Cartons perform this function. All quantities are specified as finished pieces per wrapped/boxed/carton or palletized package. The model for packaging is that products are wrapped together, wrapped packages are placed in *boxes*, boxes are placed in *cartons*, and cartons are stacked on *pallets*.

Resource Properties
Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
BoxedQuantity ?	IntegerSpan	How many units of *product* in a box.
BoxShape ?	ShapeSpan	Describes the length, width, and height of the box in points.
CartonQuantity ?	IntegerSpan	How many units of *product* in a carton.
CartonShape ?	ShapeSpan	Describes the length, width ,and height of the carton in points. For example, 288 544 1012
CartonMaxWeight ?	NumberSpan	Maximum weight of an individual carton in kilograms.
CartonStrength ?	NumberSpan	Strength of the carton in kilograms.
FoldingCatalog ?	NameSpan	Description of the folding scheme for folding the product for packaging as specified in the **FoldingParams** folding catalog attribute in the format "Fx-y". See JDF Folding Catalog descriptions in Figure 7.11 Fold Catalog part 1 and Figure 7.12 Fold Catalog part 2. Note: The folding scheme in this context refers to the folding of the finished product for packaging only. The folding has no effect on the page/folio definition.
PalletQuantity ?	IntegerSpan	Number of *product* per pallet
PalletSize ?	XYPairSpan	Describes the length and width of the pallet in points, e.g., 3500 3500
PalletMaxHeight ?	NumberSpan	Maximum height of a loaded pallet in points.
PalletMaxWeight ?	NumberSpan	Maximum weight of a loaded pallet in kilograms.

Name	Data Type	Description
PalletType ?	NameSpan	Type of pallet used. Examples include: *2Way*: Two-way entry *4Way:* Four-way entry *Euro:* Standard 1*1 m Euro pallet
PalletWrapping ?	NameSpan	Wrapping of the completed pallet. Examples include: *StretchWrap* *Banding* Default = *None*
WrappedQuantity ?	IntegerSpan	Number of units of product per wrapped package.
WrappingMaterial ?	NameSpan	Examples include: *RubberBand* *ShrinkWrap* *PaperBand* *Polyethylene* Default = *None*

7.1.15 ProductionIntent

This resource specifies the manufacturing intent and considerations for a JDF job using information that identifies the desired result or specified manufacturing path.

Resource Properties

Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
PrintPreference ?	Enumeration-Span	Intended result or goal. Possible values are: *Balanced* – Request for a manufacturing process that balances the requirements for cost, speed and quality. The default. *CostEffective* – Request for the most cost effective manufacturing process. *Fastest* – Request for the most time effective manufacturing process. Cost and Quality may be sacrificed for a fast turn-around time. *HighestQuality* – Request for the manufacturing process which will result in the highest quality.

Name	Data Type	Description
PrintProcess ?	Enumeration-Span	Print process requested. Allowed values are: *Electrophotography* *Flexography* *Gravure* *Inkjet* *Lithography* *Letterpress* *Screen* *Thermography*

7.1.16 ProofingIntent

This resource specifies the prepress proofing intent for a JDF job, using information that identifies the type, quality, brand name and overlay of the proof. The delivery options of proofs are specified in **DeliveryIntent**.

Resource Properties

Resource class:	Intent
Resource referenced by:	-
Example Partition:	*Option*
Input of processes:	Any product node
Output of processes:	-

Resource Structure

Name	Data Type	Description
ProofItem * New in JDF 1.1	element	Specifies the details of the proofs that are required. If no ProofItem exists in a ProofingIntent, it explicitly specifies that no proofs are desired.

Structure of the ProofItem Element

All parameters of **ProofingIntent** have been moved into ProofItem in JDF 1.1.

Name	Data Type	Description
Amount ? Modified in JDF 1.1	IntegerSpan	Specifies the total number of copies of this proof that is required. If not specified, it defaults to an IntegerSpan with *Preferred* = 1.
BrandName ? Modified in JDF 1.1	StringSpan	Brand name of the proof, such as "Iris".
ColorType ? Modified in JDF 1.1	Enumeration-Span	Color quality of the proof. Possible values are: *Monochrome* – Black and white. *BasicColor* – Color does not match precisely. This implies the absence of a color matching system. *MatchedColor* –Color is matched to the output of the press using a color matching system.
Contract ? Modified in JDF 1.1	boolean	Requires proof to be a legally binding, accurate representation of the image to be printed, e.g., color quality requirements have been met when the printed piece acceptably matches the proof. If *true*, a contract proof is required. If false, a lesser proof demonstrating content, color-breaks, or position is adequate. Default = *false*

Name	Data Type	Description
HalfTone ? Modified in JDF 1.1	OptionSpan	Specifies whether the proof should emulate halftone screens. Default = *false*
PageIndex ? New in JDF 1.1	IntegerRangeList	List of pages in the numbering scheme given by the *FolioCount* attribute of the component that should be proofed. Default = 0~-1, i.e., all pages.
ProofName ? New in JDF 1.1	string	Name of the ProofItem. This field must exist, if delivery of a proof is specified in **DeliveryIntent**.
ProofTarget ? Modified in JDF 1.1	URL	Identifies a remote target for the proof output. This can be either a soft or a hard proofing target.
Technology ? Modified in JDF 1.1	NameSpan	Technology used for making the proof. Possible values are: *BlueLine* *DyeSub* *InkJet* *Laser* *PressProof* *SoftProof*
ProofType ? Modified in JDF 1.1	EnumerationSpan	The kind of proof. Possible values are: *Page* – Page proof *Imposition* – Imposition proof *None* – No Proof is required. The default.
*SeparationSpec ** New in JDF 1.1	element	Separations that are to be proofed. Default = all separations

7.1.17 ShapeCuttingIntent

This resource specifies form and line cutting for a JDF job. The cutting processes are applied for producing special shapes like an envelope window or a heart-shaped beer mat. Information that identifies the type and shape of cuts can be described. The cutting process(es) can be performed using tools such as hollow form punching, perforating, or die-cutting equipment.

Resource Properties
Resource class: Intent
Resource referenced by: -
Example Partition: *Option*
Input of processes: Any product node
Output of processes: -

Resource Structure

Name	Data Type	Description
ShapeCut *	element	Array of all ShapeCut elements. Used when each shape is exactly specified.

Structure of ShapeCut Subelement

Name	Data Type	Description
CutBox ?	rectangle	Specification of a rectangular window.

Name	Data Type	Description
CutOut ?	boolean	If *true*, the inside of a specified shape must be removed. If *false*, the outside of a specified shape must be removed. An example of an inside shape is a window, while an example of an outside shape is a shaped greeting card. Default = *false*
CutPath ?	path	Specification of a complex path. This may be an open path in the case of a single line.
Material ?	StringSpan	Transparent material that fills a shape, such as an envelope window, that was cut out when **CutOut** = *true*.
CutType ? Modified in JDF 1.1	Enumeration-Span	Type of cut or perforation used. Possible values are: *Cut:* Full cut. The default. *Perforate*: Interrupted perforation that does not span the entire sheet.
ShapeDepth ? New in JDF 1.1	NumberSpan	Depth of the shape cut. Measured in micron[μm]. If not specified, the shape is completely cut.
Pages ?	Integer-RangeList	List of pages to which this shape must be applied. Only the pages of face-up surfaces should be specified.
ShapeType	Enumeration-Span	Describes any precision cutting other than hole making. Possible values are: *Rectangular* *Round* *Path*
TeethPerDimension ?	NumberSpan	Number of teeth in a given perforation extent in teeth/point. *MicroPerforation* is defined by specifying a large number of teeth (n>1000).

7.1.18 SizeIntent

Deprecated in JDF 1.1

SizeIntent has been deprecated in JDF 1.1. All contents have been moved to **LayoutIntent**. This resource records the size of the finished pages for the product component. It does not, however, specify the size of any intermediate results, such as press sheets.

Resource Properties

Resource class: Intent
Example Partition: *Option*
Resource referenced by: -
Input of processes: Any Product Node
Output of processes: -

Resource Structure

Name	Data Type	Description
Dimensions	XYPairSpan	Specifies the height and width of the product component in pts. Note: Height and width are ambiguously specified in JDF 1.0.
Pages ?	IntegerSpan	Specifies the number of pages of the product component.

Name	Data Type	Description
Type ?	enumeration	Specifies whether the product component referred to is flat or finished. Possible values are:
		Folded = Size of the product after folding. Default value
		Flat = Size of the unfolded sheet. Note that this describes the size of a sheet that is folded to create a product, not the size of the sheet in the press.

7.2 Process Resources

The rest of the resources described in this chapter are what are known as process resources. This means that they serve as necessary components in each of the JDF processes. Section 7.2.1 describes the template for all of the sections that follow. Then every resource already defined for JDF is chronicled, in alphabetical order, below.

7.2.1 Process Resource Template

Each of the following sections begins with a brief narrative description of the resource. Following that is a list containing details about the properties of the resource, as shown below. The first item in the list provides the class of the resource. As was described in Section 3.7.1 Resource Classes, all resources are derived from one of the following eight superclasses: *Intent*, *Parameter*, *Implementation*, *Consumable*, *Quantity*, *Handling* and *PlaceHolder*. All resources inherit additional contents (which may be attributes or elements) from their respective superclasses, and those attributes and elements are not repeated in this section. Thus those attributes associated with a resource of class *Parameter*, for example, can be found in Table 3-11. Note that this inheritance is only valid for atomic resources, i.e., resources that reside directly in a ResourcePool.

Resource elements are listed in separate sections if they may be referenced by more than one resource. For an example, see the resource element **SeparationSpec**. If the resource is not referenced by multiple resources, it is described inside the resource section of the resource to which it belongs. For an example, see the ColorSpaceConversionOp element of the **ColorSpaceConversionParams** resource. The resource class of an atomic resource also defines the superclasses from which the resource inherits additional contents. The Consumable, Quantity, and Handling resource elements inherit from the PhysicalResource element, which in turn inherits from the Resource element. Parameter and Implementation resource elements inherit from the Resource element directly. Non-atomic resources, i.e., resource subelements, do not inherit contents from resource superclasses.

Examples for resources that may be used as atomic resources or resource elements are: **Employee**, **InsertSheet**, **LayoutElement**, and **Media**. For example, if the **Media** is used as an atomic resource, it inherits all content from the resource class Consumable. If it is used as a resource element, then the **Media** may have only an ID as defined by Table 3-22 Contents of the abstract ResourceElement.

After the list describing the resource properties, each section contains tables that outline the structure of each resource and, when applicable, the abstract or subelement information that pertains to the resource structure. The first column contains the name of the attribute or element. In some cases, a resource will contain an element with more than one value associated with it. If this is the case, the element name is listed as often as it appears, and a term in parentheses that identifies the kind of element is included in the column. For an example, see Section 7.2.50 EndSheetGluingParams or 7.2.123 Sheet. An example of the tables in this section is provided below.

Resource Properties Template

Resource class:	Defines the resource class or specifies ResourceElement if the element does not inherit content from a resource class.
Resource referenced by:	List of parent resources that contain elements of this type. Only valid for elements.
Example Partition:	List of valid partitioning boundaries: *BlockName, DocIndex, DocRunIndex, DocSheetIndex, FountainNumber, LayerIDs, Location, Option, PageNumber, PartVersion, PreviewType, RibbonName, Run, RunIndex, RunTag, RunPage, Separation, SetIndex, SheetIndex, SheetName, Side, SignatureName, TileID, WebName* If a partition is specified, the resource may contain nested elements of its own type.
	The list of partitions represents a list of example partition keys for the respective resources. Note that resources may also be partitioned by keys that are not included in the

list, e.g., *PartVersion* and *Location,* which is valid for any resource, respectively physical resource.

Input of processes: List of node types that use the resource as an input resource.

Output of processes: List of node types that create the resource as an output resource

Resource Structure Template

Name	Data Type	Description
Name of attribute	data type of attribute	Usage of the attribute.
Name of element	element	Subelements that must be defined locally within the resource.
Name of element	refelement	Elements that are based on other atomic resources or resource elements. These may either be in-line elements or instances of ResourceRef elements (see Section 3.8.6). In case of ResourceRef elements a "Ref" must be appended to the name specified in the table column entitled "Name".

7.2.2 Address

Definition of an address. The structure is derived from the vCard format and, therefore, is comprised of all address subtypes (ADR:) of the delivery address of the vCard format. The corresponding XML types of the vCard are quoted in the table.

Resource Properties

Resource class: Parameter
Resource referenced by: Contact, Location (see Table 3-14)
Example: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
City ?	string	City or locality of address (vCard: ADR:locality).
Country ?	string	Country of address (vCard: ADR:country).
CountryCode ?	string	Country of address. This value conforms to the ISO 3166 standard in which countries are represented as 2-character codes.
PostBox ?	string	Post office address (vCard: ADR:pobox. For example: P.O. Box 101).
PostalCode ?	string	Zip code or postal code of address (vCard: ADR:pcode).
Region ?	string	State or province (vCard: ADR:region).
Street ?	string	Street address (vCard: ADR:street).
ExtendedAddress ?	telem	Extended address (vCard: ADR:extadd. For example: Suite 245).

7.2.3 AdhesiveBindingParams

Deprecated in JDF 1.1

This resource describes the details of the following four subprocesses of the **AdhesiveBinding** process:

- back preparation
- multiple glue applications
- spine taping
- cover application

These subprocesses are identified as instances of the abstract ABOperation element. Although a workflow may exist that groups these processes according to its own capabilities, it is likely that they will be performed in the order presented. A description of each follows the table containing the contents of the **AdhesiveBindingParams** resource.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *AdhesiveBinding*
Output of processes: -

Resource Structure

Name	Data Type	Description
FlexValue ?	double	Flex quality parameter given in [N/cm].
PullOutValue ?	double	Pull out quality parameter given in [N/cm].
ABOperation +	Element	An abstract element which is a placeholder for an operation (SpinePreparation, GlueApplication, SpineTaping, and CoverApplication). Each ABOperation element describes the parameters of one single operation of the complete *AdhesiveBinding* process.

Side gluing on front side   Spine gluing   Side gluing on back side
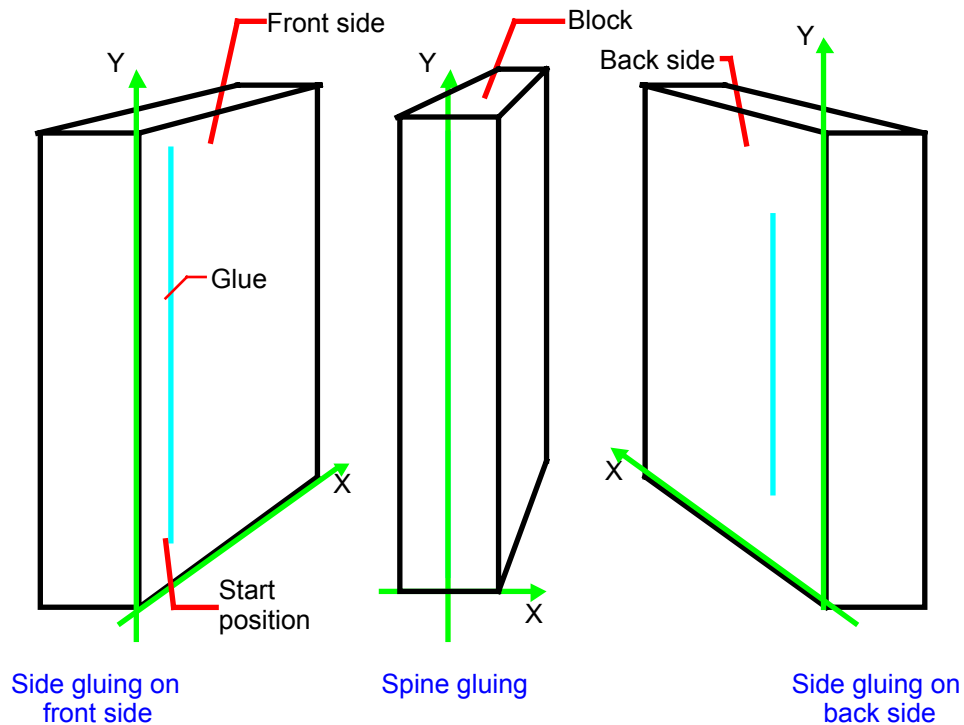
Figure 7.1 Parameters and coordinate system for glue application

7.2.4 ApprovalParams

This resource provides the details of an approval process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Approval*
Output of processes:

Resource Structure

Name	Data Type	Description
ApprovalPerson *	element	List of people (such as a customer, printer, or manager) who can sign the approval.

Structure of ApprovalPerson Subelement

Name	Data Type	Description
Obligated ?	boolean	If *true*, the person has to sign this approval. Default = *true*
Contact	refelement	Contact (such as a customer, printer, or manager) who must sign the approval. The value of the *ContactTypes* attribute of this Contact element should be *Administrator*.

7.2.5 ApprovalSuccess

The signed **ApprovalSuccess** resource indicates the success of a soft proof, color proof, printing proof, or any other sort of proof.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, DocRunIndex, RunIndex, RunPage, RunTag, SetIndex, SheetName, Side, SignatureName, TileID*
Input of processes: any process
Output of processes: *Approval, Verification*

Resource Structure

Name	Data Type	Description
FileSpec ?	refelement	The file that contains the approval signature. If FileSpec does not exist, **ApprovalSuccess** is a logical placeholder.

7.2.6 AutomatedOverprintParams

This resource provides controls for the automated selection of overprinting of black text or graphics.

Resource Properties

Resource class: Parameter
Resource referenced by: RenderingParams, SeparationControlParams
Example Partition: -
Input of processes: -
Output of processes: *-*

Resource Structure

Name	Data Type	Description
OverPrintBlackText ?	boolean	Indicates whether OverPrint should be set to true for black text. If *false*, *TextSizeThreshold* and *TextBlackLevel* are ignored. Default = *false*

Name	Data Type	Description
OverPrintBlackLineArt ?	boolean	Indicates whether overprint should be set to true for black line art. If *false*, **LineArtBlackLevel** is ignored. Default = *false*
TextSizeThreshold ?	integer	Indicates the point size for text below which black text will be set to overprint. For asymmetrically scaled text, the minimum point size between both axes will be used. Default = 99999, i.e., all text is set to overprint.
TextBlackLevel ?	number	A value between 0.0 and 1.0 which indicates the minimum black level for the text stroke or fill colors that cause the text to be set to overprint. Default = 1
LineArtBlackLevel ?	number	A value between 0.0 and 1.0 which indicates the minimum black level for the stroke or fill colors that cause the line art to be set to overprint. Defaults to the value of **TextBlackLevel**.

7.2.7   BlockPreparationParams

New in JDF 1.1

This resource describes the settings of a **BlockPreparation** process. For the tightbacking there are four different kinds of book forms:
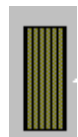
flat   round   flat and backed   rounded and backed
Flat   *Round*   *FlatBacked*   *RoundBacked*

For the rounding and for the backing there are two additional measurements:

Rounding: rounding way   Backing: backing way
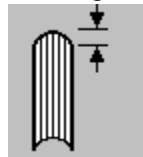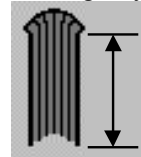
Resource Properties

Resource class: Parameter
Resource referenced by:
Example Partition: -
Input of processes: *BlockPreparation*

Resource Structure

Name	Data Type	Description
Backing ?	number	Backing distance in points. Default =system specified.
Rounding ?	number	Rounding distance in points. Default =system specified.
TightBacking?	enumeration	Definition of the geometry of the back of the book block. This can be one of: *Flat:* The default. *Round*: Rounding way *FlatBacked*: Backing way *RoundBacked*: Rounding way, backing way

Name	Data Type	Description
RegisterRibbon*	refelement	Description of the register ribbons that are included within the book block.

7.2.8   BoxPackingParams

New in JDF 1.1

This resource defines the parameters for packing a box of components. Details of the box used for **BoxPacking** can be found in the **Component (Box)** resource that is also an input of the **BoxPacking** process.

Resource Properties

Resource class: Parameter
Resource referenced by:
Example Partition: -
Input of processes: **BoxPacking**
Output of processes: -

Resource Structure

Name	Data Type	Description
Pattern ?	string	Name of the box packing pattern. Used to store a predefined pattern that defines the layers and positioning of individual component in the box or carton.
FillMaterial ?	NMTOKEN	Material to fill boxes that are not completely filled. Includes *Paper* *Styrofoam* *BlisterPack* Default = *None*

7.2.9   BufferParams

New in JDF 1.1

This resource provides controls for **Buffer** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: **Buffer**
Output of processes: *-*

Resource Structure

Name	Data Type	Description
MinimumWait ?	duration	Minimum amount of time that an individual resource must be buffered.

7.2.10 Bundle

New in JDF 1.1

Bundles are used to describe sets of components.

Resource Properties

Resource class: Parameter
Resource referenced by: **Component**
Example Partition: -

Input of processes: -
Output of processes: -

Structure of the Bundle Element

Name	Data Type	Description
BundleType	enumeration	One of: *BoundSet:* Stack of components that are bound together. *Box* *Carton* *Palette* *Sheet* – Multiple individual items printed onto one Sheet. *Stack* – Loose stack of components. The default. *WrappedBundle*
FolioCount ?	integer	Total Amount of individual finished pages that this bundle contains. If not specified, it must be calculated from the individual BundleItems.
ReaderPageCount ?	integer	Total Amount of individual reader pages that this bundle contains. If not specified, it must be calculated from the individual BundleItems.
TotalAmount ?	integer	Total Amount of individual products that this bundle contains. If not specified, it must be calculated from the individual BundleItems.
BundleItem *	refelement	References to the individual items that form this Bundle.

Structure of the BundleItem Element

Name	Data Type	Description
Amount	integer	Number of this type of items.
Orientation ?	enumeration	Named Orientation of the **Component** respective to the Bundle coordinate system. Allowed values are: *Rotate0* *Rotate90* *Rotate180* *Rotate0* *Flip270* *Flip90* *Flip180* *Flip270* For details, of the semantics of the enumeration, see Table 2-3 Only one of *Orientation* or *Transformation* may be specified.
Transformation ?	matrix	Orientation of the **Component** respective to the Bundle coordinate system.
Component	refelement	Reference to a Component that is part of this Bundle.

The following example code shows a JDF that describes Boxing and Palletizing for 4200 books. The appropriate Bundle elements are ==highlighted== The resources have not yet been completely filled in.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="Bundle" Type="ProcessGroup" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting"
Version="1.1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <!-- The BoxPacking process consumes the thing to pack and the boxes-->
  <!-- The BoxPacking process creates packed boxes -->
  <JDF ID="n0235" Type="BoxPacking" Status="Waiting">
    <ResourceLinkPool>
      <ComponentLink rRef="BoxID" Usage="Input" ProcessUsage="Box"/>
```

```
         <BoxPackingParamsLink rRef="BoxParamsID" Usage="Input"/>
         <ComponentLink rRef="ComponentID" Usage="Input"/>
         <ComponentLink rRef="PackedBoxID" Usage="Output"/>
      </ResourceLinkPool>
      <!-- The BoxPacking process has the following local resources -->
      <ResourcePool>
         <BoxPackingParams ID="BoxParamsID" Class="Parameter"  Status="Available" Quantity="42"/>
         <Component ID="BoxID" Class="Quantity" Amount="100"  Status="Available"/>
      </ResourcePool>
   </JDF>
   <ResourcePool>
      <!-- This Component describes a Box with 42 Books -->
      <Component ID="PackedBoxID" Class="Quantity" rRefs="ComponentID" Amount="100"
Status="Unavailable">
         <Bundle BundleType="Box" TotalAmount="42">
           <BundleItem Amount="42">
             <ComponentRef rRef="ComponentID"/>
           </BundleItem>
         </Bundle>
      </Component>
      <Component ID="ComponentID" Class="Quantity" Amount="4200"  Status="Available"/>
      <!-- This Component describes the contents of the palette: 100 Boxes with 42 Books -->
      <Component ID="PaletteContentsID" Class="Quantity" rRefs="PackedBoxID" Amount="10"
Status="Unavailable">
         <Bundle BundleType="Palette" TotalAmount="420">
           <BundleItem Amount="10">
             <ComponentRef rRef="PackedBoxID"/>
           </BundleItem>
         </Bundle>
      </Component>
   </ResourcePool>
   <JDF ID="n0239" Type="Palletizing" Status="Waiting">
      <ResourceLinkPool>
         <ComponentLink rRef="PackedBoxID" Usage="Input"/>
         <PaletteLink rRef="PaletteID" Usage="Input"/>
         <PalletizingParamsLink rRef="PaletteParamsID" Usage="Input"/>
         <ComponentLink rRef="PaletteContentsID" Usage="Output"/>
      </ResourceLinkPool>
      <ResourcePool>
         <Palette ID="PaletteID" Class="Consumable" Amount="10"  Status="Available"/>
         <PalletizingParams ID="PaletteParamsID" Class="Parameter"  Status="Available" Quan-
tity="10"/>
      </ResourcePool>
   </JDF>
</JDF>
```

7.2.11 ByteMap

This resource specifies the structure of bytemaps produced by various processes within a JDF system. A **ByteMap** represents a raster of image data. This data may have multiple bits per pixel, may represent a varying set of color planes, and may or may not be interleaved. A Bitmap is a special case of a **ByteMap** in which each pixel is represented by a single bit per color.

Personalized printing requires that certain regions of a given page be dynamically replaced. The optional mask associated with each band of data allows for omitting certain pixels from the base image represented by the **Byte-Map** so that they may be replaced.

Resource Properties

Resource class:	Parameter
Resource references:	**RunList**
Example Partition:	-
Input of processes:	*Screening*
Output of processes:	*Scanning*, *Rendering*, *Screening*

Resource Structure

Name	Data Type	Description
BandOrdering ?	enumeration	Identifies the precedence given when ordering the produced bands. Possible values are: *BandMajor* – The position of the bands on the page is prioritized over the color. *ColorMajor* – All bands of a single color are played in order before progressing to the next plane. This is only possible with non-interleaved data. This field is required for non-interleaved data and is ignored for interleaved data.
FrameHeight	integer	Height of the overall image that may be broken into multiple bands
FrameWidth	integer	Width of overall image that may be broken into multiple columns
Halftoned	boolean	Indicates whether or not the data has been halftoned.
Interleaved	boolean	If *true*, the data is interleaved, or chunky. Otherwise the data is non-interleaved, or planar.
PixelSkip ?	integer	Number of bits to skip between pixels of interleaved data.
Resolution	XYPair	Output resolution.
Band +	element	Array of bands containing raster data.
FileSpec ?	refelement	A **FileSpec** resource pointing to a location where the raster should be (or already is) stored. The *ResourceUsage* attribute of the FileSpec must be "*RasterFileLocation*".
PixelColorant +	element	Ordered list containing information about which colorants are represented and how many bits per pixel are used.

Structure of Band Subelement

Name	Data Type	Description
Data	URL	Actual bytes of data.
Height	integer	Height in pixels of the band.
Mask ?	URL	1-bit mask of raster data indicating which bits of the band data should actually be used. It is required that the mask dimensions and resolution be equivalent to the contents of the band itself.
WasMarked	boolean	Indicates whether any rendering marks were made in this band. This attribute allows a band to be skipped if no marks were made in the band.
Width	integer	Width in pixels of the band.

Structure of PixelColorant Subelement

Name	Data Type	Description
ColorantName	string	Name of colorant.
PixelDepth	integer	Number of bits per pixel for each colorant.

7.2.12 CaseMakingParams

New in JDF 1.1

This resource describes the settings of a *CaseMaking* process.
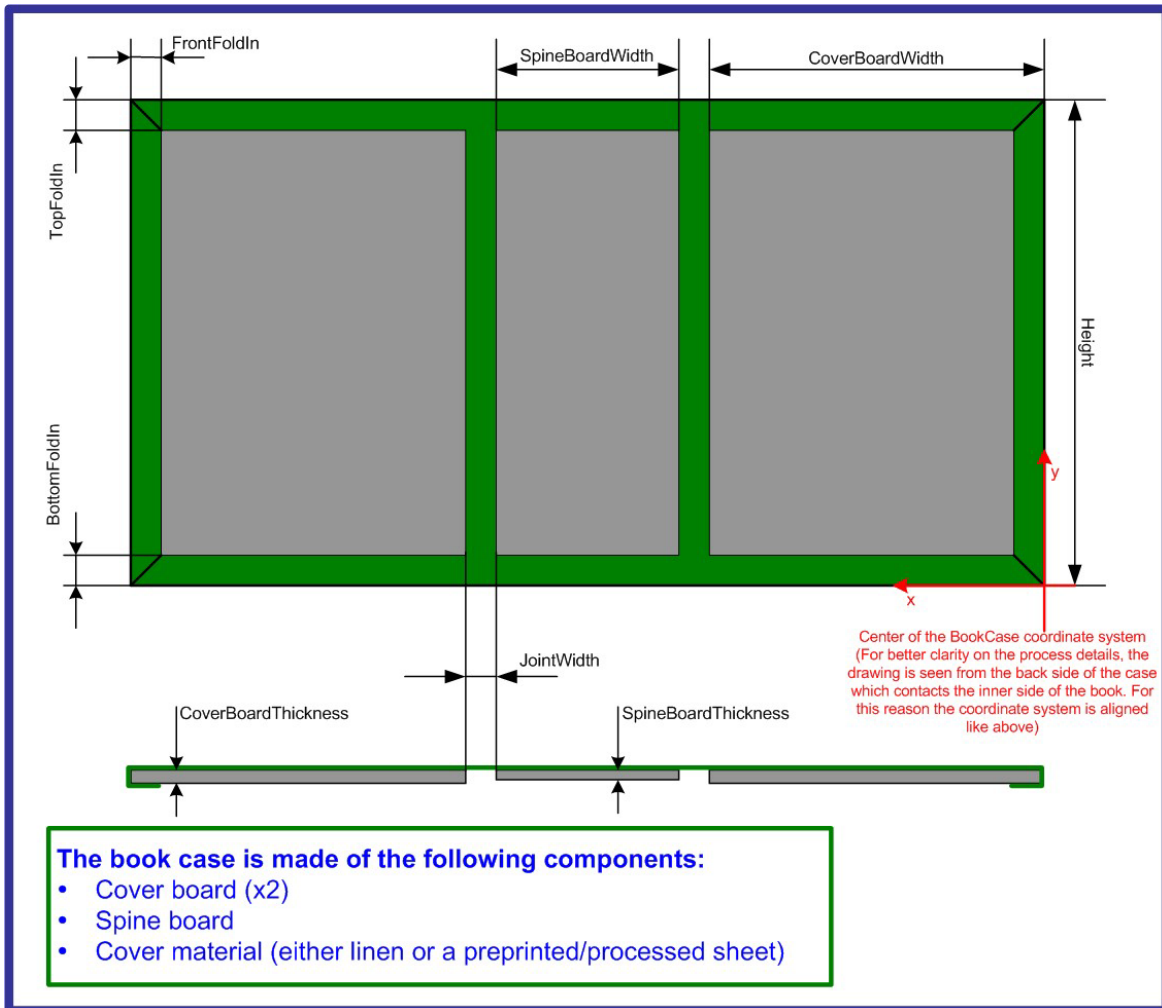
Figure 7.2 CaseMakingParams

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *CaseMaking*

Resource Structure

Name	Data Type	Description
BottomFoldIn ?	number	Defines the width of the part of the CoverMaterial on the lower edge inside of the case. If not specified, defaults to *TopFoldIn.*
CoverWidth ?	number	Width of the cover cardboard in points.
CornerType ?	NMTOKEN	Method of wrapping the corners of the cover material around the corners of the board. Possible values include: *LibraryCorner*: the American Library Corner style. If not specified defaults to the equipment specific setting.
FrontFoldIn ?	number	Defines the width of the part of the cover material on the front edges inside of the case.
Height	number	Height of the book case in points.
JointWidth	number	Width of the joint in points as seen when laying the cardboard on the CoverMaterial.
SpineWidth	number	Width of the spine cardboard in points.
TopFoldIn ?	number	Defines the width of the part of the CoverMaterial on the top edge inside of the case.
GlueLine	refelement	As the glue is applied to the whole back side of the cover material, *AreaGluing* must be set to true.

7.2.13 CasingInParams

New in JDF 1.1

This resource describes the settings of a *CasingIn* process. The geometry is always centered.
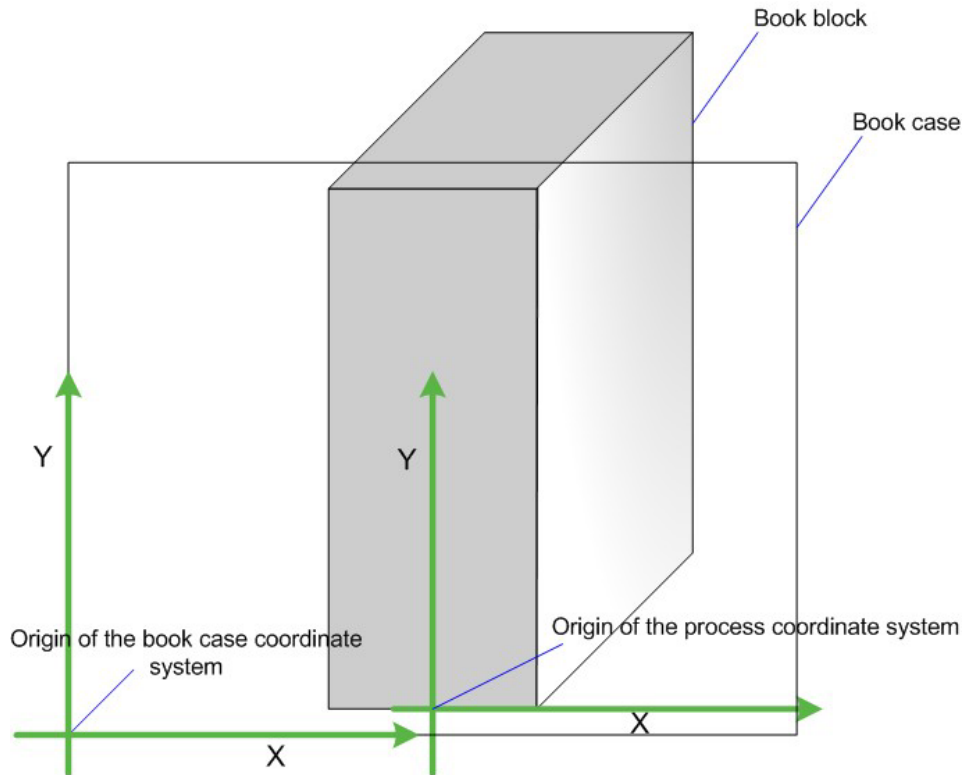
Figure 7.3 Parameters and Coordinate System for CasingIn

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: CasingIn

Resource Structure

Name	Data Type	Description
CaseRadius ?	number	Inner radius of the case spine rounding. If not specified, no rounding of the case spine is performed.
GlueLine +	refelement	Properties of the glue used.

7.2.14 ChannelBindingParams

This resource describes the details of the **ChannelBinding** process. Figure 7.4 depicts the **ChannelBinding** process.
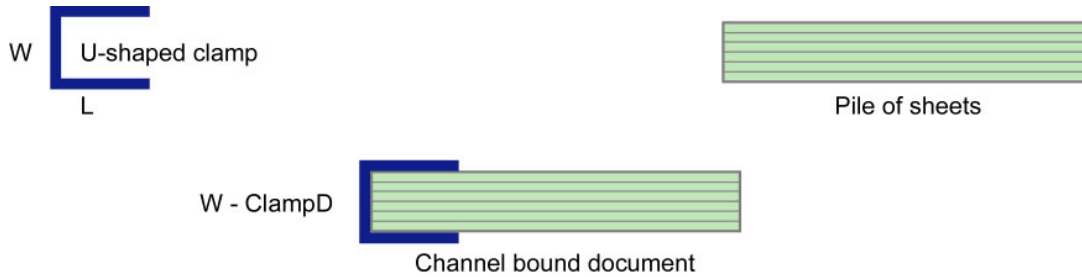
Figure 7.4 Parameters used for channel binding

The symbols W, L, and ClampD of Figure 7.4 are described by the attributes *ClampD* and *ClampSize* of the table below.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ChannelBinding*
Output of processes: -

Resource Structure

Name	Data Type	Description
Brand ?	string	The name of the clamp (or preassembled cover with clamp) manufacturer and the name of the specific item.
ClampColor ?	NamedColor	Determines the color of the clamp/cover. If the clamp is inside of a preassembled cover, then the color of the cover is meant. Default= Default =system specified.
ClampD ?	double	The distance of the clamp that was "pressed away" (see Figure 7.4).
ClampSize ?	shape	The shape size of the clamp. The first number of the shape data type corresponds to the clamp width W (see Figure 7.4) which is determined by the final height of the block of sheets to be bound. The second number corresponds to the length L (see Figure 7.4). The third corresponds to the spine length (not visible in Figure 7.4. The spine length is perpendicular on the paper plane).
ClampSystem ?	boolean	If *true* the clamp is inside of a pre assembled cover. Default = *false*

7.2.15 CIELABMeasuringField

Information about a color measuring field. The color is specified as CIE-L*a*b* value.

Resource Properties

Resource class: Parameter
Resource referenced by: ColorControlStrip, Surface
Example Partition: -
Input of processes: Any printing process
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the color measuring field in the coordinates of the MarkObject that contains this mark. If the measuring field is defined within a ColorControlStrip, *Center* refers to the rectangle defined by *Center* and *Size* of the ColorControlStrip.

Name	Data Type	Description
CIELab	LabColor	L*a*b* color specification.
DensityStandard ? Deprecated in JDF 1.1	enumeration	Density filter standard used during density measurements. Possible values are: *ANSIA* – ANSI Status A *ANSIE* – ANSI Status E *ANSII* – ANSI Status I *ANSIT* – ANSI Status T. The default value *DIN16536* *DIN16536NB*
Diameter ? Modified in JDF 1.1	double	Diameter of measuring field.
Light Deprecated in JDF 1.1	NMTOKEN	Type of light. Possible values include: *D50* *D65*
Observer Deprecated in JDF 1.1	integer	Observer in degree (2 or 10)
Percentages ?	NumberList	Percentage values for each separation. The number of array elements must match the number of separations.
ScreenRuling ?	NumberList	Screen ruling values in lines per inch for each separation. The number of array elements must match the number of separations.
ScreenShape ?	string	Shape of screening dots.
Setup ? Deprecated in JDF 1.1	string	Description of measurement setup.
Tolerance ? Modified in JDF 1.1	double	Tolerance in ΔE.
ColorMeasurement-Conditions ? New in JDF 1.1	refelement	Detailed description of the measurement conditions for color measurements.

7.2.16 CoilBindingParams

This resource describes the details of the **CoilBinding** process.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *CoilBinding*
Output of processes: -

Resource Structure

Name	Data Type	Description
Brand ?	string	The name of the coil manufacturer and the name of the specific item. Default =system specified.

Name	Data Type	Description
Color ?	NamedColor	Determines the color of the coil. Default =system specified.
Diameter ?	double	The coil diameter to be produced is determined by the height of the block of sheets to be bound. Default =system specified.
Material ?	enumeration	The material used for forming the coil binding: *LaqueredSteel* *NylonCoatedSteel* *PVC* *TinnedSteel* *ZincsSteel* *Default* = system specified
Shift ?	double	Amount of vertical shift that occurs as a result of the coil action while opening the document. It is determined by the distance between the holes. Default =system specified.
Thickness ?	double	The coil's thickness. Default =system specified.
Tucked ?	boolean	If true, the ends of the coils are "tucked in". Default = false

7.2.17 CollectingParams

The **Collecting** process needs no special attributes. However, this resource is provided as a container for extensions of the **Collecting** process.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Input of processes: *Collecting*
Output of processes: -
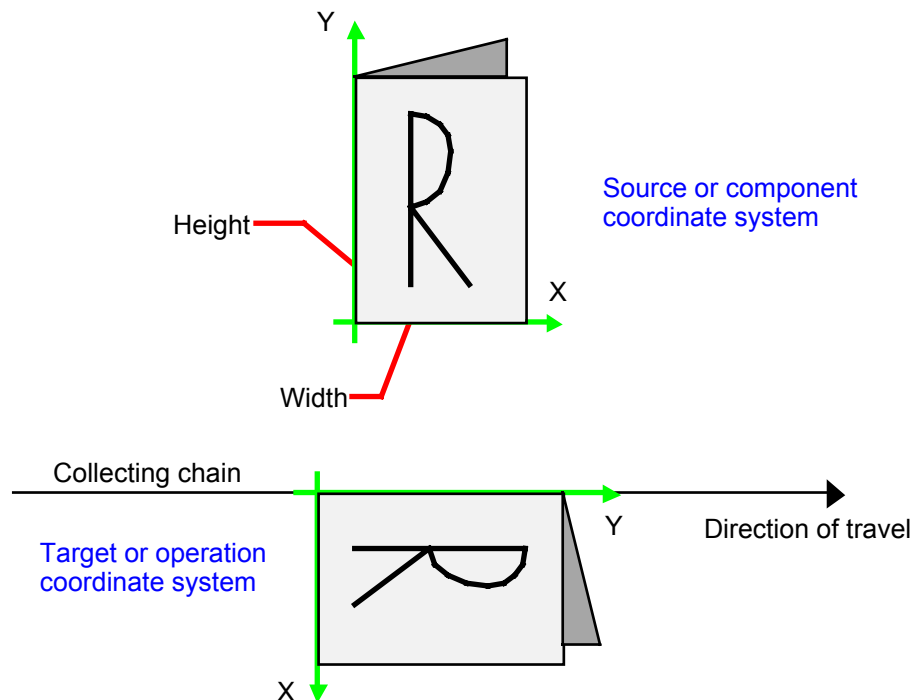
Figure 7.5 Coordinate systems used for collecting

7.2.18 Color

JDF describes spot color inks and, along that line, process color (inks). Spot colors are named colors that may either be separated or converted to process colors. It is important to know the density of the colorant (for trapping) and, in many cases, the *Lab* values (for representing them on screen). If you know the *Lab* value, you can calculate the density. When representing colors on screen, a conversion to process colors must be defined. This conversion is a simple linear interpolation between the *CMYK* value of the 100% spot color and its tint.

A color is represented by a Color element. It has a required *Name* attribute, which represents the name of either a spot color or a process color. The four names that are reserved for representing process CMYK color names are *Cyan*, *Magenta*, *Yellow*, and *Black*. Every colorant can have a *Lab* and/or *CMYK* color value. If both are specified and a system is capable of interpreting both values, the *Lab* value overrides the *CMYK* definition, unless the target device is compatible with the *CMYKTarget*. In this case the CMYK value has precedence.

The *Lab* value represents the *Lab* readings of the ink on certain media. This means that spot inks printed on two different kinds of stocks have different *Lab* values. Pantone books, for example, provide *Lab* values for two kinds of paper: *coated* (not necessarily glossy) and *uncoated*. Thus a color of ink should identify the media for which it is specified. CMYK colors are used to approximate spot colors when they are not separated. This conversion can be done by a color management system, or there can be fixed CMYK representation defined by colorbooks such as Pantone.

Resource Properties

Resource class: Parameter
Resource referenced by: ColorPool, Media, TrappingDetails
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
CMYK ?	CMYKColor	CMYK value of the 100 % tint value of the colorant. Although optional, it is highly recommended that this value be filled.
ColorBook ?	NMTOKEN	Definition of the color identification standard that is used to represent this color. Examples include: *HKS* *Pantone* *Toyo* Default = *None*
ColorBookEntry ?	string	Definition of the **Color** within the *ColorBook* standard. Maps to the NCL2 value of a namedColorType tag of an ICC color profile. Defaults to an empty string. This entry is used to map from the JDF **Color** to an ICC namedColorType tag.
ColorBookPrefix ?	string	Definition of the name prefix of the color book entry within a named ICC profile. Default = empty string. This entry is used to map from the JDF **Color** to an ICC namedColorType tag.
ColorBookSuffix ?	string	Definition of the name suffix of the color book entry within a named ICC profile. Default = empty string. This entry is used to map from the JDF **Color** to an ICC namedColorType tag.
ColorName ? New in JDF 1.1	NamedColor	Mapping to a color name. Allowed values are defined in the appendix section A.2.8 NamedColor.
ColorType ?	enumeration	A name that characterizes the colorant. If no value is specified, the device must provide a default value. Possible values are: *DieLine* – Marks made with colorants of this type are ignored for

Name	Data Type	Description
		trapping. Trapping processes need not generate a color plane for this colorant. *DieLine* can be used for auxiliary process separations. *Normal* – Marks made with colorants of this type, marks covered by colorants of this type, and marks on top of colorants of this type are trapped. *Transparent* – Marks made with colorants of this type are ignored for trapping. Trapping processes need not generate a color plane for this colorant. Transparent can be used for varnish. *Opaque* – Marks covered by colorants of this type are ignored for trapping. *Opaque* can be used for metallic inks. *OpaqueIgnore* – Marks made with colorants of this type and marks covered by colorants of this type are ignored for trapping. *OpaqueIgnore* can be used for metallic inks.
Lab ?	LabColor	Lab value of the 100 % tint value of the colorant.
MediaType ?	string	Specifies the media type. Possible values are: *Coated* *Uncoated*
Name	string	Name of the colorant. This is the value that must match the *Name* attribute of a SeparationSpec that references this color.
NeutralDensity ?	number	A number in the range of 0.001 to 10 that represents the neutral density of the colorant, defined as 10*log(1/Y). Y is the tristimulus value in CIEXYZ coordinates, normalized to 1.0. If no value is specified, the device must provide a default.
sRGB ?	sRGBColor	sRGB value of the 100 % tint value of the colorant.
UsePDLAlternateCS ?	boolean	If *true*, the alternate colorspace definition defined in the PDL must be used for color space transformations when available. If *false*, the alternate color space definitions defined in *sRGB*, *CMYK* or DeviceNColor of this **Color** must be used depending on the value of ProcessColorModel in ColorantControl. Default = *true*
ColorMeasurement-Conditions ? New in JDF 1.1	refelement	Detailed description of the measurement conditions for color measurements.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC named color profile that describes further details of the color. The *ResourceUsage* attribute of the FileSpec must be "*ColorProfile*". See Section 7.2.52 FileSpec.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that defines the target output device in case the object that uses the **Color** has been colorspace converted to a device color space. TargetProfile applies to the alternate color defined by the value of *UsePDLAlternateCS*. The *ResourceUsage* attribute of the FileSpec must be "*TargetProfile*". See Section 7.2.52 FileSpec.
DeviceNColor *	element	Elements that defines the colorant in a non-standard device-dependent process color space.
TransferCurve * Modified in JDF 1.1	refelement	A list of color transfer functions that is used to convert a tint value to one of the alternative colorspaces. The transfer functions that are not specified here default to a linear transfer: "0 0

Name	Data Type	Description
		1 1"

Structure of DeviceNColor Subelement

Name	Data Type	Description
ColorList	NumberList	Value of the 100 % tint value of the colorant in the ordered De-viceN space. The list must have *N* elements. A value of 0 specifies no ink and a value of 1 specifies full ink. The mapping of indices to colors is specified in the DeviceNSpace element of the **ColorantControl** resource.
N	integer	Number of colors that define the color space.
Name	string	Color space name, such as HexaChrome or HiFi. *Name* must match the *Name* attribute of a DeviceNSpace element defined in a ColorantControl resource.

Color Example

This is an example of the structure for colorant. The transfer curves in this example are defined for process CMYK and sRGB, independently.

```
<Color Name="Pantone Deep Blue" Density="3.14" MediaType="Coated"
Lab="20. 30. 40." CMYK="0.2 0.3 0.4 0.5" sRGB="0.6 0.7 0.9">
<TransferCurve Separation ="Cyan" Curve="0 0 .5 .4 1 1"/>
<TransferCurve Separation ="Magenta" Curve="0 0 .5 .6 1 1"/>
<TransferCurve Separation ="Yellow" Curve="0 0 1 1"/>
<TransferCurve Separation ="Black" Curve="0 0 1 1"/>
<TransferCurve Separation ="sRed" Curve="0 0 1 1"/>
<TransferCurve Separation ="sGreen" Curve="0 0 1 1"/>
<TransferCurve Separation ="sBlue" Curve="0 0 1 1"/>
<Color/>
```

7.2.19 ColorantControl

ColorantControl is a resource used to control the use of color when processing PDL pages. The attributes and elements of the **ColorantControl** resource describe how color information embedded in PDL pages must be trans-lated into device colorant information.

Colorants are referenced in **ColorantControl** by name only. Additional details about individual colorants can be found in the Color element of the **ColorPool** resource. **ColorantControl** resources control which device col-orants will be used as well as how document colors will be converted into device color spaces and how conflicting color information should be resolved. Separation control is specified by the *Separation* process being present.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of Processes: *ColorSpaceConversion, Screening, Separation, Trapping*
Output of processes: *ColorSpaceConversion*

Resource Structure

Name	Data Type	Description
ForceSeparations ?	boolean	If *true*, forces all colorants to be output as individual separations, re-gardless of any values defined in ColorantControl, i.e., all separations in a document are assumed to be valid and are output individually.
		Default = *false*, which means respect the parameters specified in **Col-orantControl** and elsewhere in the JDF.

Name	Data Type	Description
ProcessColorModel ? Modified in JDF 1.1	NMTOKEN	Specifies the model to be used for rendering the colorants defined in color spaces into process colorants. Possible values include: *DeviceCMY* *DeviceCMYK* *DeviceGray* *DeviceN* *DeviceRGB* Default = system specified
ColorantAlias *	element	Identify one or more named colorants that should be replaced with a specified named colorant.
ColorantOrder ? Modified in JDF 1.1a	element	The ordering of named colorants to be processed, for example in the RIP. All of the colorants named must either occur in the Colorant-Params list, or be implied by the *ProcessColorModel.* If present, then only the colorants specified by *ColorantOrder* are output. If not present, then all colorants specified in ColorantParams and implied by *ProcessColorModel* are output.
ColorantParams ?	element	A set of named colorants. This list defines all the colorants that are expected to be available on the device where the process will be exe-cuted. The colorants implied by the value of *ProcessColorModel* are assumed and must not be specified in this list.
ColorPool ?	refelement	Pool of Color elements that define the specifics of the colors named in ColorantControl.[1]
ColorSpaceSubstitute *	element	These subelements identify a colorant that should be replaced by an-other colorant.
DeviceColorantOrder ?	element	The ordering of named colorants to be output on the device[2], such as press modules. All of the colorants named must occur in the Color-antParams list, or be implied by the *ProcessColorModel.* If the De-viceColorantOrder element is not specified, the element defaults to ColorantOrder.
DeviceNSpace *	element	Defines the colorants that make up a DeviceN color space.

Structure of ColorantAlias Subelement

Name	Data Type	Description
ReplacementColorant-Name	string	The name of the colorant to be substituted for the colorants named in the SeparationSpec element list.
SeparationSpec *	element	The names of the colorants to be replaced in PDL files.

Structure of ColorantOrder, ColorantParams, and DeviceColorantOrder Elements

Name	Data Type	Description
SeparationSpec *	element	The names of the colorants that define the respective lists.

---

[1] Note that this will generally be an inter-resource link.
[2] Note that this must be synchronized with the device output ICC profile.

Structure of ColorSpaceSubstitute Subelement

Name	Data Type	Description
PDLResourceAlias	element	A reference to a color space description that replaces the color space defined by TargetColorantName.
SeparationSpec +	element	A list of names that defines the colorants to be replaced. This could be a single name in the case of a *Separation* color space, or more than one name in the case of a DeviceN color space.

Structure of DeviceNSpace Subelement

Name	Data Type	Description
Name ?	string	Color space name, such as *HexaChrome* or HiFi.
N	integer	The number of colors that define the color space.
SeparationSpec *	element	Ordered list of colorant names that define the DeviceN colorspace. The ordering maps to the ordering of elements in the corresponding Color::DeviceNColor::*ColorList* attribute. Note that these colorants must be specified in the ColorantParams element of the **ColorantControl** or be implied by *ProcessColorModel* . In other words, they must be real physical colorants.

7.2.20 ColorControlStrip

This resource describes a color control strip. The type of the color control strip is given in the *StripType* attribute. If it is known at the system reading the JDF file, there is no need to define the elements of the strip, and the attribute *DensityMeasuringFields* is not needed. Otherwise, this attribute must contain a definition of the contained measuring fields. The lower left corner of the control strip box is used as the origin of the coordinate system used for the definition of the measuring fields. It can be calculated using the following formula:

$$x_0 = x - \frac{w}{2}\cos(\varphi) + \frac{h}{2}\sin(\varphi)$$

$$y_0 = y - \frac{w}{2}\sin(\varphi) - \frac{h}{2}\cos(\varphi)$$

where x = X element of the *Center* attribute
y = Y element of the *Center* attribute
w = X element of the *Size* attribute
h = Y element of the *Size* attribute
φ = Value of the *Rotation* attribute

Resource Properties
Resource class: Parameter
Resource referenced by: Surface
Example Partition: -
Input of processes: *Any printing process*:
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the color control strip in the coordinates of the MarkObject that contains this mark.
Rotation ?	double	Rotation in degrees. Positive graduation figures indicate counter-clockwise rotation; negative figures indicate clockwise rotation.
Size	XYPair	Size of the color control strip.
StripType ?	NMTOKEN	Type of color control strip. This attribute can be used for specify-

Name	Data Type	Description
		ing a predefined, company-specific color control strip.
CIELABMeasuringField * New in JDF 1.1	refelement	Details of a CIELab measuring field that is part of this **ColorControlStrip**.
DensityMeasuringField * New in JDF 1.1	refelement	Details of a density measuring field that is part of this **ColorControlStrip**.

7.2.21 ColorCorrectionParams

This resource provides the information needed for an operator to correct colors on some PDL pages or content elements such as image, graphics, or formatted text.

Resource Properties

Resource class:　　　　　Parameter
Resource referenced by: -
Example Partition:　　　*DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes:　　　*ColorCorrection*
Output of processes:　　 -

Resource Structure

Name	Data Type	Description
ColorManagementSystem ?	string	Identifies the preferred ICC color-management system to use when performing corrections. Overrides the default selection of the application or the selection contained in any of the profiles when specified.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the characterization of the final output target device. The *ResourceUsage* attribute of the FileSpec must be "*FinalTargetDevice*".
FileSpec ? Deprecated in JDF 1.1	refelement	A **FileSpec** resource pointing to an ICC profile that describes the assumed characterization of *CMYK*, *RGB* and *Gray* colorspaces. The *ResourceUsage* attribute of the FileSpec must be "*WorkingColorSpace*".
ColorCorrectionOp *	element	List of ColorCorrectionOp subelements.

It is assumed that color correction will be performed by a human operator. No attempt is made to encode specific types of operations. Subelements of the **ColorCorrectionParams** resource should contain a Comment to describe the desired correction operation, and, optionally, to provide a region to be corrected via the Comment::Path or Comment::Box elements.

Structure of ColorCorrectionOp Subelement

Name	Data Type	Description
SourceObjects ?	enumerations	Identifies which class(es) of incoming graphical objects will be operated on. Possible values are: *All* – Default value. *ImagePhotographic* – Contone images. *ImageScreenShot* – Images largely comprised of rasterized vector art. *Text* *LineArt* *SmoothShades* – Gradients and blends.

7.2.22 ColorMeasurementConditions

New in JDF 1.1

This resource contains information about the specific measurement conditions for spectral or densitometric color measurements. Spectral measurements refer to CIE Publication 15.2 - 1986 "Colorimetry, Second Edition" and ISO 13655:1996 "Graphic technology - Spectral measurement and colorimetric computation for graphic arts images." The default measurement conditions for spectral measurements are illuminant D50 and 2 degree observer.

Density measurements refer to ISO 5-3:1995 "Photography – Density measurements – Part 3: Spectral conditions" and ISO 5-4:1995 "Photography – Density measurements – Part 4: Geometric conditions for reflection density." The default measurement conditions for densitometric measurements are density standard ISO/ANSI Status T, calibration to absolute white and using no polarization filter.

Resource Properties

Resource class: Parameter
Resource referenced by: CIELABMeasuringField, Color, DensityMeasuringField
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
DensityStandard ?	enumeration	Density filter standard used during density measurements. Possible values are: *ANSIA* – ANSI Status A *ANSIE* – ANSI Status E *ANSII* – ANSI Status I *ANSIT* – ANSI Status T. The default value *DIN16536* *DIN16536NB*
Illumination ?	enumeration	Illumination used during spectral measurements. Possible values are: *D50* – Default value. *D65* *Unknown*
InkState ?	enumeration	State of the ink during color measurements. Possible values are: *Dry* – Default value. *Wet* *NA*
Instrumentation ?	string	Specific instrumentation used for color measurements, e.g., manufacturer, model number and serial number.
MeasurementFilter ?	enumeration	Optical Filter used during color measurements. Possible values are: *None* – No filter used. Default value. *Pol* – Polarization filter used *UV* – Ultraviolet cut filter used
Observer ?	integer	CIE standard observer function (2 degree and 10 degree) used during spectral measurements. Values are in degree (2 or 10). Default = 2

Name	Data Type	Description
SampleBacking ?	enumeration	Backing material used behind the sample during color measurements. Possible values are: *Black* – Default value. *White* *NA*
WhiteBase ?	enumeration	Reference for white calibration used for density measurements. Possible values are: *Absolute* – Means the instrument is calibrated to a device specific calibration target (absolute white) for absolute density measurements. Default value *Paper* – Means the instrument is calibrated relative to paper white

7.2.23 ColorPool

The **ColorPool** resource contains a pool of all Color elements referred to in the job. In general it will be referenced as a ResourceRef from within resources that require access to color information.

Resource Properties

Resource class: Parameter
Resource referenced by: ColorantControl
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Color *	element	Individual named color.
ColorantSetName ?	string	A string used to identify the named colorant parameter set. This string will be used to identify a set of color definitions (typically associated with a particular class of job or a particular press).

7.2.24 ColorSpaceConversionParams

This set of parameters defines the rules for a ***ColorSpaceConversion*** process, the elements of which define the set of operations to be performed. Information inside the ColorSpaceConversionOp elements, described below, defines the operation and identifies the colorspaces and types of objects to operate on. Other attributes define the color management system to use, as well as the working color space and the final target device.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: ***ColorSpaceConversion***, ***Proofing***, ***SoftProofing***
Output of processes: -

Resource Structure

Name	Data Type	Description
ColorManagementSystem ?	string	Identifies the preferred ICC color management system to use when performing transformations. Overrides the default selection of the application or that contained in any of the profiles when specified. This string should match the ICC CMMType value.

Name	Data Type	Description
ConvertDevIndepColors ? Deprecated in JDF 1.1	boolean	When *true*, incoming device-independent colors are processed to the selected device space. If the chosen operation is *untag* and the characterization data are in the form of an ICC profile, then the profile is removed. Otherwise, these colors are left untouched. Default = *false.* The functionality of *ConvertDevIndepColors* is superceded by including one or more Color-SpaceConversionOp with *SourceCS="DevIndep"* in JDF 1.1.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the characterization of the final output target device. This item is required when converting, but optional for tagging or untagging. The *ResourceUsage* attribute of the FileSpec must be *"FinalTargetDevice"*.
FileSpec ? Deprecated in JDF 1.1	refelement	A **FileSpec** resource pointing to an ICC profile that describes the assumed characterization of *CMYK*, *RGB* and *Gray* color-spaces. The *ResourceUsage* attribute of the FileSpec must be *"WorkingColorSpace"*.
ColorSpaceConversionOp *	element	List of ColorSpaceConversionOp subelements.

Structure of ColorSpaceConversionOp Subelement

Name	Data Type	Description
IgnoreEmbeddedICC ?	boolean	If *true*, specifies that embedded source ICC profiles must be ignored and that the ICC profile defined by *SourceProfile* must be used instead. Default = *false*
Operation	enumeration	Controls which of five functions the color space conversion utility performs. Possible values are: *Convert* – Transforms graphical elements to final target color space. *Tag* – Associates appropriate working space profile with uncharacterized graphical element. *Untag* – Removes all profiles and color characterizations from graphical elements *Retag* – Removes all profiles and color characterizations from graphical elements and replaces them with the appropriate values. Equivalent to a sequence of *UnTag –> Tag*. *ConvertIgnore* – Removes all profiles and color characterizations from graphical elements and converts to the appropriate values. Equivalent to a sequence of *UnTag –> Convert*.
PreserveBlack ? New in JDF 1.1	boolean	Controls how the tints of black (K in CMYK) should be handled. If PreserveBlack is *false*, these colors are processed through the standard ICC workflow. If PreserveBlack is *true*, these colors should be converted into other shades of black. The algorithm is implementation-specific. Default = *false*

Name	Data Type	Description
RenderingIntent ?	enumeration	Identifies the rendering intents associated with SourceObjects elements. Possible ICC-defined rendering intent values are: *Saturation* *Perceptual* – The default. *RelativeColorimetric* *AbsoluteColorimetric*
RGBGray2Black ?	boolean	This feature controls what happens to gray values (R = G = B) when converting from RGB to CMYK. In the case of MS Office applications and screen dumps, there are a number of gray values in the images and line art. Printers do not want to have CMY under the K (registration). Therefore, they prefer to have K only. This is not true for photographic images. In that case, everything is moved through a link. Default = *false*
SourceCS	enumeration	Identifies which of the incoming color spaces will be operated on. Possible values are: *CMYK* – Operates on *deviceCMYK* or 4-component ICC-based colorspaces. *DevIndep*– Operates on device independent colorspaces. *RGB* – Operates on *deviceRGB*, *calRGB* or 3-component ICC-based colorspaces *Gray* – Operates on *deviceGray*, *calGray* or 1-component ICC-based colorspaces.
SourceObjects ?	enumerations	List of object classes that identifies which incoming graphical objects will be operated on. Possible values are: *All* – Default value. *ImagePhotographic* – Contone images. *ImageScreenShot* – Images largely comprised of rasterized vector art. *Text* *LineArt* *SmoothShades* – Gradients and blends.
FileSpec?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the assumed color space. The default is to use embedded profiles. The *ResourceUsage* attribute of the FileSpec may be "*SourceProfile"*.

7.2.25 ComChannel

A communication channel to a person or company such as an email address, phone number, or fax number.

Resource Properties
Resource class: Parameter
Resource referenced by: Contact, Person
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
ChannelType	enumeration	Type of the communication channel. Possible values are: *Phone* – Telephone number. *Email* – E-mail address. *Fax* – Fax machine. *WWW* – WWW home page or form. *JMF* – JMF messaging channel.
Locator	string	Locator of this type of channel in a form such as a phone number or an email address.

7.2.26 Company

Specifies contacts to a company including detailed information about contact persons and addresses. This structure can be used in many situations where addresses or contact persons are needed. Examples of contacts are customer, supplier, company, and addressees. The structure is derived from the vCard format. It comprises the organization name and organizational units (ORG) of the organizational properties defined in the vCard format. The corresponding XML types of the vCard are quoted in the table.

Resource Properties

Resource class: Parameter
Resource referenced by: Contact
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
OrganizationName	string	Name of the organization or company (vCard: ORG:orgnam. For example: ABC, Inc.).
Contact * Deprecated in JDF 1.1	refelement	A contact of the company.
OrganizationalUnit *	telem	Describes the organizational unit (vCard: ORG:orgunit. For example, if two elements are present: 1. "North American Division" and 2. "Marketing").

7.2.27 Component

Component is used to describe the various versions of semi-finished goods in the press and postpress area, such as a pile of folded sheets that have been collected and must then be joined and trimmed. Nearly every postpress process has a **Component** resource as an input as well as an output. Typically the first components in the process chain are some printed sheets or ribbons, while the last component is a book or a brochure. **Component** resources are grouped by kind in much the same way that nodes are classified as Combined, Process, or Product. The five categories of **Component** resources are: *Ribbon*, *Sheet*, *Block*, *PartialProduct*, and *FinalProduct*. These categories are defined in greater detail below:

Ribbon   Part of the web that enters the folder, divider etc. In case the web is not slit, the web and the ribbon are identical.

Sheet   This source type is appropriate if a flat sheet, e.g., a postcard to be glued in, is used as an input component. "Flat" in this case means that the sheet has not been folded or cut before the operation.

Block   This source type is appropriate if a folded sheet, a cut portion of the sheet, or a cut and folded portion of a sheet is used as an input component.

PartialProduct   This source type is appropriate if a partial product should be used as an input component.

FinalProduct   This source type is appropriate if this **Component** is the final product.

Terms and Definitions for Components

The descriptions of **Component**-specific attributes use some terms whose meaning depends on the culture in which they are used. For example, different cultures mean different things when they refer to the "front" side of a magazine. Other terms, such as binding, are defined by the production process and therefore do not depend on the culture.

     Whenever possible, this specification endeavors to use culturally independent terms. In cases where this is not possible, Western style (left-to-right writing) is assumed. Please note that these terms may have a different meaning in other cultures, e.g., those writing from right to left.

Book-like partial product viewed from first page
(front side)

Calendar-like partial product viewed from first page
(front side)
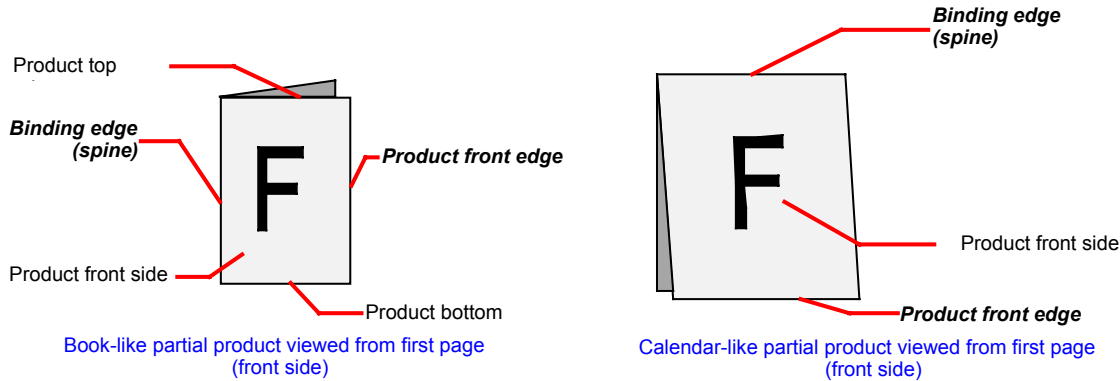
Figure 7.6 Terms and definitions for components

The table below describes the terms used to define the components.

Table 7-1 Terms and definitions for components

Edge	Description
Binding edge	The edge on which the (partial) product is glued or stitched. This edge is also often called *working edge* or *spine*.
Product front edge	The side, where you open the (partial) product. This edge is opposite to the binding edge.
Registered edge	A side on which a collection of sheets or partial products is aligned during a production step. All production steps require two registered edges, which must not be opposite to each other. The two registered edges define the coordinate system used within the production step. When there is a binding edge, this is one of the registered edges.

Resource Properties

Resource class: Quantity
Resource referenced by: -
Example Partition: *RibbonName, SheetName, SignatureName, WebName*
Input of processes: Many
Output of processes: Many

Resource Structure

Name	Data Type	Description
ComponentType	enumeration	Specifies the category of the component. Possible values are: *Ribbon* *Sheet* *Block* *PartialProduct* *FinalProduct*
Dimensions ?	shape	The dimensions of the component. These dimensions differ from the original size of the original product. For example, the dimensions of a folded sheet may not be equal to the dimensions of the sheet before it was folded. The dimension is always the bounding box around the **Component**. Default = 0 0 0, which specifies unknown. In this case a portrait orientation (Y>X) is assumed Note: It is crucial for postpress to specify the dimensions unless they really are unknown.
IsWaste ?	boolean	If true, the component waste may be used to set up a machine. Default = *false*
MaxHeat ?	double	Maximum temperature the **Component** can resist (in degrees centigrade). Default = no restriction in terms of heat, e.g., fusers in electrophotographic process or shrink wrapping.
Overfold ? New in JDF 1.1	double	Expansion of the overfold of a **Component**. This attribute may be needed for the ***Inserting*** or other postpress processes. Default = 0
OverfoldSide ? New in JDF 1.1	enumeration	Specifies the longer side of a folded component. One of "*Front"* or "*Back"*. Default = *Front*
ProductType ?	NMTOKEN	Type of product that this component specifies. Possible values include: *BackCover* *Book* *BookBlock* *BookCase* *Box* – Convenience packaging that is not envisioned to be protection for shipping. *Brochure* *BusinessCard* *Carton* – Protection packaging for shipping. *Cover* *FrontCover* *Jacket* – Hard cover case jacket. *Label* *Poster* Default = *unknown*
ReaderPageCount ? New in JDF 1.1	integer	Total Amount of individual reader pages that this **Component** contains. Count of –1 means "unknown." Default = -1, i.e., unknown.

Name	Data Type	Description
SheetPart ?	rectangle	Only useful when *ComponentType* = *Block* and when *Source-Sheet* is present. Part of the Sheet in *SurfaceContentsBox* coordinates used in this **Component**.
SourceRibbon ?	string	Only required when *ComponentType* = *Ribbon*. *RibbonName* of the ribbon used in this **Component**.
SourceSheet ?	string	Only required when *ComponentType* = *Sheet* or *Block*. *Sheet-Name* of the sheet used in this **Component**.
SourceWeb ?	string	Only required when *ComponentType* = *Ribbon*. *WebName* of the ribbon used in this **Component**.
SurfaceCount ? New in JDF 1.1	integer	Total Amount of individual surfaces that this **Component** contains. Count of –1 means "unknown." Default =-1, i.e., unknown
Transformation ? Deprecated in JDF 1.1	matrix	Matrix describing the transformation of the orientation of a component for the process using this resource as input. This is needed to convert the coordinate system of the component to the coordinate system of the process. When this attribute is not present, the identity matrix (1 0 0 1 0 0) is assumed. In version 1.1 and beyond, use ResourceLink::*Transformation* or ResourceLink::*Orientation.*
Bundle ? New in JDF 1.1	refelement	Description of a bundle of Components if the Component represents multiple individual items. If no Bundle is present, the Component represents an individual item. Note that it is essential to keep a reference of the child **Components** that comprise a **Component**, as this information is useful to postpress processes.
Disjointing ?	refelement	A stack of components can be processed using physical separators. This is useful in operations such as feeding. Default = no physical separators
Sheet ?	refelement	The **Sheet** resource that describes the details of this **Component** if *ComponentType* = *Sheet* or *Block*.

7.2.28 Contact

Element describing a contact to a person or address.

Resource Properties

Resource class: Parameter
Resource referenced by: **ApprovalParams**, , **ArtDeliveryIntent**, **DeliveryIntent**, **DeliveryParams**, **DropIntent**
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
ContactTypes	NMTOKENS	Classification of the contact. Possible values include:
		Administrator – Person to contact for queries concerning the execution of the job.
		Accounting – Address of where to send to the bill.
		Billing – Contact information that refers to a payment method, e.g., credit card.
		Customer – The end customer.
		Delivery – Delivery address for all products of this job.
		Owner – The owner of a resource.
		Pickup – The pickup address for all products of this job.
		Supplier – Address of a supplier of needed goods.
		SurplusReturn – Return delivery or pickup address for surplus products of this job.
		ArtReturn – Return delivery or pickup address for artwork of this job.
Address ?	refelement	Element describing the address.
ComChannel *	refelement	Communication channels to the contact.
Company ? New in JDF 1.1	refelement	Company that this **Contact** is associated with.
Person ?	refelement	Name of the contact person.

7.2.29 ContactCopyParams

New in JDF 1.1

Element describing the parameters of 6.3.3ContactCopying.

Resource Properties

Resource class: Parameter
Resource referenced by: *ContactCopying*.
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
ContactScreen ?	boolean	True, if a halftone screen on film should be used to produce half-tones. Default="*false*".
Cycle ?	integer	Number of exposure light units to be used. The amount depends on the subject to be exposed.
Diffusion ?	enumeration	The diffusion foil setting. Possible values are: *On* *Off*
PolarityChange ?	boolean	True, if the copy should change polarity w.r.t. the original image. Default="*true*".
RepeatStep ?	XYPair	Number of copies in each direction for a Step/Repeat camera. Default = 1 1
Vacuum ?	double	Amount of vacuum pressure to be used. Measured in bars.

Name	Data Type	Description
ScreeningParams ?	refelement	Properties of the halftone screen on film. Ignored if *ContactScreen* ="*false*".

7.2.30 ConventionalPrintingParams

This resource defines the attributes and elements of the **ConventionalPrinting** process. The specific parameters of individual printer modules are modeled by using the standard partitioning methods. These methods are described in Section 3.9.2.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *BlockName, FountainNumber, RibbonName, Separation, SheetName, Side, SignatureName, WebName*
Input of processes: *ConventionalPrinting*
Output of processes: -

Resource Structure

Name	Data Type	Description
DirectProof ?	boolean	If *true*, the proof is directly produced and subsequently an approval may be given by a person such as the customer, foreman, or floor manager shortly after the first final-quality printed sheet is printed. The approval is not required for setup, but it is required for the actual print run. If the **ConventionalPrinting** process is waiting for a *DirectProof,* its *Status* is switched to *Stopped* with the *StatusDetails* = *WaitForApproval*. Default = *false*
Drying ?	enumeration	The way in which ink is dried after a print run. Possible values are: *UV* – Ultraviolet dryer *Heatset* – Heatset dryer *IR* – Infrared dryer *On* – Use the device default drying unit. *Off* – Default value.
FirstSurface ?	enumeration	Printing order of the surfaces on the sheet. Possible values are: *Either* – Default value. The printer may choose. *Front* *Back*
FountainSolution ?	enumeration	State of the fountain solution module in the printing units. Possible values are: *On* *Off* If not specified use the system specified setting, which may be either *On* or *Off*.

Name	Data Type	Description
MediaLocation ?	string	Identifies the location of the **Media**. The value identifies a physical location on the press, such as unwinder 1, unwinder 2, and unwinder 3.
		If the media resource is partitioned by *Location* (see also Section 3.9.2.2 Locations of Physical Resources) there should be a match between one *Location* partition key and this *MediaLocation* value.
ModuleAvailableIndex ? New in JDF 1.1	IntegerRange-List	Zero-based list of print modules that are available for printing. In some cases modules are not available because the print module is replaced with in-line tooling, e.g. a perforating unit. Default = 0~-1, i.e., all modules are used for printing. The list is based on all modules of the printer and is not influenced by the value of *ModuleIndex*.
ModuleDrying ?	enumeration	The way in which ink is dried in individual modules. Possible values are:
		UV – Ultraviolet dryer
		Heatset – Heatset dryer
		IR – Infrared dryer
		On – Use the device default drying unit.
		Off – *The d*efault.
ModuleIndex ?	IntegerRange-List	Zero-based, ordered list of print modules that are used. The list is based on all modules of the printer and is not influenced by the value of *ModuleAvailableIndex*. Defaults to system specified.
PerfectingModule ? New in JDF 1.1	integer	Index of the perfecting module if **WorkStyle =** *Perfecting* and multiple perfecting modules are installed. Default = 0, i.e., the first installed perfecting module.
Powder ?	double	Quantity of powder (in %).
PrintingType	enumeration	Type of printing machine. Possible values are:
		SheetFed
		WebFed
		The principal difference between *SheetFed* and *WebFed* is the shape of the paper each is equipped to accept. Presses that execute *WebFed* processes use substrates that are continuous and cut after printing is accomplished. Most newspapers are printed on web-fed presses. *SheetFed* printing, on the other hand, accepts precut substrates.
SheetLay ?	enumeration	Lay of input media. Reference edge of where paper is placed in feeder. Possible values are:
		Left
		Right
		Center
		Default is the system specified value.
Speed ?	number	Maximum print speed in sheets/hour (sheet fed) or meters/hour (web fed). Defaults to device specific full speed.
WorkStyle ?	enumeration	The direction in which to turn. Possible values are:
		Simplex – No turning
		WorkAndBack – This **WorkStyle** describes the printing on both

Name	Data Type	Description
		sides of the substrate with a different plate (set) in the second run. After the first run the side lays are altered but the front lays stay as they were. Lays can be turned by hand or using a pile reverser. Two-plate sets are necessary for *WorkAndBack*.
		Perfecting – Many sheetfed printing presses have perfecting cylinder(s) built in. The leading edge of the print sheet changes as the sheet is turned by the perfecting cylinder, but the side lays remain unaltered. In this regard, this **WorkStyle** is similar to *WorkAndTumble*, but **Perfecting** is an in-line operation during the press run. Therefore, an additional plate (set) is required during this press run.
		WorkAndTurn – Refers to the turning of the first-run sheet for subsequent perfecting. The front lays remain unchanged but the side lays must be altered. The alteration can be made by hand or using a pile turner. The plate (set) stay(s) in the machine and, during each run, half of the surface is imaged.
		WorkAndTumble – The *WorkAndTumble* method is also used for perfecting. The leading edge of the print sheet changes as the sheet is turned, but the side lays remain unaltered. Tumbling happens after the first press run and the plate (set) is used again in the second press run, imaging the other sheet surface.
		WorkAndTwist – Done between two press runs. The palette is twisted 180 degree before the second run is performed so that the front lay and the side lay both change. The surface to be imaged is the same at both runs. Each run prints only part of the surface. The plate (set) stay in the machine. This **WorkStyle** is used for saving plate or film material. It is no longer a common **WorkStyle**.
Ink ?	refelement	Kind of varnishing. Defines the varnish to be used for coatings on printed sides. Coatings are applied after printing all the colors. Other coating sequences must use the partition mechanism of this parameter resource. Selective varnishing has to use a separate separation for the respective varnish.
		Note: The color inks are direct input resources of the **ConventionalPrinting** process.

7.2.31 CostCenter

This resource describes an individual area of a company that has separated accounting.

Resource Properties

Resource class: ResourceElement
Resource referenced by: **Device**, **Employee**
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
CostCenterID	string	Identification of the cost center
Name ?	string	Name of the cost center.

7.2.32 CoverApplicationParams

New in JDF 1.1

CoverApplicationParams define the parameters for applying a cover to a book block.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *CoverApplication*
Output of processes: -

Resource Structure

Name	Data Type	Description
CoverOffset	XYPair	Position of the cover in relation to the book block given in the cover-sheet coordinate system.
GlueApplication *	refelement	Describes where and how to apply glue to the book block.
Score *	element	Describes where and how to score the cover.

Structure of Score Subelement

Name	Data Type	Description
Offset	double	Position of scoring given in the operation coordinate system.
Side	enumeration	Specifies the side from which the scoring tool works. Possible values are: *FromInside* – The default. *FromOutside*
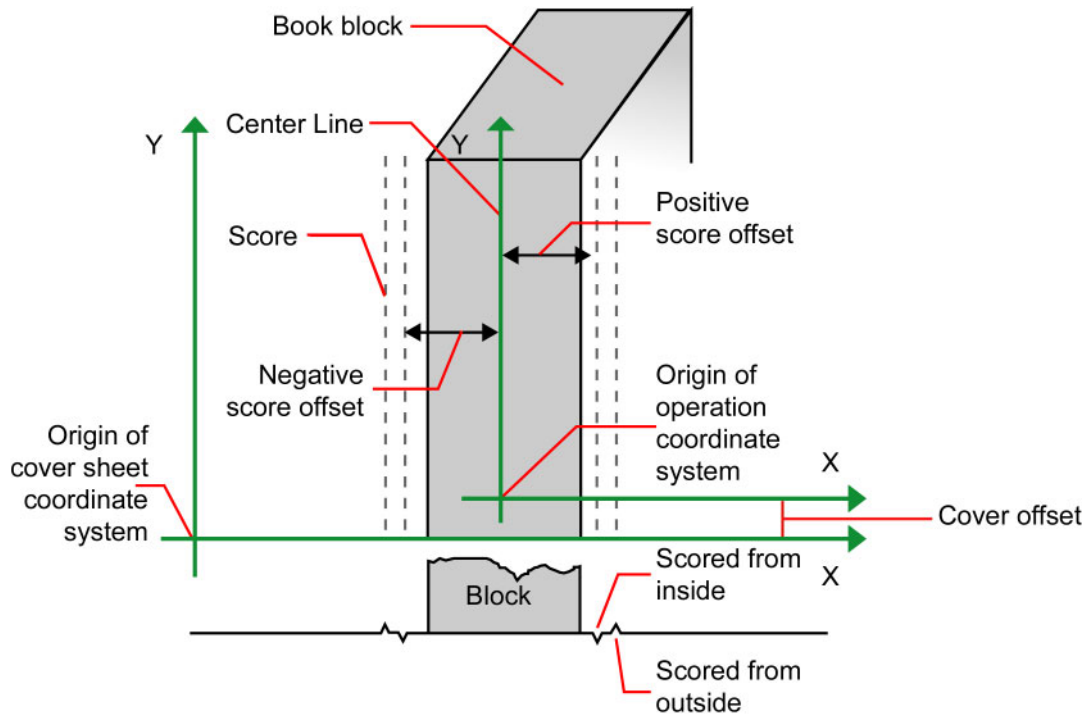
Figure 7.7 Parameters and coordinate system for cover application

7.2.33 CreasingParams

New in JDF 1.1

CreasingParams define the parameters for creasing or grooving a sheet .

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *BlockName, RibbonName, SheetName, SignatureName, WebName*
Input of processes: *Creasing*
Output of processes: -

Resource Structure

Name	Data Type	Description
Crease *	element	definition of one or more Crease lines.

Crease

Crease defines an individual crease line on a Component.

Name	Data Type	Description
StartPosition	XYPair	Starting position of the tool.
WorkingPath	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
WorkingDirection	enumeration	Direction from which the tool is working. Possible values are: *Top* – from above *Bottom* – from below

7.2.34 CutBlock

Defines a cut block on a sheet. It is possible to define a block that contains a matrix of elements of equal size. In this scenario, the intermediate cut dimension is calculated from the information about element size, block size and the number of elements in both directions. Each cut block has its own coordinate system, which is defined by the *BlockTrf* attribute.

Resource Properties

Resource class: Parameter
Resource referenced by: CuttingParams
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
BlockElementSize ?	XYPair	Element dimension in X and Y direction. Default = *BlockSize*
BlockElementType ?	enumeration	Element type. Possible values are: *CutElement* – Cutting element. *PunchElement* – Punching element. Default = system specified
BlockName	NMTOKEN	Name of the block. Used for reference by the **CutMark** resource. Note that **CutBlock** resources are not partitioned although they are nested. The semantics of nested **CutBlock**s is different.
BlockSize	XYPair	Size of the block.
BlockSubdivision ?	XYPair	Number of elements in X and Y direction. Default = (1,1,) i.e., no subdivision.
BlockTrf	matrix	Block transformation matrix. Defines the position and orientation of the block relative to the **Component** coordinate system. Default = identity
BlockType	enumeration	Block type. Possible values are: *CutBlock* – Block to be cut. *SaveBlock* – Protected block, cut only via outer contour. *TempBlock* – Auxiliary block that is not taken into account during cutting. *MarkBlock* – Contains no elements, only marks.

7.2.35 CutMark

This resource, along with **CutBlock**, provides the means to position cut marks on the sheet. After printing, these marks can be used to adapt the theoretical block positions (as specified in **CutBlock**) to the real position of the corresponding blocks on the printed sheet.

Resource Properties

Resource class: Parameter
Resource referenced by: CuttingParams, Surface
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Blocks ? Modified in JDF 1.1	NMTOKENS	Values of the *BlockName* partition attributes of the blocks defined by the **CutMark** resource.
MarkType	enumeration	Mark type. Possible values are: *CrossCutMark* *TopVerticalCutMark* *BottomVerticalCutMark* *LeftHorizontalCutMark* *RightHorizontalCutMark* *LowerLeftCutMark* *UpperLeftCutMark* *LowerRightCutMark* *UpperRightCutMark*
Position	XYPair	Position of the logical center of the cut mark in the coordinates of the MarkObject that contains this mark. Note: The logical center of the cut mark does not always directly specify the center of the visible cut mark symbol.

Symbol	Name	Position of symbol
	CrossCutMark	Centered at logical position
	TopVerticalCutMark	Slightly above logical position
	BottomVerticalCutMark	Slightly below logical position
	LeftHorizonalCutMark	Slightly to the left of logical position
	RightHorizonalCutMark	Slightly to the right of logical position
	LowerLeftCutMark	Corner at logical position
	UpperLeftCutMark	Corner at logical position
	LowerRightCutMark	Corner at logical position
	UpperRightCutMark	Corner at logical position

Figure 7.8 Cut mark types

7.2.36 CuttingParams

New in JDF 1.1
This resource describes the parameters of a *Cutting* process that uses nested **CutBlock**s as input.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *BlockName, RibbonName, SheetName, SignatureName, WebName*
Input of processes: *Cutting*
Output of processes: -

Resource Structure

Name	Data Type	Description
CutBlock *	refelement	One or several **CutBlock**s can be used to find the *Cutting* sequence. Only one of **CutBlock** or **Cut** may be specified.
CutMark *	refelement	**CutMark** resources can be used to adapt the theoretical cut positions to the real positions of the corresponding blocks on the **Component** to be cut.
Cut *	element	**Cut** elements describe an individual cut. Only one of **CutBlock** or **Cut** may be specified.

Structure of the Cut Subelement

Cut describes one straight cut with an arbitrary tool.

Name	Data Type	Description
StartPosition	XYPair	Starting position of the tool.
WorkingPath	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
WorkingDirection	enumeration	Direction from which the tool is working. Possible values are: *Top* – from above *Bottom* – from below

7.2.37 DBMergeParams

This resource specifies the parameters of the **DBTemplateMerging** process.

Resource Properties

Resource class: Parameter
Resource references: -
Resource inheritance: -
Example Partition: -
Input of processes: *DBTemplateMerging*
Output of processes: -

Resource Structure

Name	Data Type	Description
SplitDocuments ?	integer	Indicates how often to split documents to create a new file.
FileSpec ?	refelement	URL of the generated destination file. This is most often a printable file type, such as PDF of PPML. If FileSpec is not specified, **DBMergeParams** must be a **Pipe** resource.

7.2.38 DBRules

This resource specifies the rules that should be applied to convert a database record into a graphic element. It is described by a text element with a human-readable description of the selection rules. For example:

```
insert the "Age" field behind the birthday;
if income>100,000 use Porsche.gif, else use bicycle.jpeg for image #2.
```

The internal representation of the mapping of database fields to graphic content within the document template is implementation-dependent. It can vary from fully variable, multi-page, automated document layout to simply inserting some line-feed characters between database records in an address field. Therefore, **DBRules** is defined as a simple human-readable text element.

Resource Properties

Resource class: Parameter
Resource references: -
Resource inheritance: -
Example Partition: -
Input of processes: *DBDocTemplateLayout*, *Inserting*, *Collecting*, *Gathering*
Output of processes: -

Resource Structure

Name	Data Type	Description
Comment +	telem	Human-readable description of the database rules that map database fields to image or text content.

7.2.39 DBSchema

This resource specifies the formal structure of a database record, regardless of type. It is encoded as a text element with a human-readable description of the database schema.

Resource Properties

Resource class:	Parameter
Resource references:	-
Resource inheritance:	-
Example Partition:	-
Input of processes:	*DBDocTemplateLayout*, *Verification*
Output of processes:	-

Resource Structure

Name	Data Type	Description
DBSchemaType	enumeration	Database type. Possible values are: *CommaDelimited* *SQL* *XML*
Comment +	telem	Human-readable description of the database schema.

7.2.40 DBSelection

This resource specifies a selection of records from a database.

Resource Properties

Resource class:	Parameter
Resource references:	-
Resource inheritance:	-
Example Partition:	-
Input of processes:	*DBTemplateMerging*, *Inserting*, *Collecting*, *Gathering*, *Verification*
Output of processes:	*Verification*

Resource Structure

Name	Data Type	Description
DataBase	URL	URL of the database
Records ?	IntegerRangeList	The indices of the database records.
Select ?	string	Database selection criteria in the native language of the database, e.g., SQL.

7.2.41 DeliveryParams

Provides information needed by a *Delivery* process. A *Delivery* process consists of sending a quantity of a product to a specific location at, in some cases, a required date and time.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Example Partition:	-

Input of processes: *Delivery*
Output of processes: -

Resource Structure

Name	Data Type	Description
Earliest ?	dateTime	Specifies the earliest time after which the delivery may be made.
Method ?	string	Identifies a required delivery method, such as *ExpressMail* or *InterofficeMail*.
Pickup ?	boolean	If *true*, the merchandise is picked up. If *false*, the merchandise is delivered. Default = *false*
Required ?	dateTime	Specifies the time by which the delivery must be made.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the addressee.
Contact * New in JDF 1.1	refelement	Address and further information of the **Contact** responsible for this delivery.
Drop +	element	All locations where the product will be delivered.

Structure of the Drop Subelement

Name	Data Type	Description
Earliest ?	dateTime	Specified the earliest time after which the delivery may be made. Default = *Earliest* in the root DeliveryParams.
Method ?	string	Identifies a required delivery method, such as *ExpressMail* or *InterofficeMail*. Default = *Method* in the root DeliveryParams.
Pickup ?	boolean	If *true*, the merchandise is picked up. If *false*, the merchandise is delivered. Default = *Pickup* in the root DeliveryParams.
Required ?	dateTime	Specifies the time by which the delivery must be made. Default = *Required* in the root DeliveryParams.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the addressee. Defaults to the value of Company specified in the root **DeliveryParams** resource.
Contact * New in JDF 1.1	refelement	Address and further information of the **Contact** responsible for this delivery.
DropItem +	element	A Drop may consist of multiple products, which are represented by their respective **Component** resources. Each DropItem describes an individual resource that is part of this Drop.

Structure of the DropItem Subelement

Name	Data Type	Description
Amount ?	integer	Specifies the number of **Components** ordered. If *Amount* is not specified, defaults to the total amount of the **Component** that is referenced by *rRef*.
Unit ?	string	Unit of measurement for the *Amount* specified in *ComponentLink*. Defaults to the value of *Unit* defined in the **Component** resource linked by *rRef*.
Component	refelement	Description of the individual item.

7.2.42 DensityMeasuringField

This resource contains information about a density measuring field.

Resource Properties

Resource class: Parameter
Resource referenced by: **ColorControlStrip**, **Surface**
Example Partition: -
Input of processes: *Any printing process*
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the density measuring field in the coordinates of the MarkObject that contains this mark. If the measuring field is defined within a **ColorControlStrip**, *Center* refers to the rectangle defined by *Center* and *Size* of the **ColorControlStrip**.
Density `Modified in JDF 1.1A`	NumberList	Density value for each process color measured with filter. The data type was modified to NumberList in JDF 1.1A in order to accommodate density values >1.0.
Diameter	double	Diameter of measuring field.
DotGain	double	Percentage of dot gain.
Percentage	double	Film percentage or equivalent.
Screen	string	Description of the screen.
Separation	string	Reference to separation.
Setup ?	string	Description of measurement setup.
ToleranceCyan	XYPair	Upper and lower cyan tolerance (in density units).
ToleranceMagenta	XYPair	Upper and lower magenta tolerance (in density units).
ToleranceYellow	XYPair	Upper and lower yellow tolerance (in density units).
ToleranceBlack	XYPair	Upper and lower black tolerance (in density units).
ToleranceDotGain	XYPair	Upper and lower tolerance (in percentage).
ColorMeasurement-Conditions ? `New in JDF 1.1`	refelement	Detailed description of the measurement conditions for color measurements.

7.2.43 DevelopingParams

`New in JDF 1.1`

DevelopingParams specifies information about the chemical and physical properties of the developing and fixing process for film and plates. Includes details of preheating, postbaking and postexposure.

Preheating is necessary for negative working plates. It hardens the exposed areas of the plate to make it durable for the following developing process. The stability and uniformity of the preheat temperature influence the evenness of tints and the run length of the plate on press.

Postbaking is an optional process of heating that is applied to most polymer plates to enhance the run length of the plate. A factor 5 to 10 can be gained compared to plates that are not postbaked.

Postexposure is an optional exposure process for photopolymer plates to enhance the run length of the plate. A factor of 5 to 10 can be gained compared with plates that are not postexposed.

Note: Postbaking and postexposure are mutually exclusive.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ContactCopying, FilmToPlateCopying, ImageSetting*
Output of processes: -

Resource Structure

Name	Data Type	Description
PreHeatTemp ?	number	Temperature of the preheating process in °C. Default = 0, i.e., no preheating.
PreHeatTime ?	duration	Duration of the preheating process. Default = 0M, i.e., no preheating.
PostBakeTemp ?	number	Temperature of the post baking process in °C. Default = 0, i.e., no post process baking.
PostBakeTime ?	duration	Duration of the post baking process. Default = 0M, i.e., no post process baking.
PostExposeTime ?	duration	Duration of the post exposing process. Default = 0M, i.e., no post process baking. Note: Only one of **PostBakeTime** and **PostExposeTime** may be non-zero.

7.2.44 Device

Information about a specific device. This optionally includes information about the devices capabilities. For more information, see Section 3.7.1.3 Implementation Resources and 4.8 Describing Device Capabilities with JDF.

Resource Properties

Resource class: Implementation
Resource referenced by: -
Example Partition: -
Input of processes: any process
Output of processes: -

Resource Structure

Name	Data Type	Description
DeviceFamily? Deprecated in JDF 1.1	string	Manufacturer family type ID. **DeviceFamily** is replaced by the appropriate **ModelXXX** attributes in this list.
DeviceID	string	Name of the device. This is a unique name within the workflow. Must be the same over time for a specific device instance, i.e., must survive reboots. Equivalent to the UPNP:UDN.
DeviceType ?	string	Manufacturer type ID, including a revision stamp.
Directory ? New in JDF 1.1	URL	Defines a directory where the URLs that are associated with this Device can be located. If **Directory** is not specified, all **URLs** must be completely specified.
FriendlyName ? New in JDF 1.1	string	Short user-friendly title
JDFVersions ? New in JDF 1.1	string	Whitespace separated list of supported JDF versions that this device supports, e.g, "1.0 1.1" specifies that both the 1.0 and 1.1 version are supported.
JMFSenderID ? New in JDF 1.1	string	ID of the controller will process JMF messages for the device. This corresponds to the SenderID attribute that must be specified for the device in JMF messages.
JMFURL ? New in JDF 1.1	URL	URL of the device port that will accept JMF messages.

Name	Data Type	Description
Manufacturer ? New in JDF 1.1	string	Manufacturer name
ManufacturerURL ? New in JDF 1.1	string	Web site for manufacturer
ModelDescription ? New in JDF 1.1	string	Long description for end user
ModelName ? New in JDF 1.1	string	Model name
ModelNumber ? New in JDF 1.1	string	Model number
ModelURL ? New in JDF 1.1	string	Web site for model.
SerialNumber ? New in JDF 1.1	string	Serial number of the device.
PresentationURL ? New in JDF 1.1	string	URL to presentation for device It is a URL to a device-provided UI for user configuration, status, etc. Thus, if the device has an embedded Web server, this is a URL to the configuration page hosted on that Web server.
UPC ? New in JDF 1.1	string	Universal Product Code for the device. A 12 -digit, all-numeric code that identifies the consumer package. Managed by the Uniform Code
CostCenter ?	element	MIS cost center ID.
DeviceCap * New in JDF 1.1	element	Description of the capabilities of the device. The **DeviceCap** elements are combined with a logical OR, i.e., if a JDF resides within any parameter space defined by a **DeviceCap**, the device can process the job. For details see 7.3 Device Capability Definitions.
IconList ? New in JDF 1.1	element	List of locations of icons that can be used to represent the device.

Structure of the IconList Subelement
New in JDF 1.1

The **IconList** is a list of individual icon descriptions.

Name	Data Type	Description
Icon +	refelement	Individual icon description.

Structure of the Icon Subelement
New in JDF 1.1

An **Icon** represents a device in the user interface.

Name	Data Type	Description
Size	XYPair	Height and width of the icon.
BitDepth	integer	Bit depth of one color

Name	Data Type	Description
IconUsage ?	enumerations	Definition of the *Status* of the device that this Icon represents. Any combination of: *Unknown – N*o link to the device exists *Idle* *Down* *Setup* *Running* *Cleanup* *Stopped* Defaults to all of the above. The meaning of the individual enumerations is described in the DeviceInfo message element. See 5.5.1.3 KnownDevices
FileSpec	element	Details of the file containing the icon data.

7.2.45 DigitalPrintingParams

This resource contains attributes and elements used in executing the **DigitalPrinting** process. The *PrintingType* attribute in this resource defines two types of printing: *SheetFed* and *WebFed*. The principal difference between them is the shape of the paper each is equipped to accept. Presses that execute *WebFed* processes use substrates that are continuous and cut after printing is accomplished. Most newspapers are printed on web-fed presses. *SheetFed* printing, on the other hand, accepts precut substrates.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *BlockName, DocRunIndex, DocSheetIndex, PartVersion, Run, RunIndex, RunTag, Sheet-Index, Separation, SheetName, Side, SignatureName, DocIndex*

Input of processes: *DigitalPrinting*
Output of processes: -

Resource Structure

Name	Data Type	Description
Collate ? New in JDF 1.1	enumeration	Determines the sequencing of the sheets in the document and the documents in the job when multiple copies of a document or a job are requested as output. Document copies can be requested by specifying **RunList**:*DocCopies* and job copies can be requested by specifying the output **Component** *Amount*. *None* – Do not collate sheets in the document or document(s) in the job. *Sheet* – Collate the sheets in each document; do not collate the documents in the job. The result of *Sheet* and *SheetAndSet* is the same when there is one document in the set. The result of *Sheet* and *SheetSetAndJob* is the same when there is one document in the set and one set in the job. *SheetAndSet* – Collate the sheets in the document and collate the documents in the set. Do not collate the sets in the job. The result of *SheetAndSet* and *SheetSetAnd-Job* is the same when there is one set in the job. *SheetSetAndJob* – Collate the sheets in the document and collate the documents in the set and collate the sets in the job. *SystemSpecified* – Collate as defined by system Default = *SystemSpecified* The following example consists of 2 documents, A and B, each having 2 sheets,

Name	Data Type	Description
		A1, A2 and B1, B2. The number of document copies requested is 1 for both documents and the number of job copies requested is 3 (Component Amount=3). The job contains no document set boundaries. If Collate=None, the sheet order will be: A1A1A1 A2A2A2 B1B1B1 B2B2B2 If Collate=Sheet, the sheet order will be: A1A2 A1A2 A1A2 B1B2 B1B2 B1B2 If Collate=SheetAndSet or SheetSetAndJob, the sheet order will be: A1A2 B1B2 A1A2 B1B2 A1A2 B1B2
ManualFeed ? New in JDF 1.1	boolean	Indicates whether the media will be fed manually. Default = *false*
OutputBin ? New in JDF 1.1	NMTOKEN	Specifies the bin to which the finished document should be output. Possible values include: *Top* – The bin that, when facing the device, can best be identified as 'top'. *Middle* – The bin that, when facing the device, can best be identified as 'middle'. *Bottom* – The bin that, when facing the device, can best be identified as 'bottom'. *Side* – The bin that, when facing the device, can best be identified as 'side'. *Left* – The bin that, when facing the device, can best be identified as 'left'. *Right* – The bin that, when facing the device, can best be identified as 'right'. *Center* – The bin that, when facing the device, can best be identified as 'center'. *Rear* – The bin that, when facing the device, can best be identified as 'rear'. *FaceUp* – The bin that can best be identified as 'face up' with respect to the device. *FaceDown* –The bin that can best be identified as 'face down' with respect to the device. *FitMedia* – Requests the device to select a bin based on the size of the media. *LargeCapacity* – The bin that can best be identified as the 'large capacity' bin (in terms of the number of sheets) with respect to the device. *Mailbox-N* – The job will be output to the bin that is best identified as 'Mailbox-1', 'Mailbox-2'…etc. *Stacker-N* –The job will be output to the bin that is best identified as 'Stacker-1', 'Stacker-2' …etc. *Tray-N* – The job will be output to the tray that is best identified as 'Tray-1', 'Tray-2' … etc. SystemSpecified – The job will be output to the tray that is defined by the system. Default = *SystemSpecified*
PageDelivery ? New in JDF 1.1	enumeration	Indicates how pages are to be delivered to the output bin or finisher. Possible values are: *FanFold* – The output is alternating face-up, face down. *SameOrderFaceUp* – Order as defined by the RunList, with the "front" sides of the media up. *SameOrderFaceDown* – Order as defined by the RunList, with the "front" sides of the media up. *ReverseOrderFaceUp* – Order reversed, as defined by the RunList, with the "front" sides of the media up. *ReverseOrderFaceDown* – Order reversed, as defined by the RunList, with the

Name	Data Type	Description
		"front" sides of the media down.
		SystemSpecified – Order and face-up/face-down as defined by the system.
		Default = *SystemSpecified*
PrintingType ? Modified in JDF 1.1	enumeration	Type of printing machine. Possible values are: *SheetFed* *WebFed* *SystemSpecified* Default = *SystemSpecified*
PrintQuality ? Deprecated in JDF 1.1	enumeration	Indicates how pages are to be delivered to the output bin or finisher. Possible values are: *High* – Highest quality available on the printer. *Normal* – The default quality provided by the printer. *Draft* – Lowest quality available on the printer Default = *SystemSpecified* Replaced by **InterpretingParams**:*PrintQuality*
SheetLay ?	enumeration	Lay of input media. Reference edge of where paper is placed in feeder. Possible values are: *Left* *Right* *Center* *SystemSpecified* = The device-specific machine default Default = *SystemSpecified*
Component ? New in JDF 1.1	refelement	Describes the preprocessed media to be used. For any given partition, only one of **Media** or **Component** may be specified.
Disjointing ? New in JDF 1.1	refelement	Describes how individual components are separated from one another in the output bin.
Media ? New in JDF 1.1	refelement	Describes the media to be used. For any given partition, only one of **Media** or **Component** may be specified.
MediaSource ? Deprecated in JDF 1.1	refelement	Describes the media to be used. For any given partition, only one of **MediaSource** or **Component** may be specified.

7.2.46 Disjointing

The **Disjointing** resource describes how individual components are separated from one another on a stack.

Resource Properties

Resource class: Parameter
Resource referenced by: **Component**, **DigitalPrintingParams GatheringParams**
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Number ?	integer	Number of sheets that make up one component. Default = -1, i.e. unknown
Offset ?	XYPair	Offset dimension in X- and Y-dimensions that separates the components. Default = system specified.
OffsetAmount ?	integer	The number of components that are shifted in *OffsetDirection* simultaneously. Default = 1
OffsetDirection ?	enumeration	Offset-shift action for the first component. Possible values are: *Alternate* *Left* *None* – The default. *Right* *Straight* *SystemSpecified*
Overfold ? Deprecated in JDF 1.1	double	Expansion of the overfold of a sheet. This attribute may be needed for the **Inserting** or other postpress processes. Moved to **Component**.
IdentificationField * Modified in JDF 1.1	element	Marks that identify the range of sheets to be used in a process. A scanner will scan the sheets and detect a component boundary by scanning a mark, such as a bar code, that matches the description in the IdentificationField element.
InsertSheet ?	refelement	Some kind of physical marker (such as a paper strip or a yellow paper sheet) that separates the components. Default = no physical marker

7.2.47 DividingParams
Deprecated in JDF 1.1.
This resource contains attributes and elements used in executing the **Dividing** process.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *RibbonName, SheetName, SignatureName, WebName*
Input of processes:
Dividing
Output of processes: -

Resource Structure

Name	Data Type	Description
DividePositions	NumberList	Array containing the cross cut positions in y-direction (direction of web traveling).

7.2.48 EmbossingParams
New in JDF 1.1
This resource contains attributes and elements used in executing the **Embossing** process.

Resource Properties
Resource class: Parameter

Resource referenced by: -
Example Partition: *BlockName, RibbonName, SheetName, SignatureName, WebName*
Input of processes: *Embossing*
Output of processes: -

Resource Structure

Name	Data Type	Description
Emboss *	refelement	One Emboss element is specified for each impression.

Structure of the Emboss Subelement

Name	Data Type	Description
Direction	enumeration	The direction of the image. Possible values are: *Both* – Both debossing and embossing in one stamp. *Raised* – Embossing, *Depressed* – Debossing
EdgeAngle ?	number	The angle of a beveled edge in degrees. Typical values are an angle of: 30, 40, 45, 50 or 60 degrees. For *EdgeAngle* to exist, *EdgeShape = Beveled* must be specified.
EdgeShape ?	enumeration	The transition between the embossed surface and the surrounding media may be rounded or beveled (angled). Possible values are: *Rounded* – The default. *Beveled*
EmbossingType	enumeration	Possible values include *BlindEmbossing* – Embossed forms that are not inked or foiled. The color of the image is the same as the paper. *EmbossedFinish* – The overall design or pattern impressed in laminated paper when passed between metal rolls engraved with the desired pattern. Produced on a special embossing to create finishes such as linen. *FoilEmbossing* – Combines embossing with foil stamping in one single impression. *FoilStamping* – Using a heated die to place a metallic or pigmented image from a coated foil on the paper. *RegisteredEmbossing* – Creates an embossed image that exactly registers to a printed image.
Height ?	number	The height of the levels. This value specifies the *vertical* distance between the highest and lowest point of the stamp, regardless of the value of **Direction**.
ImageSize ?	XYPair	The size of the bounding box of one single image.
Level ?	enumeration	The level of embossing. Possible values are: *SingleLevel* *MultiLevel* *Sculpted*
Position ?	XYPair	Position of the lower left corner of the bounding box of the embossed image in the coordinate system of the Component.

7.2.49 Employee

Information about a specific device or machine operator (see Section 3.7.1.3 Implementation Resources). **Employee** is also used to describe the contact person who is responsible for executing a node, as defined in the NodeInfo field of a JDF node.

Resource Properties

Resource class: Implementation
Resource referenced by: -
Example Partition: -
Input of processes: Any process
Output of processes: -

Resource Structure

Name	Data Type	Description
PersonalID ?	string	ID of the relevant MIS employee.
Shift ?	string	Defines the shift to which the employee belongs.
CostCenter ?	element	MIS cost center ID.
Person ?	refelement	Describes the employee. If no Person element is specified, the **Employee** resource represents any employee who fulfills the selection criteria.

7.2.50 EndSheetGluingParams

This resource describes the attributes and elements used in executing the **EndSheetGluing** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *EndSheetGluing*
Output of processes: -

Resource Structure

Name	Data Type	Description
EndSheet (Front)	element	Information about the front-end sheet. The *Side* attribute of this element must be *Front*.
EndSheet (Back)	element	Information about the back-end sheet. The *Side* attribute of this element must be *Back*.

Structure of EndSheetGluingParams Elements

EndSheet

Name	Data Type	Description
Offset	XYPair	Offset of end sheet in X and Y direction.
Side	enumeration	Location of the end sheet. Possible values are: *Front* *Back*
GlueLine	element	Description of the glue line.
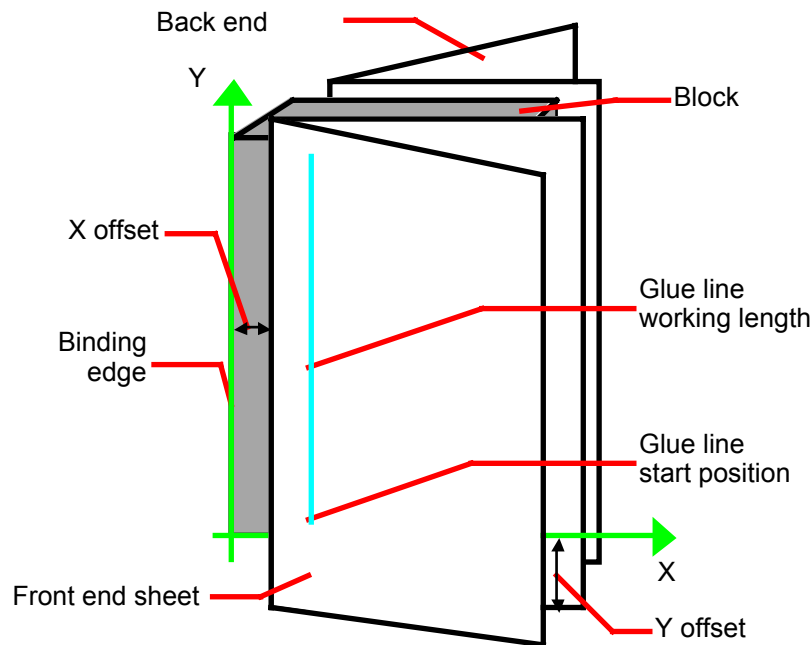
Figure 7.9 Parameters and coordinate system used for end-sheet gluing

The process coordinate system is defined as follows: The y-axis is aligned with the binding edge of the book block. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge, i.e., the product front edge.

7.2.51 ExposedMedia

This resource represents a processed **Media**-based handling resource such as film, plate, or paper proof. It is also used as an input resource for the *Scanning* process.

Resource Properties

Resource class: Handling
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, Separation, SheetName, Side, SignatureName, TileID, WebName*
Input of processes: *ContactCopying, ConventionalPrinting*, *PreviewGeneration*, *DigitalPrinting*, *Scanning*
Output of processes: *ContactCopying, ImageSetting*, *FilmToPlateCopying*, *Proofing*

Resource Structure

Name	Data Type	Description
ColorType ?	enumeration	Possible values are: *Color* *GrayScale* *Monochrome* – Black and white.
Polarity ?	boolean	*False* if the media contains a negative image. Default = *true*

Name	Data Type	Description
ProofQuality ?	enumeration	This attribute is present if the **ExposedMedia** resource describes a proof. Possible values are: *None* – Not a proof or the quality is unknown. Default value. *Halftone* – The halftones are emulated. *Contone* – No halftones, but exact color. *Conceptual* – Color does not match precisely.
ProofType ?	enumeration	*None* – Not a proof or the type is unknown. Default value. *Page* – Page proof *Imposition* – Imposition proof.
PunchType ?	string	Name of the registration punch scheme. Possible values include, but are not limited to: *Bacher* *Stoesser* Default = no punch holes.
Resolution ?	XYPair	Resolution of the output.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the output process for which this media was exposed. The *ResourceUsage* attribute of the FileSpec must be "Output*Profile*".
Media	refelement	Describes media specifics such as size and type.
ScreeningParams ?	refelement	Used to describe the screening in case of rasterized media

7.2.52 FileSpec

Specification of a file or a set of files.

Resource Properties

Resource class: Parameter
Resource referenced by: **DBMergeParams**, **LayoutElement**, **PDLResourceAlias**, **ScanParams**
Example Partition: *Separation*
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Application ?	string	Creator application, such as Photoshop.
AppOS ?	enumeration	Operating system of the application that created the file. Possible values are: *Unknown* – Default value *Mac* *Windows* *Linux* *Solaris* *IRIX* *DG_UX* *HP_UX*
AppVersion ?	string	Version of the value of the *Application* attribute.

Name	Data Type	Description
CheckSum ? New in JDF 1.1 Modified in JDF 1.1A	hexBinary	Checksum of the file being referenced using the RSA MD5 algorithm. In JDF 1.1a, the term RSA MD was completed to RSA MD5. The data type was modified to hexBinary to accommodate the 128 bit output of the MD5 algorithm.
Compression ?	enumeration	Indicates how the file is compressed. Possible values are: *None* – The file is not compressed. Default value. *Deflate* – The file is compressed using ZIP public domain compression (RFC 1951) *Gzip* – GNU zip compression technology (RFC 1952) *Compress* – UNIX compression (RFC 1977)
Disposition ?	enumeration	Indicates what the device should do with the file when the process that uses this resource as an input resource completes. Possible values are: *Unlink* – The device should release the file. *Delete* – The device should attempt to delete the file. *Retain* – The device should do nothing with the file. Default value.
DocumentNaturalLang ?	language	The natural language of the document this FileSpec refers to.
FileFormat ?	string	A formatting string used with the *Template* attribute to define a sequence of filenames in a batch process. If neither *URL* nor *UID* are present, both *FileFormat* and *FileTemplate* must be present, unless the resource is a pipe. For more information, see the text following this table.
FileSize ?	integer	Size of the file in Byte.
FileTemplate ?	string	A template, used with *FileFormat*, to define a sequence of filenames in a batch process. If neither *URL* nor *UID* is present, both *FileFormat* and *FileTemplate* must be present, unless the resource is a pipe.
FileVersion ? New in JDF 1.1	string	Version of the file referenced by this FileSpec.
MimeType ?	string	Mime type of the file.
OSVersion ?	string	Version of the operating system.
PageOrder ?	enumeration	Indicates whether the pages in the file are in reverse order. Possible values are: *Ascending* – The first page in the file is the lowest numbered page. *Descending* – The first page in the file is the highest numbered page.
ResourceUsage ?	NMTOKEN	If an element uses more than one FileSpec subelement, this attribute is used to refer from the parent element to a certain child element of this type, for example, see **ColorSpaceConversionParams**.
UID ? New in JDF 1.1	string	Unique internal ID of the referenced file. This attribute is dependent on the type of file that is referenced: PDF: Variable unique identifier in the ID field of the PDF file's trailer. ICC Profile: Profile ID in byte 84-99 of the ICC profile header. Others – Format specific.

Name	Data Type	Description
URL ?	URL	Location of the file. If *URL* is not present, and neither *FileFormat* nor *FileTemplate* are present, the referencing resource must be a pipe.
UserFileName ?	string	A user-friendly name which may be used to identify the file.
FileAlias *	element	Defines a set of mappings between file names that may occur in the document and URLs (which may refer to external files or parts of a MIME message).

Structure of FileAlias Subelement

Name	Data Type	Description
Alias	string	The filename which is expected to occur in the file.
Disposition	enumeration	Indicates what the device should do with the file referenced by this alias when the process that uses this resource as an input resource completes. Possible values are: *Unlink* – The device should release the file. *Delete* – The device should attempt to delete the file. *Retain* – The device should do nothing with the file.
MimeType ?	string	Mime type of the file.
URL	URL	The URL which identifies the file the alias refers to.

Usage of Format and Template

The function defined when using the attributes *FileFormat* and *FileTemplate* is drawn from the same root as the standard C print function and, therefore, overtly resembles the model of that function. *FileFormat* is the first argument and *FileTemplate* is a comma-separated list of the additional arguments. *FileTemplate* may contain the following operators : +,-,*,/,%,(,) which are evaluated using standard C-operator precedence and the variables defined in the following table:

Table 7-2 Predefined variables used in FileTemplate

Name	Description
element	Integer iterator over all elements in a given page. Restarts at 0 for each page.
i	Integer iterator over all files produced by this process. 0-based numbering.
page	Integer iterator over the page number of a document. This is equivalent to r for the case that each run contains exactly one page.
r	Integer iterator over all RunList partitions with a partition key of "*Run*" in an input **RunList**.
ri	Integer iterator over all indices in an input Run of a **RunList**. This index is equivalent to looping over a RunIndex.
sep	Separation as defined in the separation PartIDKey of a partitioned resource.
surf	Surface string, "Front" or "Back"
SheetName	SheetName string of a partitioned resource.
SignatureName	SignatureName string of a partitioned resource.
TileX	X coordinate of a Tile
TileY	Y coordinate of a Tile
PartVersion	PartVersion string of a partitioned resource.
jobPartID	JobPartID string
jobID	Job ID string

Name	Description
jobName	*DescriptiveName* of the Node that is being processed.
Time	Current *Time* in ISO 8601 format.
Date	Current *Date* in ISO 8601 format.
CustomerID	CustomerID

Example:

```
<FileSpec FileFormat = "file://here/next/%s/%4.i/m%4.i.pdf" FileTemplate = "Jo-
bID,i/100,i%100"/>
```

with JobID = "j001" and a **RunList** defining 2023 created files will iterate all created files and place them into:

```
"file://here/next/j001/0000/m0000.pdf"
```
…
```
"file://here/next/j001/0020/m0023.pdf"
```

7.2.53 FitPolicy

New in JDF 1.1

This resource specifies how to fit content into a receiving container, e.g., a **RunList** entry into a **PlacedObject,** or image onto media**.**

Resource Properties

Resource class: Parameter
Resource referenced by: **InterpretingParams, LayoutPreparationParams**
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
ClipOffset ?	XYPair	Defines the offset (position) of the imaged area in the non-rotated source image when *SizePolicy* is *ClipToMaxPage*. The values 0.0 0.0 mean that the imaged area starts at the lower left point of the job. If absent, the imaged area is taken from the center of the image source. If **FitPolicy** is defined in the context of a PageCell ClipOffset is ignored when PageCell::*ImageShift* is specified.
GutterPolicy ?	enumeration	Allows printing of NUp grids even if the media size does not match the requirements of the data. One of: *Distribute*: The gutters may grow or shrink to the value specified in *MinGutter.* *Fixed*: The gutters are fixed. The default.
MinGutter ?	XYPair	Minimum width in points of the horizontal and vertical gutters formed between rows and columns of pages of a multi-up sheet layout. The first value specifies the width of all horizontal gutters and the second value specifies the minimum width of all vertical gutters. If not specified, the gutter may be reduced to 0.
RotatePolicy ?	enumeration	Specifies the policy for the device to automatically rotate the image to optimize the fit of the image to the container. *NoRotate* – The default *RotateOrthogonal* – Rotate by 90° in either direction. *RotateClockwise* – Rotate clockwise by 90°. *RotateCounterClockwise* – Rotate counter-clockwise by 90°.

Name	Data Type	Description
SizePolicy ? Modified in JDF 1.1A	enumeration	Allows printing even if the container size does not match the requirements of the data. *ClipToMaxPage* – The page contents should be clipped to the size of the container. The printed area is either centered in the source image, if no *ClipOffset* key is given, or from that position which is determined by *ClipOffset*. *Abort* – Emit an error and abort printing. Default value. *FitToPage* – The page contents should be scaled up or down to fit the media. The aspect ratio is maintained. *ReduceToFit* – The page contents should be scaled down but not scaled up to fit the media. The aspect ratio is maintained. *Tile* – the page contents should be split into several tiles, each printed on its own surface.

7.2.54   Fold

New in JDF 1.1

Fold describes an individual folding operation of the **Component**.

Resource Properties

Resource class: Parameter
Resource referenced by: **FoldingIntent**, **FoldingParams**
Example Partition: -
Input of processes: *-*
Output of processes: -

Resource Structure

Name	Data Type	Description
From	enumeration	Edge from which the page is folded. Possible values are: *Front* *Left*
To	enumeration	Direction in which it is folded. Possible values are: *Up* – upwards *Down* – downwards
Travel	double	Distance of the reference edge relative to *From*

7.2.55 FoldingParams

This resource describes the folding parameters, including the sequence of folding steps. It is also possible to execute the predefined steps of the folding catalog. After each folding step of a folding procedure, the origin of the coordinate system is moved to the lower left corner of the intermediate folding product.

    The specification of reference edges (*Front*, *Rear*, *Left,* and *Right*) for the description of an operation (such as the positioning of a tool) is done by means of determined names. These names are case-sensitive. They must be written exactly as shown in Figure 7.9, below.
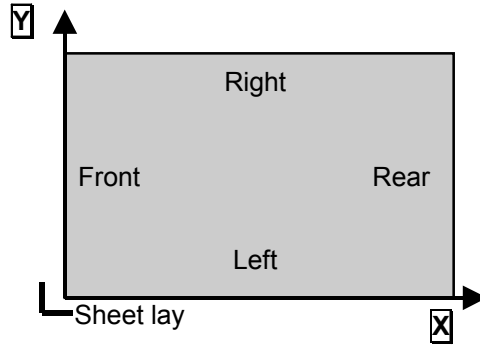
Figure 7.10 Names of the reference edges of a sheet in the FoldingParams resource

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *BlockName, RibbonName*, *SheetName, SignatureName,* W*ebName*
Input of processes: *Folding*
Output of processes: -

Resource Structure

Name	Data Type	Description
DescriptionType	enumeration	How the folding operations are described. Possible values are: *FoldProc* – Detailed description of each individual fold. *FoldCatalog* – Selection of fold procedure from *FoldCatalog.*
FoldCatalog ?	string	Description of the type of fold according to the folding catalog in the format "Fx-y" as shown in Figure 7.11. Required when *DescriptionType* = *FoldCatalog*.
FoldSheetIn ? Deprecated in JDF 1.1	XYPair	Input sheet format. If the specified size does not match the size of the *X* and *Y* dimensions of the input Component, all coordinates of the folding procedure are scaled accordingly. The scaling factors in X and Y direction may differ. Required when *DescriptionType* = *FoldProc*. Implementation note: This attribute should always match the *Size* attribute of the input component.
SheetLay ?	enumeration	Lay of input media. Possible values are: *Left* – The default. *Right*
Fold * New in JDF 1.1	element	This describes the folding operations in the sequence in which they should be carried out. At least one Fold is required when *DescriptionType* = *FoldProc*. It is recommended to specify a set of subsequent Fold operations as multiple Fold elements in one **Folding** procedure, rather than specifying a *Combined* process that combines multiple **Folding** processes.
FoldOperation * Deprecated in JDF 1.1	element	This describes the folding operations in the sequence in which they should be carried out. Replaced by Fold * in JDF 1.1.

F4-1 2x1	F6-1 3x1	F6-2 3x1	F6-3 3x1	F6-4 3x1
↑1/2	↑1/3 ↓1/3	↓1/3 ↑1/3	↑1/4 ↑1/2	↑1/3 ↑1/3

F6-5 3x1	F8-1 4x1	F8-2 4x1	F8-3 4x1	F8-4 4x1
↑2/3 ↓1/3	↑1/2 ↑1/4	↑1/2 ↓1/4	↑1/4 ↓1/4 ↑1/4	↑1/4 ↑1/2 ↓1/4

F8-5 4x1	F8-6 4x1	F8-7 2x2	F10-1 5x1	F10-2 5x1
↑1/4 ↑1/4 ↑1/4	↑3/4 ↓1/4 ↑1/4	↑1/2 +↑1/2	↑1/5 ↓1/5 ↑1/5 ↓1/5	↑4/5 ↓1/5 ↓1/5 ↓1/5

F10-3 5x1	F12-1 6x1	F12-2 6x1	F12-3 6x1	F12-4 6x1
↑2/5 ↓2/5 ↑1/5	↑1/3 ↓1/3 ↑1/6	↑1/3 ↑1/3 ↓1/6	↑1/2 ↓1/6 ↑1/6	↑1/2 ↓1/6 ↑1/6

F12-5 6x1	F12-6 6x1	F12-7 3x2	F12-8 3x2	F12-9 3x2
↑1/2 ↓1/3 ↑1/6	↑1/6 ↓1/6 ↑1/6 ↓1/6 ↑1/6	↑1/3 ↓1/3 +↑1/2	↑2/3 ↑1/3 +↑1/2	↑1/3 ↑1/3 + ↑1/2

F12-10 3x2	F12-11 3x2	F12-12 2x3	F12-13 2x3	F12-14 2x3
↑2/3 ↓1/3 +↑1/2	↑1/3 +↑1/2 ↑1/3	↑1/2 +↑2/3 ↓1/3	↑1/2 +↑1/3 ↑1/3	↑1/2 +↑1/3 ↓1/3

F14-1 7x1	F16-1 8x1	F16-2 8x1	F16-3 8x1	F16-4 8x1
↑1/7 ↓1/7 ↑1/7 ↓1/7 ↑1/7 ↓1/7	↑1/2 ↓1/4 ↑1/8	↑1/2 ↓1/4 ↓1/8	↑1/2 ↑1/4 ↓1/8	↑1/2 ↑1/4 ↑1/8

F16-5 8x1	F16-6 4x2	F16-7 4x2	F16-8 4x2	F16-9 4x2
↓1/8 ↑1/8 ↓1/8 ↑1/8 ↓1/8 ↑1/8 ↓1/8	↑1/2 + ↑1/2 + ↑1/4	↑1/2+ ↑1/2+ ↓1/4	↑1/2 +↓1/2 + ↓1/4	↑1/2 ↓1/4 + ↑1/2

F16-10 4x2	F16-11 4x2	F16-12 4x2	F16-13 2x4	F18-1 9x1
↑1/2 ↑1/4+ ↑1/2	↑1/4 ↓1/4 ↑1/4 +↑1/2	↑1/4 ↑1/4 ↑1/4 +↑1/2	↑1/2 +↑1/2 ↓1/4	↑1/9 ↓1/9 ↑1/9 ↓1/9 ↑1/9 ↓1/9 ↑1/9 ↓1/9

Figure 7.11 Fold Catalog part 1

F18-2 9x1	F18-3 9x1	F18-4 9x1	F18-5 3x3	F18-6 3x3
2 3 4 1	1 2 4 3	1 2 3 4	1 2 / 4 3	1 2 / 3 4
↑2/3 ↓1/3 ↑1/9 ↓1/9	↑1/3 ↓1/3 ↑2/9 ↓1/9	↑1/3 ↓1/3 ↑1/9 ↓1/9	↑1/3 ↓1/3 + ↑1/3 ↓1/3	↑1/3 ↓1/3 + ↑2/3 ↓1/3

F18-7 3x3	F18-8 3x3	F20-1 5x2	F20-2 5x2	F24-1 6x2
1 2 / 4 3	1 2 / 3 4	1 3 2 / 4	1 2 3 4 / 5	1 2 4 / 3
↑1/3 ↑1/3+ ↑1/3 ↓1/3	↑1/3 ↑1/3+ ↑2/3 ↓1/3	↑2/5 ↓2/5 ↑1/5 ↑1/2	↑1/5 ↓1/5 ↑1/5 ↓1/5 + ↑1/2	↑1/3 ↓1/3+ ↑1/2 + ↑1/6

F24-2 6x2	F24-3 6x2	F24-4 6x2	F24-5 6x2	F24-6 6x2
1 2 4 / 3	1 2 3 / 4	1 2 3 / 4	1 2 3 / 4	1 2 3 4 5 / 6
↑1/3 ↑1/3+ ↑1/2+ ↑1/6	↑1/3 ↓1/3 ↑1/6 +↑1/2	↑1/3 ↓1/3 ↑1/6 +↑1/2	↑1/3 ↑1/3 ↓1/6 +↑1/2	↑1/6 ↓1/6 ↑1/6 ↓1/6 ↑1/6 + ↑1/2

F24-7 6x2	F24-8 3x4	F24-9 3x4	F24-10 3x4	F28-1 7x2
1 3 4 / 2	1 2 / 4 3	2 1 / 4 3	1 2 / 4 3	1 2 3 4 5 6 / 7
↑1/3 + ↑1/2 + ↑1/3 ↓1/6	↑1/3 ↓1/3 + ↑1/2 ↓1/4	↑2/3 ↑1/3 + ↑1/2 ↓1/4	↑1/3 ↑1/3+ ↑1/2 ↓1/4	↑1/7 ↓1/7 ↑1/7 ↓1/7 ↑1/7 ↓1/7+ ↑1/2

F32-1 16x1	F32-2 8x2	F32-3 8x2	F32-4 4x4	F32-5 4x4
1 2 3 4	1 2 4 / 3	1 2 4 / 3	1 3 / 4 2	1 3 / 4 2
↑1/2 ↓1/4 ↑1/8 ↓1/16	↑1/2 ↓1/4 + ↑1/2+ ↑1/8	↑1/2 ↓1/4+ ↑1/2+ ↓1/8	↑1/2+ ↑1/2+ ↑1/4+ ↑1/4	↑1/2+ ↑1/2+ ↓1/4+ ↓1/4

F32-6 4x4	F32-7 4x4	F32-8 4x4	F32-9 4x4	F36-1 9x2
1 3 / 4 2	1 2 3 / 5 4	1 2 / 4 3	1 4 / 3 2	1 2 3 4 / 5
↑1/2 + ↑1/2 + ↑1/4 + ↓1/4	↑1/4 ↓1/4 ↑1/4 + ↑1/2 ↓1/4	↑1/2 ↓1/4 + ↑1/2 ↓1/4	↑1/2 + ↑1/2 ↓1/4+ ↑1/4	↑1/3 ↓1/3 ↑1/9 ↓1/9 + ↑1/2

F36-2 6x3	F40-1 5x4	F48-1 6x4	F48-2 4x6	F64-1 8x4
1 2 5 / 4 3	1 2 3 4 / 6 5	1 2 6 / 5 4 3	1 2 3 / 6 5 4	1 5 6 / 4 3 2
↑1/3 ↓1/3+ ↑1/3 ↓1/3 + ↑1/6	↑1/5 ↓1/5 ↑1/5 ↓1/5 + ↑1/2 ↓1/4	↑1/3 ↓1/3+ ↑1/4 ↓1/4 ↑1/4 + ↑1/6	↑1/4 ↓1/4 ↑1/4 + ↑1/3 ↓1/3 ↑1/6	↑1/2 + ↑1/4 ↓1/4 ↑1/4 + ↑1/4 ↓1/8

F64-2 8x4
1 2 3 7 / 6 5 4
↑1/4 ↑1/4 ↑1/4 + ↑1/4 ↑1/4 ↑1/4 + ↑1/8

Legend:

————— Fold up

- - - - - Fold down

▓ Finished format folded sheet

1, 2, 3... Folds in numeric order

∟ lay

green: open sheet length
red : open sheet width

Example: F32-3   8x2

- F32-3: Signature with 32 pages
- 8x2 : Split: 8 sheet parts lengthwise 2 sheet parts cross
- ↑ 1/2: Fold up with 1/2 of the open sheet format length
- ↓ 1/4: Fold down with 1/4 of the open sheet format length
- +   : Fold direction change: 90°
- ↑ 1/2: Fold up with 1/2 of the open sheet format
- +   : Fold direction change: 90°
- ↓ 1/8: Fold down with 1/8 of the open sheet format length

Figure 7.12 Fold Catalog part 2

7.2.56 FontParams

This resource describes how fonts must be handled when converting PostScript files to PDF.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *PSToPDFConversion*
Output of processes: -

Resource Structure

Name	Data Type	Description
AlwaysEmbed ?	NMTOKENS	One or more names of fonts that are always to be embedded in the PDF file. Each name must be the PostScript language name of the font. An entry that occurs in both the *AlwaysEmbed* and *NeverEmbed* lists constitutes an error. Default = an empty list.
CannotEmbedFontPolicy ?	enumeration	Determines what occurs when a font cannot be embedded. Possible values are: *Error* – Log an error and abort the process if any font can not be found or embedded. *Warning* – Warn and continue if any font cannot be found or embedded. The default. *OK* – Continue without warning or error if any font can not be found or embedded.
EmbedAllFonts ?	boolean	If *true*, specifies that all fonts, except those in the *NeverEmbed* list, are to be embedded in the PDF file. Default = false
MaxSubsetPct ?	integer	The maximum percentage of glyphs in a font that can be used before the entire font is embedded instead of a subset. This value is only used if *SubsetFonts* = *true*.
NeverEmbed ?	NMTOKENS	One or more names of fonts that are never to be embedded in the PDF file. Each name must be the PostScript language name of the font. An entry that occurs in both the *AlwaysEmbed* and *NeverEmbed* lists constitutes an error.
SubsetFonts?	boolean	If *true*, font subsetting is enabled. If *false*, it is not. Font subsetting embeds only those glyphs that are used, instead of the entire font. This reduces the size of a PDF file that contains embedded fonts. If font subsetting is enabled, the decision whether to embed the entire font or a subset is determined by number of glyphs in the font that are used, and the value of *MaxSubsetPct*. Note: Embedded instances of multiple master fonts are always subsetted, regardless of the setting of SubsetFonts. The *AlwaysEmbed* and *NeverEmbed* fonts lists are restored to their default values between each job.

7.2.57 FontPolicy

This resource defines the policies that devices must follow when font errors occur while PDL files are being processed. When fonts are referenced by PDL files but are not provided, devices may provide one of the following two fallback behaviors:

1. The device may provide a standard default font which is substituted whenever a font cannot be found.

2. The device may provide an emulation of the missing font.

If neither fallback behavior is requested, i.e., both *UseDefaultFont* and *UseFontEmulation* are false, then the job will fail if a referenced font is not provided. **FontPolicy** allows jobs to specify whether or not either of these fallback behaviors should be employed when missing fonts occur.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: IDPrinting *Interpreting*, *SoftProofing*
Output of processes: -

Resource Structure

Name	Data Type	Description
PreferredFont	NMTOKEN	The name of a font to be used as the default font for this job. It is not an error if the device cannot use the specified font as its default font.
UseDefaultFont	boolean	If *true*, the device must resort to a default font if a font cannot be found. This is the normal behavior of the PostScript interpreter, which defaults to courier when a font cannot be found.
UseFontEmulation	boolean	If *true*, the device must emulate a required font if a font cannot be found.

7.2.58 FormatConversionParams

New in JDF 1.1

This resource defines the parameters needed for generic *FormatConversion* of digital files.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag*
Input of processes: *FormatConversion*
Output of processes: -

Resource Structure

Name	Data Type	Description
FileSpec ?	refelement	The format of the original file is specified in a **FileSpec** with *ResourceUsage* = *InputFormat*. No URL should be specified, because the list of files is given by the input RunList of the *FormatConversion* process.
FileSpec ?	refelement	The format of the original file is specified in a **FileSpec** with *ResourceUsage* = *OutputFormat*. No URL should be specified, because the list of files is given by the output RunList of the *FormatConversion* process.

7.2.59 GatheringParams

This resource contains the attributes of the *Gathering* process.
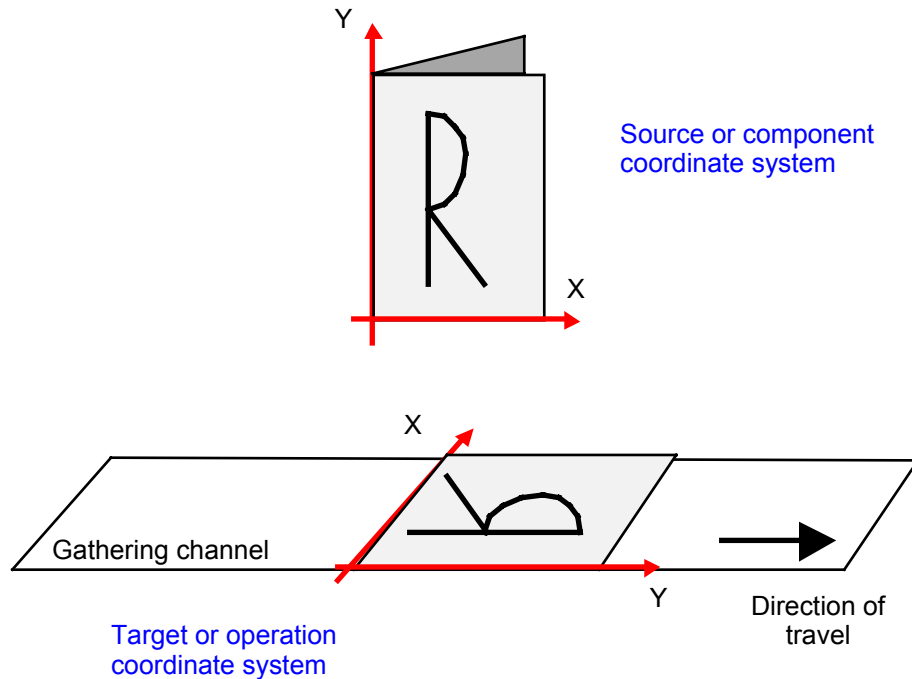
Figure 7.13 Coordinate system used for gathering

Resource Properties

Resource class: Parameter
Resource referenced by: -
Input of processes: *Gathering*
Output of processes: -

Resource Structure

Name	Data Type	Description
Disjointing ?	element	Description of the separation properties between individual components on a gathered pile. Default = no physical separation.

7.2.60 GlueApplication

New in JDF 1.1

This resource specifies glue application in hard and soft cover book production.

Resource Properties

Resource class: Parameter
Resource referenced by: **CoverApplicationParams, SpineTapingParams**
Input of processes: *-*
Output of processes: -

Resource Structure

Name	Data Type	Description
GluingTechnique	enumeration	Type or technique of gluing application. Possible values are: *SpineGluing* *SideGluingFront* *SideGluingBack*
GlueLine	refelement	Structure of the glue line.
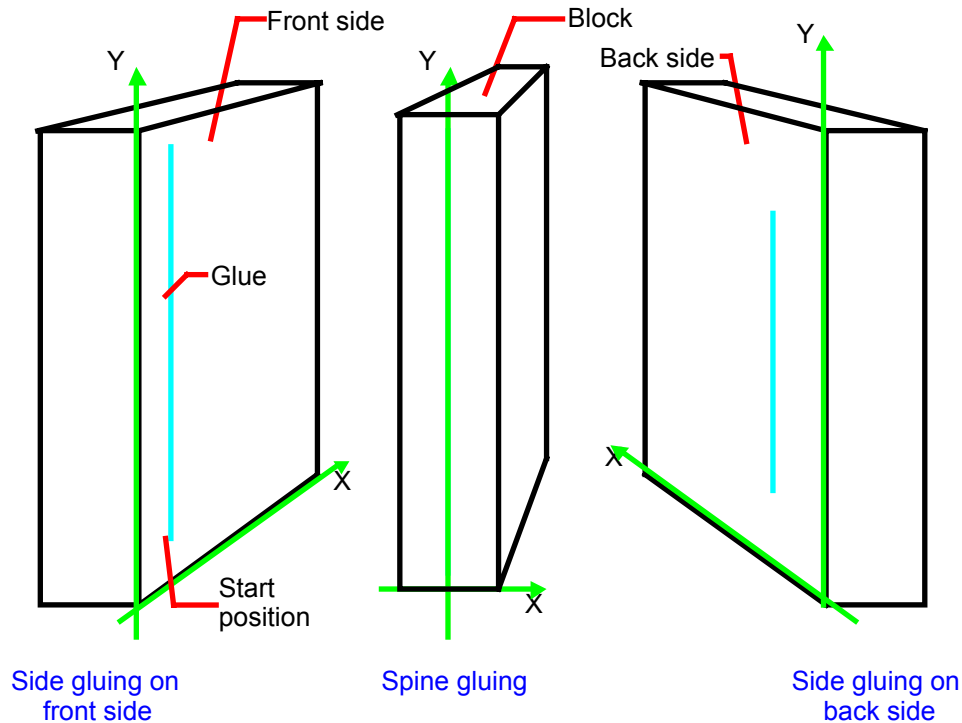
Figure 7.14 Parameters and coordinate system for glue application

7.2.61 GluingParams

New in JDF 1.1

GluingParams define the parameters applying a generic line of glue to a component.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Gluing*
Output of processes: -

Resource Structure

Name	Data Type	Description
Glue *	element	Definition of one or more Glue line applications.

Properties of the Glue Element

The Glue element describes how to apply a line of glue.

Name	Data Type	Description
WorkingDirection	enumeration	Direction from which the tool is working. Possible values are: *Top* – from above *Bottom* – from below
GlueApplication	element	Description of the glue application.

7.2.62 GlueLine

This resource provides the information to determine where and how to apply glue.

Resource Properties

Resource class: Parameter
Resource referenced by: CaseMakingParams; EndSheetGluingParams, FoldingParams,
                             CoverApplicationParams, InsertingParams, SpineTapingParams,
                             ThreadSewingParams
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
AreaGlue ? New in JDF 1.1	boolean	Specifies that this GlueLine should cover the complete width of the **Component** it is applied to. Default=*false*.
GlueBrand ?	string	Glue brand. Default = empty string, i.e., system specified
GlueLineWidth ?	double	Width of the glue line. Default = equipment-specific setting Note: In extreme cases the glue line could cover the input component over the hole width.
GluingPattern ?	XYPair	Glue line pattern defined by the length of a glue line segment (X element) and glue line gap (Y element). A solid line is expressed by the pattern (1 0), the default.
GlueType ?	enumeration	Glue type. Possible values are: *ColdGlue* – Any type of glue that needs no heat treatment. *Hotmelt* – Hotmelt EVA (Ethyl-Vinyl-Acetat-Copolymere) *PUR* – Polyurethane Default = equipment specific setting
MeltingTemperature ?	integer	Required temperature for melting the glue (in degrees centigrade). Used only when **GlueType** = *Hotmelt* or **GlueType** = *PUR*. Default = equipment-specific setting
StartPosition	XYPair	Start position of glue line. The start position is given in the coordinate system of the mother sheet. Default = (0 0)
WorkingPath	XYPair	Relative working path of the gluing tool. Default = equipment-specific setting.

7.2.63 HeadBandApplicationParams

New in JDF 1.1

This resource specifies how to apply headbands in hard cover book production.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *HeadBandApplication*
Output of processes: -

Resource Structure

Name	Data Type	Description
BottomBrand ?	string	Bottom head band brand. If not specified, defaults to *TopBrand*.
BottomColor ?	NamedColor	Color of the bottom head band. If not specified, defaults to *Top-Color*.
BottomLength ?	double	Length of the carrier material of the bottom head band along binding edge. If not specified, both head bands are on one carrier.
TopBrand ?	string	Top head band brand. Default =system specified.
TopColor ?	NamedColor	Color of the top head band. Default =system specified.
TopLength ?	double	Length of carrier material of the top head band along binding edge. If not specified, both head bands are on one carrier which has the length of the book block.
StripMaterial ?	enumeration	Strip material. Possible values are: *Calico* *Cardboard* *CrepePaper* *Gauze* *Paper* *PaperlinedMules* *Tape* Default =system specified.
Width ?	number	Width of the head bands and carrier.
GlueLine *	refelement	The carrier may be applied to the bookblock with glue. The coordinate system for the **GlueLine** is defined in the Section 7.2.50

7.2.64 Hole

The Hole element describes an individual hole.

Resource Properties

Resource class: **Parameter**
Resource referenced by: **HoleLine**, **HoleMakingIntent**, **HoleMakingParams**
Example Partition: -
Input of processes: *-*
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the hole relative to the Component coordinate system. For more information, see Section 6.5.45.2.
Extent	XYPair	Size (Bounding Box) of the hole in points. If *Shape* is *Round*, only the first entry of *Extent* is evaluated and defines the hole diameter.

Name	Data Type	Description
Shape Modified in JDF 1.1	enumeration	Shape of the hole. Possible values are: *Elliptic* *Round* *Rectangular*

7.2.65 HoleLine

New in JDF 1.1

Line Hole Punching generates a series of holes with identical distance (pitch) running parallel to the edge of a web, which is mainly used to transport paper through continuous-feed printers and finishing devices (form processing). The final product typically is a web with two lines of holes, one at each edge of the web. The parameters for one line of Holes are specified in the HoleLine element. The distance between holes within each line of holes is identical (constant pitch).

Resource Properties

Resource class: Parameter
Resource referenced by: HoleMakingIntent, HoleMakingParams
Example Partition: -
Input of processes: -
Output of processes: -
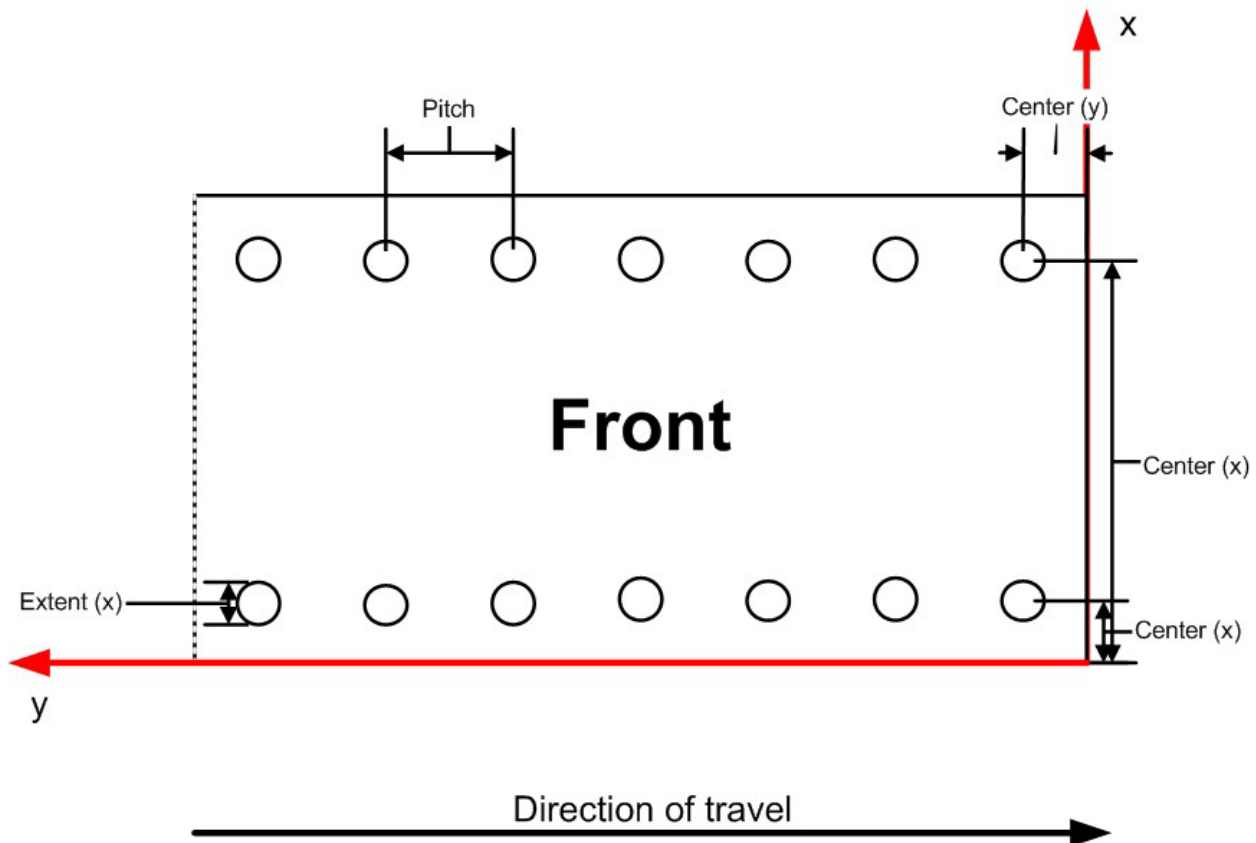
However, sometimes Line Hole Punching is performed for multiple webs before dividing the web after the Hole-Making process as illustrated below:
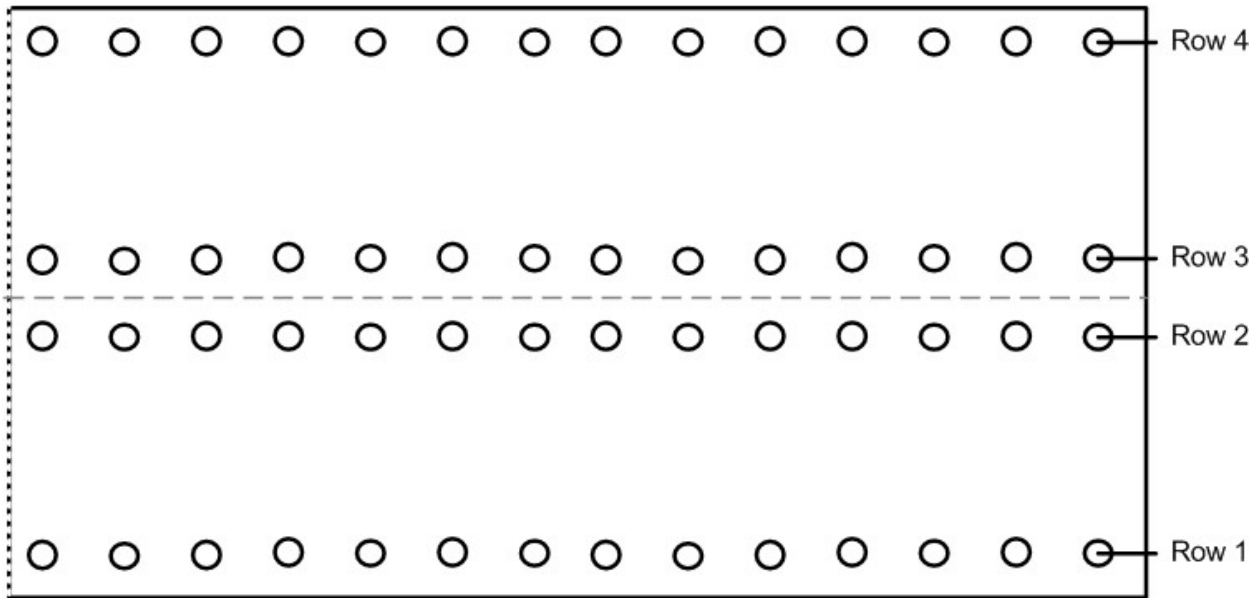
The parameters of the HoleLine element are:

Resource Structure

Name	Data Type	Description
Pitch	number	Center-hole to center-hole distance within a line of holes.
Hole	element	Size and position of the first hole in the HoleLine.

7.2.66 HoleMakingParams

This resource specifies where to make a hole of what shape in components. This information is used by the **Hole-Making** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *SheetName*
Input of processes: *HoleMaking,*
Output of processes: -

Resource Structure

Name	Data Type	Description
Center ? Modified in JDF 1.1	XYPair	Position of the center of the hole pattern relative to the Component coordinate system if HoleType is not equal *Explicit*. If not specified, it defaults to the value implied by *HoleType*.
CenterReference ? New in JDF 1.1	enumeration	Defines the reference coordinate system for *Center*. One of: *TrailingEdge* – Physical coordinate system of the component. The default. *RegistrationMark* – The center is relative to a registration mark.
Extent ?	XYPair	Size (Bounding Box) of the hole in pt if *HoleType* is not equal *Explicit*. If *Shape* is *Round*, only the first entry of *Extent* is evaluated and defines the hole diameter. If not specified, it defaults to the value implied by *HoleType*.
HoleReferenceEdge ?	enumeration	The edge of the media relative to where the holes should be punched.

Name	Data Type	Description
New in JDF 1.1		Use with *HoleType*. Possible values are: *Left* *Right* *Top* *Bottom* *Pattern* – Specifies that the reference edge implied by the value of *HoleType* in Appendix L JDF/CIP4 Hole Pattern Catalog is used. The default if *HoleType* is *Explicit*, otherwise *Left*.
HoleType New in JDF 1.1	enumerations	Predefined hole pattern. Multiple hole patterns are allowed, e.g., 3-hole ring binding and 4-hole ring binding holes on one piece of media. For details of the hole types, refer to Appendix L JDF/CIP4 Hole Pattern Catalog. Allowed values are: *R2-generic* *R6-generic* *R2m-DIN* *R6m-4h2s* *R2m-ISO* *R6m-DIN-A5* *R2i-US-a* *R7-generic* *R2i-US-b* *R7i-US-a* *R3-generic* *R7i-US-b* *R3i-US* *R7i-US-c* *R4-generic* *R11m-7h4s* *R4m-DIN-A4* *P12m-rect-0t* *R4m-DIN-A5* *P16_9i-rect-0t* *R4m-swedish* *W2_1i-round-0t* *R4i-US* *W2_1i-square-0t* *R5-generic* *W3_1i-square-0t* *R5i-US-a* *C9.5m-round-0t* *R5i-US-b* *Explicit* – Holes are defined in an array of Hole or HoleLine elements. *R5i-US-c* The following values are deprecated from JDF 1.0 *2HoleEuro* – Replace by either R2m-DIN or R2m-ISO. *3HoleUS* – Replace by R3I-US *4HoleEuro* – Replace by R4m-DIN-A4 or R4m-DIN-A5
Shape ? Modified in JDF 1.1	enumeration	Shape of the holes if *HoleType* is not equal *Explicit*.. Possible values are: *Elliptic* *Round* *Rectangular* If not specified, it defaults to the value implied by *HoleType*.
Hole *	element	Description of individual Hole elements.
HoleLine * New in JDF 1.1	element	Description of HoleLine elements.
RegisterMark ? New in JDF 1.1	refelement	Reference to the registration mark that defines the coordinate system.

Structure of HoleMakingParams Elements

7.2.67 IdentificationField

This resource contains information about a mark on a document, such as a bar code, used for OCR-based verification purposes or document separation.

Resource Properties

Resource class: Parameter
Resource referenced by: Disjointing, Sheet, Surface, any physical resource
Example Partition: -
Input of processes: *Verification*, *Inserting*, *Collecting*, *Gathering*,
Output of processes: -

Resource Structure

Name	Data Type	Description
BoundingBox ?	rectangle	Box that provides the boundaries in the coordinate system of the mark that indicates where the component is to be placed. If no *BoundingBox* is defined, the complete visible surface must be scanned for an appropriate bar code.
Encoding	enumeration	Encoding of the information. Possible values are: *ASCII* – Plain-text font. *BarCode1D* – One-dimensional bar code. *BarCode2D* – Two-dimensional bar code.
EncodingDetails	NMTOKEN	Details about the encoding type. An example is the bar code scheme. Possible values are: *Code39* *Interleave25* *Plessey* *EAN*
Format ?	string	Regular expression[3] that defines the expected format of the expression, such as the number of digits, alphanumeric, or numeric. Note that this field may also be used to define constant fields, such as the end of document markers or packaging labels. Default is that any expression is valid (*Format* = "*").
Orientation ?	matrix	Orientation of the contents within the field. The coordinate system is defined in the system of the sheet or component where the **IdentificationField** resides. Default = identity matrix.

---

[3] This is a regular expression as in UNIX grep.

Name	Data Type	Description
Position ?	enumeration	Position with respect to the instance document or physical resource to which the **IdentificationField** resource refers. Possible values are: *Header* – Sheet before the document. *Trailer* – Sheet after the document. *Page* – A page of the document. *Top* – The top of the resource. *Bottom* – The bottom of the resource. *Left* – The left side of the resource. *Right* – The right side of the resource. *Front* – The front side of the resource. *Back* – The back side of the resource. *Any*: the default.
Page ?	integer	If *Position* = *Page*, this refers to the page where the **Identification-Field** can be found. Negative values denote an offset relative to the last page in a stack of pages.
Purpose ?	enumeration	Purpose defines the usage of the field. Possible values are: *Label* – The default, used to mark a product or component. *Separation* – used to separate documents. *Verification* – used for verification of documents.
Value ? New in JDF 1.1	string	Fixed value of the **IdentificationField**, e.g., on a label.

7.2.68 IDPrintingParams

Deprecated in JDF 1.1

This resource contains the parameters needed to control the *IDPrinting* process.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, DocRunIndex, DocSheetIndex, PartVersion, Run, RunIndex, RunTag, SheetIndex, SheetName, Side*

Input of processes: *IDPrinting*
Output of processes: -

Resource Structure

Name	Data Type	Description
AttributesNaturalLang ?	language	Language selected for communicating attributes. Default = US English
IDPAttributeFidelity ?	boolean	Indicates whether or not the device must reject the job if there are attribute values or elements that it does not support. Default = false
IPPJobPriority ?	integer	The scheduling priority for the job where 100 is the highest and 1 is the lowest. Amongst the jobs that can be printed, all higher priority jobs must be printed before any lower priority ones. Default = 50

Name	Data Type	Description
IPPVersion ?	XYPair	A pair of numbers indicating the version of the IPP protocol to use when communicating to IPP devices. The X value is the major version number. Default = system specified
OutputBin ?	NMTOKEN	Specifies the bin to which the finished document should be output. Possible values are: *Top* – The bin that, when facing the device, can best be identified as "top". *Middle* – The bin that, when facing the device, can best be identified as "middle". *Bottom* – The bin that, when facing the device, can best be identified as "bottom". *Side* – The bin that, when facing the device, can best be identified as "side". *Left* – The bin that, when facing the device, can best be identified as "left". *Right* – The bin that, when facing the device, can best be identified as "right". *Center* – The bin that, when facing the device, can best be identified as "center". *Rear* – The bin that, when facing the device, can best be identified as "rear". *FaceUp* – The bin that can best be identified as "face up" with respect to the device. *FaceDown* –The bin that can best be identified as "face down" with respect to the device. *FitMedia* – Requests the device to select a bin based on the size of the media. *LargeCapacity* – The bin that can best be identified as the "large capacity" bin (in terms of the number of sheets) with respect to the device. *Mailbox-N* – The job will be output to the bin that is best identified as "Mailbox-1", "Mailbox-2"…etc. *Stacker-N* –The job will be output to the bin that is best identified as "Stacker-1', 'Stacker-2" …etc. *Tray-N* – The job will be output to the tray that is best identified as "Tray-1", "Tray-2" … etc. *SystemSpecified* - The job will be output to the tray that is best identified as "*SystemSpecified*" Default = *SystemSpecified*
PageDelivery ?	enumeration	Indicates how pages are to be delivered to the output bin or finisher. Possible values are: *SameOrderFaceUp* – Order as defined by the RunList, with the "front" sides of the media up. *SameOrderFaceDown* – Order as defined by the RunList, with the "front" sides of the media up. *ReverseOrderFaceUp* – Order reversed, as defined by the RunList, with the "front" sides of the media up. *ReverseOrderFaceDown* – Order reversed, as defined by the RunList, with the "front" sides of the media down. *SystemSpecified* – Order and face-up/face-down as defined by the system.

Name	Data Type	Description
		Default = *SystemSpecified*
PrintQuality ?	enumera-tion	Indicates how pages are to be delivered to the output bin or finisher. Possible values are:
		High – Highest quality available on the printer.
		Normal – The default quality provided by the printer.
		Draft – Lowest quality available on the printer
		SystemSpecified – System specified default print quality.
		Default = *SystemSpecified*
SheetCollate ?	boolean	Determines whether the sequencing of the pages in the output of the job.
		If *true*, pages for each copy of the document are sequenced together, followed by the pages for the next copy.
		If *false*, all copies of the first page are sequenced, followed by the second and subsequent pages.
		SheetCollate describes the order of the final pages, but does not prescribe the order in which they are produced.
		Default = system specified.
Cover *	element	0, 1 or 2 Cover elements.
		Default = no cover
IDPFinishing ?	refelement	This element provides the details of how media for each instance document should be finished.
		Default = system specified
IDPLayout ?	refelement	This element provides the details of how page contents will be imaged onto media.
		Default = system specified
JobSheet *	element	A set of sheets which must be produced with the job.
		Default = no job sheets produced
MediaIntent ?	refelement	A MediaIntent element. This element is ignored if a **MediaSource** resource is present and can be honored for the *IDPrinting* process. If **MediaSource** is absent or cannot be honored, this element describes the intended media for the job to allow the device to select from among the available media.
MediaSource?	refelement	Describes the source and physical orientation of the media to be used.

Structure of the Cover Subelement
Deprecated in JDF 1.1
This element describes the cover requested for the job. Covers may be applied to the whole job, or to each instance document in the job. Note that front and back covers may be specified.

Name	Data Type	Description
BackSide ?	boolean	The next page from the RunList is imaged onto the back of this cover. This would be the inside of a *Front* cover and outside of a *Back* cover. Default = *false*
CoverType ?	enumeration	Specifies whether this Cover element specifies the front or back cover.
		Front – The front cover.
		Back – The back cover.
		Default = *Front*

Name	Data Type	Description
FrontSide ?	boolean	The next page from the RunList is imaged onto the front of this cover. This would be the outside of a *Front* cover and inside of a *Back* cover. Default = *false*
IDPFinishing?	refelement	An IDPFinishing element that describes the finishing options for the cover.
IDPLayout ?	element	This element provides the details of how page contents will be imaged onto the cover.
MediaIntent ?	refelement	A MediaIntent element. This element describes the media to be used for the job. This element is ignored if a **MediaSource** resource is present and can be honored for the *IDPrinting* process. If **MediaSource** is absent or cannot be honored, this element describes the intended media for the job to allow the device to select from among the available media.
MediaSource?	refelement	Describes the source and physical orientation of the media to be used.

Properties of the IDPFinishing Subelement

Deprecated in JDF 1.1

IDPFinishing elements describe finishing operations that should be applied to sets of pages that are output by the *IDPrinting* process. The finishings are applied to the entire job when there are no instance documents. Otherwise, each instance document is finished separately. Operation-specific subelements may also be present when a device provides controls for a finishing operation. Additional subelements are expected to be defined over time. Also, more detail will be added to the currently defined elements.

Name	Data Type	Description
Finishings ?	IntegerList	A set of finishing operations to apply to the job. The operations are encoded as an enumeration:
		Possible values are:
		3 – (none) Perform no finishing
		4 – (staple) Bind the document(s) with one or more staples. The exact number and placement of the staples is site-defined.
		5 – (punch) This value indicates that holes are required in the finished document. The exact number and placement of the holes is site-defined The punch specification may be satisfied (in a site- and implementation-specific manner) either by drilling/punching, or by substituting predrilled media.
		6 – (cover) This value is specified when it is desired to select a non-printed (or preprinted) cover for the document. This does not supplant the specification of a printed cover (on cover stock medium) by the document itself.
		7 – (bind) This value indicates that a binding is to be applied to the document; the type and placement of the binding is site-defined.
		8 – (saddle-stitch) Bind the document(s) with one or more staples (wire stitches) along the middle fold. The exact number and placement of the staples and the middle fold is implementation and/or site-defined.
		9 – (edge-stitch) Bind the document(s) with one or more staples (wire stitches) along one edge. The exact number and placement of the staples is implementation and/or site-defined.
		10 – (fold) Fold the document(s) with one or more folds. The exact number and orientations of the folds is implementation and/or site-defined.
		11 – (trim) Trim the document(s) on one or more edges. The exact number of edges and the amount to be trimmed is implementation and/or site-defined.
		12 – (bale) Bale the document(s). The type of baling is implementation and/or site-defined.
		13 – (booklet-maker) Deliver the document(s) to the signature booklet maker. This value is a short cut for specifying a job that is to be folded, trimmed and then saddle-stitched.
		14 – (jog-offset) Shift each copy of an output document from the previous copy by a small amount which is device dependent. This value has no effect on the "job-sheet." This value should not have an effect if each copy of the job consists of one sheet.
		50 – (bind-left) Bind the document(s) along the left edge. The type of the binding is site-defined.
		51 – (bind-top) Bind the document(s) along the top edge. The type of the binding is site-defined.
		52 – (bind-right) Bind the document(s) along the right edge. The type of the binding is site-defined.
		53 – (bind-bottom) Bind the document(s) along the bottom edge. The type of the binding is site-defined.
IDPFolding ?	refelement	Provides details of how to fold the set of pages (or document). When this element is present, *Finishings* is ignored.
IDPHoleMaking ?	refelement	Provides details of how to punch holes in the set of pages (or document). When this element is present, *Finishings* is ignored.
IDPStitching ?	refelement	Provides details of how to stitch the set of pages (or document). When this element is present, *Finishings* is ignored.

Name	Data Type	Description
IDPTrimming ?	refelement	Provides details of how to trim the set of pages (or document). When this element is present, *Finishings* is ignored.

Structure of IDPFolding Subelement

Deprecated in JDF 1.1

This element describes the folding requested for a set of pages in the document.

Name	Data Type	Description
FoldingParams ?	Refelement	Describes the details of how to fold the media.

Structure of IDPHoleMaking Subelement

Deprecated in JDF 1.1

This element describes the hole making requested for a set of pages in the document.

Name	Data Type	Description
HoleMakingParams ?	refelement	Describes the details of the holes to be punched into the Media.

Structure of the IDPLayout Subelement

Deprecated in JDF 1.1

Name	Data Type	Description
Border ?	number	A real number that indicates the width of a border, in points, which will be drawn around the page images on the media. Default = 0, i.e., no border will be drawn.
FinishedPageOrientation ?	enumeration	Indicates the desired orientation of the finished page. This value is used with PresentationDirection to determine how pages will be imaged onto the media. Possible values are: *Portrait* – The short edges of the media are the top and bottom. *Landscape* – The long edges of the media are the top and bottom. Default = *Portrait*.
ForceFrontSide ?	Number-RangeList	A set of numbers which identify a set of pages in the **RunList** that should always be imaged on the front side of a piece of media.
ImageShift ?	element	Element which describes how page images should be placed onto the media. When *NumberUp* is present and is not "1,1", *NumberUp* is applied before the ImageShift, and all contents for each surface are shifted the same amount.
NumberUp ?	XYPair	The number of pages to impose onto a single side of media. The way in which the pages are to be imaged onto the media is determined by the values of *FinishedPageOrientation* and *PresentationDirection*. *FinishedPageOrientation* indicates how the page will be oriented, and *PresentationDirection* indicates how page images will be distributed, given that orientation.
PresentationDirection ?	enumeration	Indicates the order in which the requested NumberUp pages will be imaged onto the media. The value of FinishedPageOrientation is used to define "top", "left", "right" and "bottom" for the media. Possible values are: *ToBottomToRight* – Pages are imaged in successive columns, from left to right, starting at the top of each column. *ToBottomToLeft* – Pages are imaged in successive columns, from right to left, starting at the top of each column. *ToTopToRight* – Pages are imaged in successive columns, from left to right, starting at the bottom of each column. *ToTopToLeft* – Pages are imaged in successive columns, from right

Name	Data Type	Description
		to left, starting at the bottom of each column.
		ToRightToBottom – Pages are imaged in successive rows, from top to bottom, starting at the left of each row.
		ToRightToTop – Pages are imaged in successive rows, from bottom to top, starting at the left of each row.
		ToLeftToBottom – Pages are imaged in successive rows, from top to bottom, starting at the right of each row.
		ToLeftToTop – Pages are imaged in successive rows, from bottom to top, starting at the right of each row.
		Default = *SystemSpecified*
Rotate ?	number	A number of degrees which the page contents are to be rotated prior to being imaged onto page contents. A positive value is taken to mean an counter-clockwise rotation. The page contents will be scaled to fit the printable area of the media after the rotation.
		Note: Text will be reflowed in cases where the PDL for the page allows reflow by the device.
		Default = 0
Sides ?	enumeration	Indicates how pages should be imposed onto sides of the medium. Possible values are:
		OneSided – Page contents will only be imaged on one side of the media. The default.
		TwoSidedLongEdge – Impose pages upon the front and back sides of media sheets so that the orientation of the pages on each side is appropriate for binding along the long edge. Equivalent to "*work-and-turn*".
		TwoSidedShortEdge – Impose pages upon the front and back sides of media sheets so that the orientation of the pages on each side is appropriate for binding along the short edge. Equivalent to "*work-and-tumble*".

Structure of IDPStitching Subelement

Deprecated in JDF 1.1

This element describes the stitching requested for a set of pages in the document.

Name	Data Type	Description
StitchingPosition ?	enumeration	Specifies the location for stitching. All locations are interpreted as if the document were a portrait document. Ignored if StitchingParams is present. Possible values are: *None* – The document is not to be stitched. *TopLeft* – Bind the document with one or more staples in the top left corner. *BottomLeft* – Bind the document with one or more staples in the Bottom left corner. *TopRight* – Bind the document with one or more staples in the top right corner. *BottomRight* – Bind the document with one or more staples in the bottom right corner. *LeftEdge* – Bind the document with one or more staples across the left edge. *TopEdge* – Bind the document with one or more staples across the top edge. *RightEdge* – Bind the document with one or more staples across the right edge. *BottomEdge* – Bind the document with one or more staples across the bottom edge. *DualLeftEdge* – Bind the document with two staples across the left edge. *DualTopEdge* – Bind the document with two staples across the top edge. *DualRightEdge* – Bind the document with two staples across the right edge. *DualBottomEdge* – Bind the document with two staples across the bottom edge.
StitchingReferenceEdge ?	enumeration	The edge of the output media relative to which the stapling or stitching must be applied. If StitchingParams is present, *StitchingReferenceEdge* defines the *BindingEdge*. Possible values are: *Bottom* – The bottom edge coincides with the x-axis of the coordinate system. *Top* – The top edge is opposite and parallel to the bottom edge. *Left* – The left edge coincides with the y-axis of the coordinate system. *Right* – The right edge is opposite and parallel to the left edge.
StitchingParams ?	refelement	A StitchingParams element which provides detailed control of the stitching. *StitchingReferenceEdge* must be present if StitchingParams is provided.

Structure of IDPTrimming Subelement

Deprecated in JDF 1.1

This element describes the trimming requested for a set of pages in the document.

Name	Data Type	Description
TrimmingParams ?	Refelement	Describes the details of how to trim the media.

Structure of the ImageShift Subelement

Deprecated in JDF 1.1

ImageShift elements describe how page contents will be imaged onto media. All attributes refer to positioning along the "X" or "Y" axis. The "X" dimension is the first number of the Media *Dimension* attribute; "Y" is the second number.

Name	Data Type	Description
PositionX ?	enumeration	Indicates how page images should be positioned horizontally on the surface. Shifts are applied after positioning. Values are: *Center* – Center the page images horizontally on the surface without regard to limitations of the printable area. *Left* – Position the left edge of the page images so they is coincident with the left edge of the printable area of the surface. *None* – Place the page images wherever the print data specifies (the default). *Right* – Position the right edge of the page images so they is coincident with the right edge of the printable area of the surface.
PositionY ?	enumeration	Indicates how page images should be positioned vertically on the surface. Shifts are applied after positioning. Values are: *Bottom* – Position the bottom edge of the page images so they is coincident with the bottom edge of the printable area of the surface. *Center* – Center the page images horizontally on the surface without regard to limitations of the printable area. *None* – Place the page images wherever the print data specifies (the default). *Top* – Position the top edge of the page images so they is coincident with the top edge of the printable area of the surface.
ShiftX ?	integer	The image is to be shifted along the x axis on both sides of the media.
ShiftY ?	integer	The image is to be shifted along the y axis on both sides of the media.
ShiftXSide1 ?	integer	The image is to be shifted along the x axis on the front side of the media.
ShiftXSide2 ?	integer	The image is to be shifted along the x axis on the back side of the media.
ShiftYSide1 ?	integer	The image is to be shifted along the y axis on the front side of the media.
ShiftYSide2 ?	integer	The image is to be shifted along the y axis on the back side of the media.

Structure of the JobSheet Subelement

Deprecated in JDF 1.1

This element describes a job sheet which may be produced along with the job. Job sheets include separators, sheets, and error sheets. The information provided on the sheet depends on the type of sheet. In addition, any sheet type may include an optional message as a comment subelement for the sheet element. Such a message comment must have a *Name* attribute with the value 'SheetMessage'.

Name	Data Type	Description
SheetFormat ?	NMTOKEN	Identifies the format of the JobSheet. The default is 'Standard', but site-specific values may be defined.
SheetOccurrence	enumeration	Indicates when the sheet is to be produced and inserted into the set of output pages. Possible values are: *Always* – Valid for *ErrorSheet* or *AccountingSheet*. The sheet is always produced at the end of the job. *End* – Valid for *JobSheet* or *SeparatorSheet.* The sheet is produced at the end of the job (for *JobSheet*) or at the end of each copy of each instance document (for *SeparatorSheet*). *OnError* – Valid for *ErrorSheet*. The sheet is produced at the end of the job when an error or warning occurs. *Slip* – Valid for *SeparatorSheet*. The sheet is produced between each copy of each instance document. *Start* – Valid for *JobSheet* or *SeparatorSheet*. The sheet is produced at the start of the job (for *JobSheet*) or at the start of each copy of each instance document (for *SeparatorSheet*). *Both* – Valid for *JobSheet* or *SeparatorSheet*. The sheet is produced at the beginning and end of the job (for *JobSheets*) or at the beginning and end of each copy of each instance document (for *SeparatorSheets*). *None* – Valid for any **SheetType**.
SheetType	enumeration	Identifies the type of sheet. Possible values are: *AccountingSheet* – A sheet that reports accounting information for the job. *ErrorSheet* – A sheet that reports errors for the job. *JobSheet* – A sheet that delimits the job. *SeparatorSheet* – A sheet that delimits one copy (set) of the job.
IDPFinishing ?	refelement	An IDPFinishing element that describes the finishing options for the job sheet.
IDPLayout ?	element	This element provides the details of how page contents will be imaged onto the job sheet.
MediaIntent ?	refelement	A MediaIntent element. This element describes the media to be used for the job sheets. This element is ignored if a **MediaSource** resource is present and can be honored. If **MediaSource** is absent or cannot be honored, this element describes the intended media for the job sheets to allow the device to select from among the available media.
MediaSource?	refelement	Describes the source and physical orientation of the media to be used.

Overriding IDPrintingParams using Partitioning

IDPrintingParams may be overridden using partitioning mechanisms as described in 3.9.2 Description of Partitionable Resources. Overrides may apply to a set of instance documents, set of copies of instance documents, or to a set of pages, output surfaces, sheets of media in a personalized printing job, or header or trailer insert sheets added by a RunList. Note: If more than one override refers to the same content, the lowest level override takes precedence. The following list defines partitioning precedence, from lowest to highest, i.e., the lower entries in the list take precedence:

Job level partitioning (lowest priority):

PartVersion, Run, SheetName, Side, RunTag

Page level partitioning:

RunIndex

SheetIndex

Instance document level partitioning (highest priority):

DocCopies

DocIndex

DocSheetIndex

DocRunIndex

Note: It is strongly discouraged to mix page level partitions and instance document level partitions. Cover elements in **IDPrintingParams** are counted when calculating *DocSheetIndex* or *DocRunIndex.*

Example of a partitioned IDPrinting Node

The following example shows how partitioning can be used to describe a fairly complex example. Three color models (**ColorantControl** partitions) are applied to a set of sheets using the *DocSheetIndex* key;

    1.) DeviceN: *DocSheetIndex* = "*0"* defines the cover;

    2.) DeviceCMYK: *DocSheetIndex* = "*1"* defines the first sheet (non cover);

    3.) DeviceGray: *DocSheetIndex* = "*2~-1"* defines all other sheets;

The cover is selected from a different input tray using the *Location* key. The same key is used to describe the **Media** in each tray.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM20010402140111" Type="IDPrinting" JobID="HDM20010402140111" Status="Waiting" Ver-
sion="1.0">
  <ResourcePool>
    <Media ID="Link0003" Class="Consumable" Locked="false" Status="Available" Dimension="700 900"
MediaType="Paper" PartIDKeys="Location">
      <Media Weight="90" Location="Tray 1"/>
      <Media Weight="120" Location="Tray 2"/>
    </Media>
    <RunList ID="Link0004" Class="Parameter" Locked="false" Status="Available" PartIDKeys="Run">
      <RunList Run="Run0005" Pages="0">
        <LayoutElement>
          <FileSpec URL="Cover.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Run="Run0006" Pages="0~7">
        <LayoutElement>
          <FileSpec URL="File2.pdf"/>
        </LayoutElement>
      </RunList>
    </RunList>
    <IDPrintingParams ID="Link0008" Class="Parameter" rRefs="Link0003" Locked="false"
Status="Available">
      <IDPLayout NumberUp="2 2"/>
      <MediaSource MediaLocation="Tray 1">
        <MediaRef rRef="Link0003"/>
      </MediaSource>
      <Cover CoverType="Front" FrontSide="true">
        <IDPLayout NumberUp="1 1"/>
        <MediaSource MediaLocation="Tray 2">
          <MediaRef rRef="Link0003"/>
        </MediaSource>
      </Cover>
    </IDPrintingParams>
    <ColorantControl ID="Link0009" Class="Parameter" Locked="false" Status="Available" PartID-
Keys="DocSheetIndex">
      <ColorantControl DocSheetIndex="0" ProcessColorModel="DeviceN"/>
      <ColorantControl DocSheetIndex="1" ProcessColorModel="DeviceCMYK"/>
      <ColorantControl DocSheetIndex="2~-1" ProcessColorModel="DeviceGray"/>
    </ColorantControl>
```

```
    </ResourcePool>
    <ResourceLinkPool>
      <MediaLink rRef="Link0003" Usage="Input"/>
      <RunListLink rRef="Link0004" Usage="Input"/>
      <IDPrintingParamsLink rRef="Link0008" Usage="Input"/>
      <ColorantControlLink rRef="Link0009" Usage="Input"/>
    </ResourceLinkPool>
</JDF>
```

7.2.69 ImageCompressionParams

This resource provides information describing how images are to be compressed in PDF files.

Resource Properties

Resource class: Parameter
Resource referenced by: Sheet
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *ImageReplacement, Preflight*
Output of processes: -

Resource Structure

Name	Data Type	Description
ImageCompression *	element	Specifies how images are to be compressed.

Structure of ImageCompression Subelement

Name	Data Type	Description
AntiAliasImages ?	boolean	If *true*, anti-aliasing is permitted on images. If *false*, anti-aliasing is not permitted. Anti-aliasing increases the number of bits per component in down-sampled images to preserve some of the information that is otherwise lost by downsampling. Anti-aliasing is only performed if the image is actually downsampled and if *ImageDepth* has a value greater than the number of bits per color component in the input image. Default = *false*
AutoFilterImages ? Modified in JDF1.1A	boolean	Used only if *EncodeImages* is *true*. This attribute is not used if *ImageType* = *monochrome*. If *true*, the *DCTEncode* filter is applied to photos and the *FlateEncode* filter is applied to screen shots. If *false*, the *ImageFilter* compression method is applied to all images. This parameter is ignored for monochrome images. Default = *true*
ConvertImagesToIndexed ?	boolean	If *true*, the application converts images that use fewer than 257 colors to an indexed colorspace for compactness. This attribute is used only when *ImageType* = *color*.
DCTQuality ?	number	A value between 0 and 1 that indicates "how much" the process should compress images when using a *DCTEncode* filter. 0.0 means "do as loss-less compression as possible." 1.0 means "do the maximum compression possible." Default = 0
DownsampleImages ? Modified in JDF1.1A	boolean	If *true*, sampled color images are downsampled using the resolution specified by *ImageResolution*. If *false*, downsampling is not carried out and the image resolution in the PDF file is the same as that in the source file. Defaults = *false*

Name	Data Type	Description
EncodeColorImages ? Deprecated in JDF 1.1	boolean	If *true*, color images are encoded using the compression filter specified by the value of the *ImageFilter* key. If *false*, no compression filters are applied to color sampled images. Default = *false*
EncodeImages ? New in JDF 1.1 Modified in JDF1.1A	boolean	If *true*, images are encoded using the compression filter specified by the value of the *ImageFilter* key. If *false*, no compression filters are applied to sampled images. Default = *false*
ImageDepth ?	integer	Specifies the number of bits per component in the downsampled image when *DownsampleImages* = *true*. Default = -1, which forces the downsampled image to have the same number of bits per sample as the original image.
ImageDownsampleThreshold ?	number	Sets the image downsample threshold for images. This is the ratio of image resolution to output resolution above which downsampling may be performed. Allowable values must be between 1.0 through 10.0, inclusive. If the threshold is set out of range, the value reverts to a default of 2.0. The following short examples provide a hypothetical configuration: To use *ImageDownsampleThreshold*, set the following attributes to the values indicated: *ImageResolution* = 72 *ImageDownsampleThreshold* = 1.5 The input image would not be downsampled unless it has a resolution greater than trunc ((72 * 1.5) + .5) = 108dpi
ImageDownsampleType ?	enumeration	Downsampling algorithm for images. Possible values are: *Average* – The program averages groups of samples to get the new downsampled value. *Bicubic* – The program uses bicubic interpolation on a group of samples to get a new downsampled value. *Subsample* – The program picks the middle sample from a group of samples to get the new downsampled value.
ImageFilter ?	enumeration	Specifies the compression filter to be used for images. Ignored if *AutoFilterImages* = *true* or if *EncodeImages* = *false*. Possible values are: *CCITTFaxEncode* – Used to select CCITT Group 3 or 4 facsimile encoding. Used only if *ImageType* = *monochrome*. *DCTEncode* – Used to select JPEG compression. *FlateEncode* – Used to select ZIP compression. If *DCTEncode* is specified, it is only used if the output image has 8 bits per color component, i.e., if *ImageDepth* is 8, or if it is –1 and the original image has 8 bits per color component. Otherwise, *FlateEncode* is used regardless of the value of *ImageFilter*.
ImageResolution ?	number	Specifies the minimum resolution for downsampled color images in dots per inch. This value is used only when *DownsampleImages* is *true*. The application downsamples to that actual resolution. The legal values are from 9.0 to 2400.0, inclusive.

Name	Data Type	Description
ImageType	enumeration	Specifies the kind of images that are to be manipulated. Possible values are: *Color* *Grayscale* *Monochrome*

7.2.70 ImageReplacementParams

This resource specifies parameters required to control image replacement within production workflows.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *ImageReplacement*
Output of processes: -

Resource Structure

Name	Data Type	Description
ImageReplacementStrat-egy	enumeration	Identifies how externally referenced images will be handled within the associated process. Possible values are: *Omit* – Complete process maintaining only references to external data. *Proxy* – Complete process using available proxy images. *Replace* – Replace external references with image data during processing. *AttemptReplacement* – Attempt to replace external references with image data during processing. If replacement fails, complete the process using available proxy images.
MaxResolution ? Deprecated in JDF 1.1	double	Reduces the resolution of images with a resolution higher than *MaxResolution.* Default = 0, which means "do not downsample." Replaced with a link to ImageCompressionParams in the process.
MinResolution ?	double	Specifies the minimum resolution that an image must have in order to be embedded. Default = 0, which means "don't care"
ResolutionReduction-Strategy ? Deprecated in JDF 1.1	enumeration	Identifies the mechanism used for reducing the image resolution. Possible values are: *Downsample* – Default value. *Subsample* *Bicubic* Replaced with a link to ImageCompressionParams in the process.

Name	Data Type	Description
IgnoreExtensions ?	NMTOKENS	Identifies a set of filename extensions that will be trimmed during searches for high-resolution images. These extensions are what will be stripped from the end of an image name to find a base name. The leading dot "." is included. Examples include: *.lay* *.e* *.samp* Default = an empty list
MaxSearchRecursion ?	integer	Identifies how many levels of recursion in the search path will be traversed while trying to locate images. A value of 0 indicates that no recursion is desired.
FileSpec + New in JDF 1.1	refelement	Specification of the paths to search when trying to locate the referenced data. The *ResourceUsage* attribute must be "*SearchPath*". Filespec replaces the SearchPath text element.
SearchPath + Deprecated in JDF 1.1	telem	String that identifies the paths to search when trying to locate the referenced data.

7.2.71 ImageSetterParams

This resource specifies the settings for the imagesetter. A number of settings are OEM-specific, while others are so widely used they may be supported between vendors. Both filmsetter settings and platesetter settings are described with this resource.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ImageSetting*
Output of processes: -

Resource Structure

Name	Data Type	Description
AdvanceDistance ?	double	Additional media advancement beyond the media dimensions on a roll-fed device.
BurnOutArea ? New in JDF 1.1	XYPair	Size of the burnout area. The area defined by *BurnOutArea* is exposed, regardless of the size of the image. Default = 0 0, i.e., only the area defined by the image is exposed.
CenterAcross ?	enumeration	This attribute specifies the axis around which a device may center an image, if the device is capable of doing so. Possible values are: *None* – Default value. *FeedDirection* – Image is centered around the feed-direction axis. *MediaWidth* – Image is centered around the media-width axis. *Both* – Image is centered around both axes.
CutMedia ?	boolean	Indicates whether or not to cut the media (roll-fed). Default = system specified.

Name	Data Type	Description
MirrorAround ?	enumeration	This attribute specifies the axis around which a device may mirror an image, if the device is capable of doing so. Possible values are: *None* – Used if the device is incapable of mirroring an image. Default value. *FeedDirection* – Image is mirrored around the feed-direction axis. *MediaWidth* – Image is mirrored around the media-width axis. *Both* – Image is mirrored around both possible axes.
Polarity ?	enumeration	Some devices can invert the image (in hardware). Possible values are: *Positive* – Default value. *Negative*
Punch ?	boolean	If *true*, indicates that the device may create registration punch holes. Default = *false*
PunchType ?	string	Name of the registration punch scheme, e.g., *Bacher*.
Resolution ?	XYPair	Resolution of the output
RollCut ?	double	Length of media to be cut off of a roll in points.
TransferCurve ?	TransferFunction	Area coverage correction of the device.
Media ? New in JDF 1.1	refelement	Describes the media to be used.

7.2.72 Ink

Resource describing what kind of ink or other colorant (such as toner or varnish) is to be used during printing or varnishing. The default unit of measurement for Ink is *Unit* = "g" (gram).

Resource Properties
Resource class: Consumable
Resource referenced by: **ConventionalPrintingParams**
Example Partition: *FountainNumber, Separation, SheetName, Side, SignatureName, WebName*
Input of processes: **ConventionalPrinting**, **DigitalPrinting**
Output of processes: -

Resource Structure

Name	Data Type	Description
ColorName ?	string	Link to a definition of the color specifics. The value of *ColorName* color should match the *Name* attribute of a Color defined in a **ColorPool** resource that is linked to the process using the **Ink** resource.
Family ?	NMTOKEN	Ink family. Possible values include: *HKS* *Pantone* *TOYO* *ISO* *EURO* *InkJet* It is also possible to specify liquids that are similar to ink. Possible values of this type include:

Name	Data Type	Description
		Varnish
		Silicon
		Toner
InkName ? Modified in JDF 1.1	string	The name of ink is dependent on its *Family*. For example, the *InkName 138 CVC* is a member of the *Pantone Family*.
SpecialInk ?	NMTOKEN	Specific ink attributes. Possible values include: *Metallic*
SpecificYield ?	double	Weight per area at total coverage in g/m2.

7.2.73 InkZoneCalculationParams

This resource specifies the parameters for the **InkZoneCalculation** process.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *TileID, WebName*
Input of processes: *InkZoneCalculation*
Output of processes: -

Resource Structure

Name	Data Type	Description
FountainPositions ?	NumberList	Even number of positions. Each pair specifies the begin and end of the ink slides belonging to a certain fountain. The positions are in coordinates of the printable width along the cylinder axis. The first pair is associated to the first fountain position (corresponds to the partition FountainNumber = 0), the second to the second position (FountainNumber = 1), etc.
PrintableArea ?	rectangle	Position and size of the printable area of the print cylinder in the coordinates of the **Preview** resource. The Partition *TileID* must be used for each plate together with this attribute in case of multiple plates per cylinder. Multiple plates per cylinder may be used in web printing. Default = the complete image
ZoneHeight ?	double	The width of one zone in the feed direction of the printing machine being used.
ZoneWidth	double	The width of one zone of the printing machine being used. Typically, the width of a zone is the width of an ink slide.
Zones	integer	The number of ink zones of the press.
ZonesY ?	integer	Number of ink zones in feed direction of the press. Default = 0, which means not required.

7.2.74 InkZoneProfile

This resource specifies ink zone settings that are specific to the geometry of the printing device being used. **InkZoneProfile**s are independent of the device details.

Resource Properties
Resource class: Parameter
Resource referenced by: -

Example Partition: *FountainNumber, Separation, SheetName, Side, SignatureName, WebName*
Input of processes: *ConventionalPrinting*
Output of processes: *InkZoneCalculation*

Resource Structure

Name	Data Type	Description
ZoneHeight ?	double	The width of one zone in the Y-Direction of the printing machine being used.
ZoneSettingsX	NumberList	Each entry of the *ZoneSettingsX* attribute is the value of one ink zone. The first entry is the first zone, and the number of entries equals the number of zones of the printing device being used. Allowed values are in the range [0..1] where 0 is no ink and 1 is 100% coverage.
ZoneSettingsY ?	NumberList	Each entry of the *ZoneSettingsY* attribute is the value of one ink zone in Y-Direction. The first entry is the first zone and the number of entries equals the number of zones of the printing device being used. Allowed values are in the range [0..1] where 0 is no ink and 1 is 100% coverage.
ZoneWidth	double	The width of one zone of the printing machine being used.

7.2.75 InsertingParams

This resource specifies the parameters for the *Inserting* process. Figure 7.13 shows the various components involved in an inserting process, and how they interact.
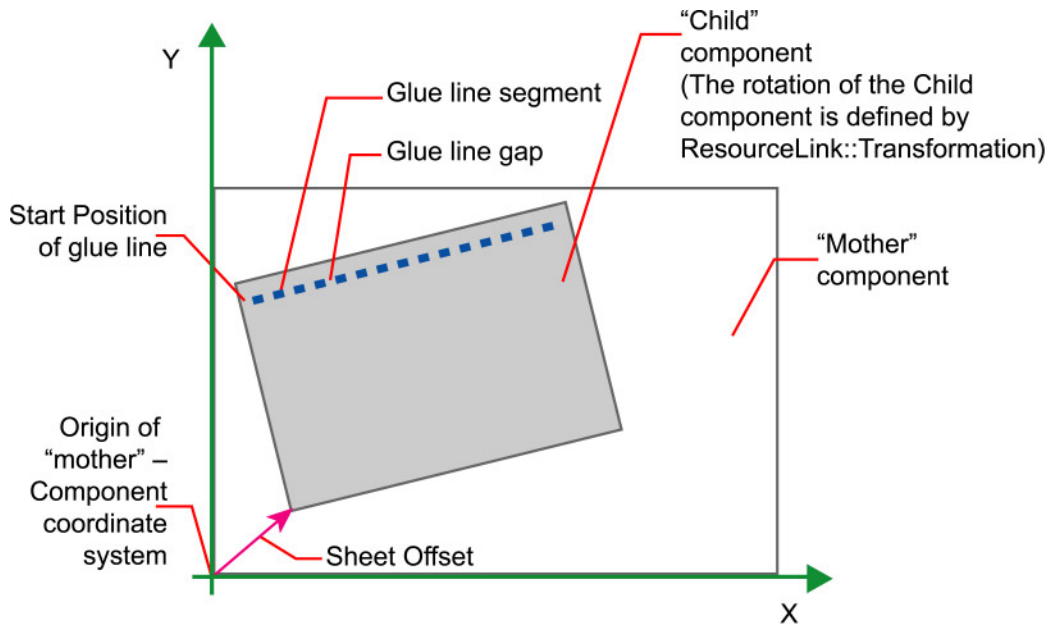
Figure 7.15 Parameters and Coordinate system used for Inserting

The process coordinate system is defined as follows: The Y-axis is aligned with the binding edge and increases from the registered edge to the edge opposite the registered edge. The X-axis, meanwhile, is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge, which is the product front edge.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -

Input of processes: *Inserting*
Output of processes: -

Resource Structure

Name	Data Type	Description
SheetOffset Deprecated in JDF 1.1	XYPair	Offset between the sheet to be inserted and the "mother" sheet. SheetOffset is implied by the Transformation matrix in Re-sourceLink:*Transformation* of the child's ComponentLink.
Location	enumeration	Where to place the "child" sheet. Possible values are: *Front* *Back* *OverfoldLeft* *OverfoldRight*
GlueLine *	element	Array of all GlueLine elements. The coordinate system is de-fined by the mother **Component**.

7.2.76 InsertSheet

InsertSheet resources define device generated images and sheets which may be produced along with the job. **In-sertSheet**s include separators sheets, error sheets, accounting sheets, and job sheets. The information provided on the sheet depends on the type of sheet. In some cases, an *Imposition* process may encounter RunList elements that do not provide enough pages to complete a **Layout** resource or its children. **InsertSheet** resources are used to provide a standard way of completing such **Layout** resources. **InsertSheet** resources may also be used to start new **Sheet** resources, e.g., to ensure that a new chapter starts on a right-hand page. In addition, **InsertSheet** may specify whether new media should be inserted, once the current **Sheet, Signature**, instance document, or job is completed.

InsertSheets may be used at the beginning or end of **RunLists** with a *SheetUsage* attribute of *Header* or *Trailer*. When an **InsertSheet** appears both in a **RunList** and in a **Layout** and/or **Sheet**, the following prece-dence applies:

1. The **InsertSheet** with *Usage FillSurface* from the **RunList** is applied first.
2. The **InsertSheet** with *Usage FillSheet* from the **RunList** is applied.
3. The **InsertSheet** with *Usage FillSignature* from the **RunList** is applied.
4. After completely processing the **RunList InsertSheets** once, apply the **Surface**, **Sheet**, and **Signa-ture InsertSheets**.

If the **InsertSheet's RunList** does not supply enough content to fill a **Sheet**, **Signature**, or **Surface**, the **RunList** will be reapplied until no **PlacedObject** slots remain to be filled. When an **InsertSheet** is used in a **RunList** of a process that does not use a **Layout** or **LayoutPreparationParams** resource, i.e., that process has not been combined with *Imposition* or *LayoutPreparation*, only *Usage Header* or *Trailer* are valid.

Resource Properties

Resource class: Parameter
Resource referenced by: Disjointing, Layout, LayoutPreparationParams, RunList, Sheet
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
IsWaste ?	boolean	Specifies whether the InsertSheet is waste that should be removed from the document before further processing. Default = *true*, i.e., the **InsertSheet** is to be discarded when finishing the document.
MarkList ?	NMTOKENS	List of marks that should be marked on this **InsertSheet**. Ignored

Name	Data Type	Description
New in JDF 1.1		if a Sheet is specified in this **InsertSheet**. Values include: *CIELABMeasuringField* *ColorControlStrip* *ColorRegisterMark* *CutMark* *DensityMeasuringField* *IdentificationField* *JobField* *PaperPathRegisterMark* *RegisterMark* *ScavengerArea*
SheetFormat ? New in JDF 1.1	NMTOKEN	Identifies that device-dependent information is to be included on the **InsertSheet**. Possible values include: *Blank* *Brief* *Full* *Standard* *SystemSpecified* Default = *SystemSpecified*
SheetType New in JDF 1.1	enumeration	Identifies the type of sheet. Possible values are: *AccountingSheet* – A sheet that reports accounting information for the job. *ErrorSheet* – A sheet that reports errors for the job. *FillSheet* – A sheet that fills ContentObjects with no matching entry in the content **RunList**. *InsertSheet* – A sheet that is inserted to the job, e.g. a preprinted cover . *JobSheet* – A sheet that delimits the job. *SeparatorSheet* – A sheet that delimits pages, sections, copies or instance documents of the job.
SheetUsage New in JDF 1.1	enumeration	Indicates where this **InsertSheet** is to be produced and inserted into the set of output pages. Possible values are: *FillForceBack* - Valid for *SheetType* = *FillSheet*. Contents of the **RunList** of the **InsertSheet** are used to fill the current sheet before forcing the next page of the content **Runlist** to the back side of the next sheet if not already on a back surface. *FillForceFront* - Valid for *SheetType* = *FillSheet*. Contents of the **RunList** of the **InsertSheet** are used to fill the current sheet before forcing the next page of the content **Runlist** to the front side of the next sheet if not already on a front surface. *FillSheet* – Valid for *SheetType* = *FillSheet*. Contents from the **RunList** of the **InsertSheet** are used to fill the current sheet. *FillSignature* – Valid for *SheetType* = *FillSheet*. Contents from the **RunList** of the **InsertSheet** are used to fill the current signature. *FillSurface* – Valid for *SheetType* = *FillSheet*. Contents from the **RunList** of the **InsertSheet** are used to fill the current surface.

Name	Data Type	Description
		Header – Valid for *SheetType = InsertSheet, JobSheet, Separator-Sheet.* The sheet is produced at the begin of the job (for *JobSheet*) or at the begin of each copy of each instance document (for *SeparatorSheet*) or is prepended before the current sheet, signature, layout, or **RunList** as defined by its context. Contents for the **Sheet** are drawn from the **RunList** included in this **InsertSheet** resource, if one is included. If a **RunList** is not included, the inserted sheet filled with system specified content defined by *SheetType.*
		Interleaved – Valid for *SeparatorSheet.* The sheet is produced after each page. Used e.g. to insert sheets under transparencies. Contents for the **Sheet** are drawn from the **RunList** included in this **Insert-Sheet** resource, if one is included. If a **RunList** is not included, the inserted sheet filled with system specified content defined by *SheetType* = *SeparatorSheet.*
		OnError – Valid for *SheetType = ErrorSheet.* The sheet is produced at the end of the job when an error or warning occurs.
		Slip – Valid for *SeparatorSheet.* The sheet is produced between each copy of each instance document. Contents for the **Sheet** are drawn from the **RunList** included in this **InsertSheet** resource, if one is included. If a **RunList** is not included, the inserted sheet filled with system specified content defined by *SheetType = SeparatorSheet.*
		Trailer – Valid for *SheetType = AccountingSheet, ErrorSheet*, InsertSheet, *JobSheet, SeparatorSheet.* The sheet is produced at the end of the job (for *AccountingSheet, ErrorSheet, JobSheet*) or at the end of each copy of each instance document (for *SeparatorSheet*) or is appended after the current sheet, signature, layout, or **RunList** as defined by its context. Contents for the **Sheet** are drawn from the **RunList** included in this **InsertSheet** resource, if one is included. If a **RunList** is not included, the inserted sheet filled with system specified content defined by *SheetType.*
Usage ? Deprecated in JDF 1.1	enumeration	Replaced by *SheetUsage*.
RunList ?	refelement	A **RunList** that provides the content for the **InsertSheet**. Any **InsertSheet** resources referenced by this **RunList** are ignored.
Sheet ?	refelement	Details of the **Sheet** that will be inserted. Contents for this Sheet are drawn from the **RunList** included in this InsertSheet, if any. If not specified, the system specified insert sheets are used. Any **In-sertSheet** resources referenced by this **Sheet** are ignored.

7.2.77 InterpretedPDLData

Represents the results of the PDL Interpretation process. The details of this resource are not specified, as it is assumed to be implementation dependent.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Rendering*
Output of processes: *Interpreting*

7.2.78 InterpretingParams

The **InterpretingParams** resource contains the parameters needed to interpret PDL pages. The resource itself is a generic resource that contains attributes that are relevant to all PDLs. PDL-specific instances of **InterpretingParams** resources may be included as subelements of this generic resource. This specification defines one additional PDL-specific resource instance: **PDFInterpretingParams**.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *Interpreting*
Output of processes: -

Structure of the InterpretingParams Resource

Name	Data Type	Description
Center ?	boolean	Indicates whether or not the page image should be centered within the imagable area of the media. Default = *false. Center* is ignored if **FitPolicy**::*SizePolicy=ClipToMaxPage* and clipping is required.
FitToPage ? Deprecated in JDF 1.1	boolean	Specifies whether the page contents should be scaled to fit the media. Default = *false*
MirrorAround ?	enumeration	This attribute specifies the axis around which a RIP may mirror an image. Note: This is mirroring in the RIP and not in the hardware of the output device. Possible values are: *None* – Default value. *FeedDirection* – Image is mirrored around the feed-direction axis. *MediaWidth* – Image is mirrored around the media-width axis. *Both* – Image is mirrored around both possible axes.
Polarity ?	enumeration	The image must be RIPped in the polarity specified. Note that this is a polarity change in the RIP and not a polarity change in the hardware of the output device. Possible values are: *Positive* – Default value. *Negative*
Poster ?	XYPair	Specifies whether the page contents should be expanded such that each page covers X by Y pieces of media. Default = 1 1
PosterOverlap ?	XYPair	This pair of real numbers identifies the amounts of overlap in points, that must be calculated for the poster tiles across the horizontal and vertical axes, respectively. Default = 0 0
PrintQuality ? New in JDF 1.1	enumeration	Generic switch for setting the quality of an otherwise inaccessible device. Possible values are: *High* – Highest quality available on the printer. *Normal* – The default quality provided by the printer. The default. *Draft* – Lowest quality available on the printer.

Name	Data Type	Description
Scaling ?	XYPair	A pair of positive real values that indicates the scaling factor for the page contents. Values between 0 and 1 specify that the contents are to be reduced, while values greater than 1 specify that the contents are to be expanded. This attribute is ignored if *FitToPage* = *true* or if *Poster* is present and has a value other than "1 1". Any scaling defined in **FitPolicy** must be applied after the scaling defined by this attribute. Default = 1. 1
ScalingOrigin ?	XYPair	A pair of real values that identify the point in the unscaled page that is to become the origin of the new, scaled page image. This point is defined in the coordinate system of the unscaled page. Default = 0 0
ObjectResolution *	refelement	Indicates the resolution at which the PDL contents will be inter-preted in DPI. These elements may be different from the ObjectResolution elements provided in the **RenderingParams** resource. Default = system specified
FitPolicy ? New in JDF 1.1	refelement	Allows printing even if the media size does not match the require-ments of the data. This replaces the deprecated *FitToPage* attrib-ute. This **FitPolicy** element must be ignored in a combined process with **LayoutPreparation**.
Media ? New in JDF 1.1	refelement	This resource provides a description of the physical media which will be marked. The physical characteristics of the media may af-fect decisions made during **Interpreting**.
PDFInterpreting-Params ? New in JDF 1.1	refelement	Details of interpreting for PDF. Note that this is a subelement in JDF 1.1, and not an instance as in JDF 1.0.

Structure of PDFInterpretingParams Subelement
New in JDF 1.1

Name	Data Type	Description
EmitPDFBG ?	boolean	Indicates whether BlackGeneration functions should be emitted. Default = *true*
EmitPDFHalftones ?	boolean	Indicates whether Halftones should be emitted. Default = *true*
EmitPDFTransfers ?	boolean	Indicates whether Transfer functions should be emitted Default = *true*
EmitPDFUCR ?	boolean	Indicates whether UnderColorRemoval functions should be emit-ted. Default = *true*
HonorPDFOverprint ?	boolean	Indicates whether or not overprint settings in the file will be hon-ored. If *true*, the setting for overprint will be honored. If *false*, it is expected that the device does not directly support overprint and that the PDF is preprocessed to simulate the effect of the overprint set-tings. Default = *true*

Name	Data Type	Description
ICCColorAsDeviceColor ?	boolean	Indicates whether colors specified by ICC colorspaces should be treated as device colorants. Default = *false*
PrintPDFAnnotations ?	boolean	Indicates whether the contents of annotations on PDF pages should be included in the output. This only refers to annotations that are set to print in the PDF file. Default = *false*
TransparencyRenderingQuality ?	number	Possible values are 0 to 1. 0 represents the lowest allowable quality. 1 represents the highest desired quality. Default = use device settings

7.2.79 JacketingParams

New in JDF 1.1

Description of the setup of the jacketing machinery. Jacket height and width (1 and 3 in the figure below) are specified within the **Component** that describes the jacket.

1: Jacket width
2: Folding width
3: Jacket height
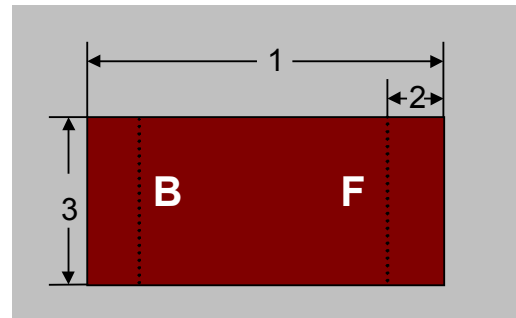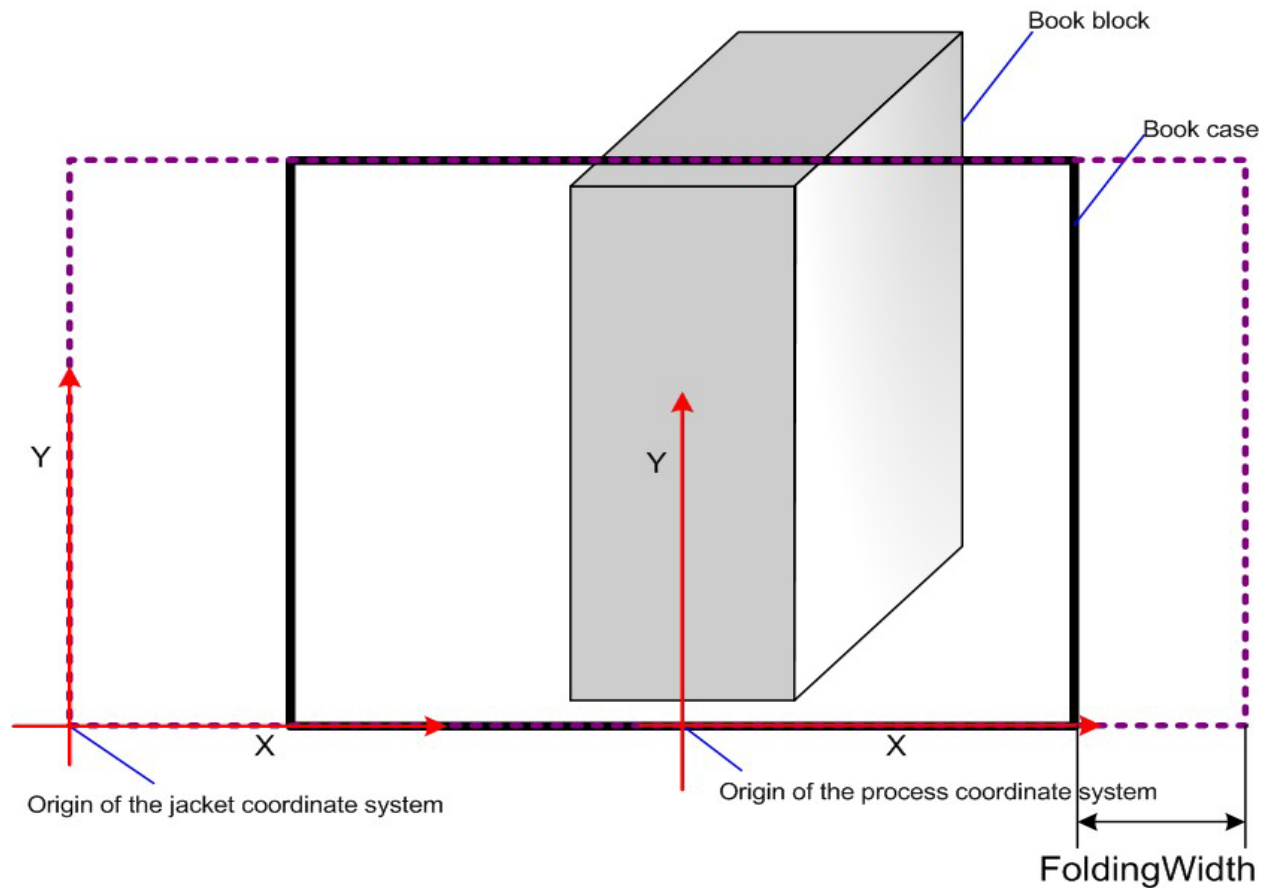
Figure 7.16 Parameters and Coordinate System for Jacketing

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Jacketing*

Resource Structure

Name	Data Type	Description
FoldingWidth	number	Definition of the dimension of the folding width of the front cover fold (see "2" in the picture above). All other measurements are implied by the dimensions of the book.

7.2.80 JobField

New in JDF 1.1

A **JobField** is a Mark object that specifies the details of a job.

Resource Properties

Resource class: Parameter
Resource referenced by: Surface
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
ShowList	NMTOKENS	List of elements to display in the **JobField**. Values include:
		DeviceID – ID of the device. This is a unique name within the workflow.
		EndTime – Actual EndTime of the job.
		Error – Errors that happened during the job.
		FriendlyName – *FriendlyName* of the device.
		JobName – DescriptiveName of the node that is executing.
		JobRecipientName – Name of the recipient of the job
		JobSubmitterName – Name of the submitter of the job
		StartTime – Actual *StartTime* of the job.
		MediaBrand – Brand of the media that is being printed.
		MediaType – DescriptiveName of the media that is being printed.
		Operator – Name of the Operator.
		OperatorText – Message to the Operator as defined in *OperatorText*
		Resolution – Output resolution.
		ResolutionX – Output resolution in X direction.
		ResolutionY – Output resolution in Y direction.
		ScreeningFamily – Name of the screening family of the output.
		UserText – User defined text as defined in *UserText*.
OperatorText ?	string	Text to the operator.
UserText ?	string	User defined text to output with JobField.
DeviceMark ?	refelement	DeviceMark defines the formatting parameters for the mark. If not specified, the DeviceMark settings defined in **LayoutPreparationParams** or in the **Layout** tree are assumed.

7.2.81 LabelingParams

New in JDF 1.1

LabelingParams defines the details of the **Labeling** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Labeling*
Output of processes: -

Resource Structure

Name	Data Type	Description
Application ?	NMTOKEN	Application method of the label. Includes: *Loose* – Loosely laid onto the component. *Staple* – Stapled onto the component. *SelfAdhesive* – Self adhesive label *Glue* – Glued onto the component. *Any* – The default.
CTM ?	matrix	Position and orientation of the label lower left corner relative to the lower left corner of the component surface as defined by *Position*. Default = device dependent
Position ?	enumeration	Position of the label on the bundle. One of: *Top* *Bottom* *Left* *Right* *Front* *Back* *An* – The default.

7.2.82 LaminatingParams

New in JDF 1.1

This resource specifies the parameters needed for laminating.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *SheetName, Side*
Input of processes: *HoleMaking.*
Laminating
Output of processes: -

Resource Structure

Name	Data Type	Description
AdhesiveType ?	string	Type of adhesive used. Default = the empty string, i.e., no adhesive is used. Valid only when *LaminatingMethod = Dispersion-Glue.*
GapList ?	NumberList	List of non laminated gap positions in the X direction of the laminating tool in the coordinate system of the **Component**. The zero-based even entries define the absolute position of the start of a gap, and the odd entries define the end of a gap. If not specified, the complete area defined by *LaminatingBox* is laminated.
HardenerType ?	string	Type of hardener used. Default = the empty string, i.e., no hardener is used. Valid only when *LaminatingMethod = Dispersion-Glue.*
LaminatingBox	rectangle	Area on the **Component** to be laminated.

Name	Data Type	Description
LaminatingMethod ?	enumeration	Laminating technology that is applied. One of: *CompoundFoil* *DispersionGlue* *Unknown*
Temperature ?	number	Temperature used in the lamination process in ° Centigrade. Default =system specified.

7.2.83 Layout

Represents the root of the layout structure. **Layout** is used both for fixed-layout and for automated printing.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: It is strongly discouraged to partition the **Layout** tree, including **Sheet** and **Surface**.
Input of processes: *ConventionalPrinting*, *DigitalPrinting, Imposition*, *InkZoneCalculation*, *Proofing*, *SoftProofing*
Output of processes: *LayoutPreparation*

Resource Structure

Name	Data Type	Description
Automated ?	boolean	If *true*, the **Imposition** process is expected to perform automated page consumption. Automated page consumption is equivalent to the PrintLayout functionality provided in PJTF. Default = *false*
MaxDocOrd ? New in JDF 1.1	integer	Zero based maximum number of instance documents that are consumed from a **RunList** each time the **Layout** is executed, assuming the **Imposition** process is automated. Default = 1.
MaxOrd ?	integer	Zero based maximum number of placed objects that are consumed from a **RunList** each time the **Layout** is executed, assuming the ***Imposition*** process is automated. Default = –1, i.e., it is unknown and must be calculated from the *Ord* values of the ContentObject s in the Layout.
MaxSetOrd ? New in JDF 1.1	integer	Zero based maximum number of document sets that are consumed from a **RunList** each time the **Layout** is executed, assuming the ***Imposition*** process is automated. Default = 1.
Name ? New in JDF 1.1	string	Unique name of the Layout. *Name* is used for external reference to a Layout.
InsertSheet *	refelement	Additional sheets that should be inserted before and/or after a document.
LayerList ? New in JDF 1.1	element	List of LayerDetails elements.
Media ? New in JDF 1.1	refelement	Describes the media to be used.
MediaSource ? Deprecated in JDF 1.1	refelement	Describes the media to be used. Replaced by **Media** in JDF 1.1.
Signature *	element	The signatures that are defined by the layout.

Name	Data Type	Description
TransferCurvePool ? New in JDF 1.1	refelement	Describes the relationship of transfer curves and coordinate systems within the various processes.

Structure of LayerList Subelement
New in JDF 1.1

This element provides a container for an ordered list of LayerDetails elements. The individual elements are referenced by their zero based index in the LayerList.

Name	Data Type	Description
LayerDetails *	refelement	Details of the individual layers.

Structure of LayerDetails Subelement
New in JDF 1.1

This element provides a container for LayerDetails elements.

Name	Data Type	Description
Name ?	string	Unique name of the layer.

Structure of Signature Subelement
This element groups individual **Sheet** resources into one Signature subelement.

Name	Data Type	Description
Name ?	string	Unique name of the signature. *Name* is used for external reference to a signature, as in a Part element.
InsertSheet *	refelement	Specifies how to complete a signature in an automated printing environment.
Media? New in JDF 1.1	refelement	Describes the media to be used.
MediaSource ? Deprecated in JDF 1.1	refelement	Describes the media to be used. Replaced by **Media** in JDF 1.1.
Sheet *	refelement	**Sheet** resources that comprise the signature.

7.2.84 LayoutElement

This resource is needed for *LayoutElementProduction*. It describes some text, an image, one or more pages, or anything else that is used in the production of the layout of a product.

Resource Properties
Resource class: Parameter
Resource referenced by: RunList, Surface
Example Partition: *PageNumber*
Input of processes: *DBDocTemplateLayout*, *DBTemplateMerging*, *LayoutElementProduction*
Output of processes: *DBDocTemplateLayout*, *LayoutElementProduction*

Resource Structure

Name	Data Type	Description
ClipPath ?	path	Path that describes the outline of the **LayoutElement** in the coordinate space of the **LayoutElement** of *ElementType Page* that results from the **LayoutElementProduction** process. Default = no clip path
ElementType ?	enumeration	Describes the content type for this **LayoutElement**. Possible values are: *Text* – Formatted or unformatted text. *Image* – Bitmap image. *Graphic* – Line art. *Reservation* – Empty element. Content for this area of the page may be provided by a subsequent process. *Composed* – Combination of elements that define an element that is not bound to a document page. *Page* – Representation of one document page. *Document* – An ordered set of one or more pages. *MultiDocument* – An ordered set of one or more Documents including document breaks, e.g., PPML, PPML/VDX, mime multipart/related. *Surface* – Representation of an imposed surface. *Tile* – Representation of the contents of one tile. *Unknown* – Unknown element type or any of the above. The default.
HasBleeds ?	boolean	If *true*, the file has bleeds. Default = *false*.
IgnorePDLCopies ? New in JDF 1.1	boolean	If *true*, any PDL defined copy count must be ignored. Default = *false*.
IgnorePDLImposition ? New in JDF 1.1	boolean	If *true*, any PDL defined imposition definition must be ignored. Examples are PDF with embedded PJTF or PPML with a PRINT_LAYOUT. If *IgnorePDLImposition=false*, and JDF also defines imposition, the imposed sheets of the PDL are treated as pages in the context of JDF imposition. The front and back surfaces of the pdl and JDF imposition should be matched. Note that it is strongly discouraged to specify imposition both in the PDL and JDF and that this may result in undesired behavior. Default = *true*.
IsPrintable ?	boolean	If *true*, the file is a PDL file and can be printed. Possible files types include PCL, PDF or PostScript files. Application files such as MS Word have *IsPrintable="false"*. *Default = true*.
IsTrapped ?	boolean	If *true*, the file has been trapped. Default = *false*.
SourceBleedBox ?	rectangle	A rectangle that describes the bleed area of the element to be included. This rectangle is expressed in the default user space. Default uses element's defined bleed box (or no bleed box if element does not supply a bleed box)

Name	Data Type	Description
SourceClipBox ?	rectangle	A rectangle that defines the region of the element to be included. This rectangle is expressed in the default user space of the source document page. Default = use element's defined clip box (or no clip box if element does not supply a clip box)
SourceTrimBox ?	rectangle	A rectangle that describes the intended trimmed size of the element to be included. This rectangle is expressed in the default user space. Default uses element's defined trim box (or no trim box if element does not supply a trim box)
Template ?	boolean	*Template* is *false* when this layout element is self-contained. This attribute is *true* if the **LayoutElement** represents a template that must be completed with information from a database. Default = *false*
FileSpec	refelement	URL + metadata about the physical characteristics of a file representing the **LayoutElement**.
SeparationSpec *	element	List of used separation names.

7.2.85 LayoutPreparationParams

New in JDF 1.1

This resource provides the parameters of the *LayoutPreparation* process, which provides the details of how page contents will be imaged onto media. This resource has a provision for specifying either a multi-up grid of content page cells or an imposition layout of finished pages.

A multi-up grid of pages can be step and repeated across, down, or through a stack of sheets in any axis order. Note: For all resources, the coordinate system for all parameters is defined with respect to the process coordinate system as defined in Section 2.5.3 Coordinate Systems of Resources and Processes.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex*, *DocRunIndex*, *RunIndex*, *SetIndex*, *SheetName*-
Input of processes: *LayoutPreparation*
Output of processes: -

Resource Structure

Name	Data Type	Description
BackMarkList ?	NMTOKENS	List of marks that should be marked on each back surface. The appearance of the marks are defined by the process implementation. Values include: *CIELABMeasuringField*: *ColorControlStrip*: *ColorRegisterMark*: *CutMark*: *DensityMeasuringField*: *IdentificationField*: *JobField* *PaperPathRegisterMark*: *RegisterMark*:

Name	Data Type	Description
		ScavengerArea:
CreepValue ?	XYPair	This parameter determines a user defined values for horizontal and vertical creep compensation. The number specifies the distance in points by which the respective gutter that creeps either increments or decrements in width from one sheet to the next for a given sequence of sheets related to the same bound component.
		If the value of a component of this attribute is positive, it specifies the amount in points by which the width of creeping gutters are incremented. If the value of a component of this attribute is negative then it specifies the amount in points by which the width of creeping gutters are decremented.
		An explicit value of "0" means that the creep compensation value for the respective axis is system specified, for example, it may be calculated based on the information taken from *Media*.
		If the *CreepValue* attribute is not present its value defaults to 0.
		NOTE: Creep is disabled for the respective axis when the *HorizontalCreep* and *VerticalCreep* attributes respectively are not present in which case the appropriate component of *CreepValue* must be ignored.
FinishingOrder ?	enumeration	Specifies the order of operations for finishing a bound booklet created from multiple imposed sheets.
		The LayoutPreparation process needs this information in order to completely determine content page distribution onto the sequence of sheets comprising the pages of a single booklet under consideration of the values of the PageDistributionScheme and FoldCatalog attributes.
		Possible values are:
		FoldGather – The sheets of a document are first folded according to the value of the FoldCatalog attribute and then gathered on a pile. Usually applies to finishing of perfect bound documents.
		FoldCollect – The sheets of a document are first folded according to the value of the FoldCatalog attribute, and then collected on a saddle. Usually applies to finishing of both perfect bound and saddle-stitched booklets.
		Gather – The sheets of a document are gathered on a pile. No folding is assumed.
		GatherFold – The sheets of a document are first gathered on a pile, then folded according to the value of the FoldCatalog attribute. Usually applies to finishing of both perfect bound and saddle-stitched booklets. The default.
FoldCatalog ?	string	Description of the type of fold that will be applied to all printed sheets according to the folding catalog in the format "Fx-y" as shown in Figure 7.11 and Figure 7.12.
		The LayoutPreparation process uses the fold description specified by this attribute in the determination of the proper distribution of pages onto the surfaces of the sheets in the context of the values of both the PageDistributionScheme and FinishingOrder attributes.
		If not present, no folding other than the folding that is implied by *PageDistributionScheme*=*Saddle* is assumed.
FrontMarkList ?	NMTOKENS	List of marks that should be marked on each front surface. The appearance of the marks are defined by the process implementation. Values include:

Name	Data Type	Description
		CIELABMeasuringField:
		ColorControlStrip:
		ColorRegisterMark:
		CutMark:
		DensityMeasuringField:
		IdentificationField:
		JobField
		PaperPathRegisterMark:
		RegisterMark:
		ScavengerArea:
Gutter ? Modified in JDF 1.1A	XYPair	Width in points of the horizontal and vertical gutters formed between rows and columns of pages of a multi-up sheet layout. The first value specifies the width of all horizontal gutters and the second value specifies the width of all vertical gutters. If no gutters are defined because either the NumberUp attribute is not present, or its explicit values are equal to one, this attribute must be ignored. In the case where a gutter is identified as creeping by either the VerticalCreep or HorizontalCreep attributes, then the value of Gutter specifies the initial gutter width where the gutter width may increment or decrement depending upon the explicit or implied value of the CreepValue attribute. If not present, the Default=”*0.0 0.0*” which means that the pages of a multi-up grid of pages must touch. The *Gutter* is applied in addition to any *Border* specified in the PageCell.
HorizontalCreep?	IntegerList	Specifies which horizontal gutters creep. The allowed values are zero based indexes that reference horizontal gutters formed by multiple rows of pages in a multi-up page layout specified by the first value of the NumberUp attribute. The value for an entry in this list must be between zero and 1 less then the first value of the NumberUp attribute.
NumberUp ?	XYPair	Specifies a regular, multi-up grid of PageCells into which content pages are mapped. The first value specifies the number of rows of page cells and the second value specifies the number of columns of page cells in the multi-up grid. The relative positioning of the page cells within the multi-up grid are defined by the explicit or implied values of the Gutter, HorizontalCreep, VerticalCreep, and CreepValue attributes. The distribution of content pages from the content RunList into the page cells is defined by the explicit or implied values of the PageDistributionScheme, PresentationDirection, Sides, FinishingOrder and FoldCatalog attributes and the implicit number of sheets comprising the bound component.
PageDistribution-Scheme ?	NMTOKEN	This attribute specifies how pages are to be distributed onto a multi-up grid of finished PageCells defined by the values of the NumberUp attribute. Possible values include:

Name	Data Type	Description
		Saddle – Distribute pages onto a sequence of one or more imposition layouts in proper order for saddle stitch binding. For this page distribution scheme, creep should only be applied to odd numbered vertical gutters where any even numbered gutters will automatically creep in the opposite direction.
		Perfect – Distribute pages onto a sequence of one or more signatures in proper order for perfect binding. For this page distribution scheme, creep is usually not used.
		Sequential – The pages are distributed onto the multi-up layout according to the value of the PresentationDirection attribute. The default.
		Note: Page distribution ordering for both Saddle and Perfect also depends upon the implied number of sheets per finished Component and how the imposed sheets are to be folded during finishing as well as the order of gathering and folding. Refer to the FoldCatalog and FinishingOrder attributes.
		Note: The NumberUp attribute must always specify a multi-up layout appropriate for a given page distribution ordering and FoldCatalog. Setting this attribute does not imply the multi-up grid dimensions are appropriate for the selected page distribution scheme.
		Note: In all cases, the order of content pages as represented by the content RunList must be either in reader order or in an order appropriate for multi-up saddle stitching. Refer to the PageOrder attribute.
PageOrder ?	NMTOKEN	The assumed ordering of the content pages in the **RunList**.
		Booklet – The pages are preordered in the **RunList** and must be processed exactly in the order as specified by *PresentationDirection*. *NumberUp* must still be set to the appropriate value and is not implied by specifying *PageOrder=Booklet*. *PageOrder= Booklet* must not be used in conjunction with *FoldCatalog*.
		Reader – The pages are in reader order in the **RunList**.
		The default.

Name	Data Type	Description
Presentation-Direction ?	enumeration	Indicates the order in which content pages will be distributed into the page cells of the NumberUp layout. If PageDistributionScheme="*Saddle*", **PresentationDirection** applies to sets of two adjacent pages. This allows positioning of multiple page pairs for SaddleStitching onto one sheet. Possible values are: *FoldCatalog* – Pages are imaged so that the result is compatible with a finished product produced from the folding catalog as specified in **FoldCatalog**. *SystemSpecified* – Pages are imaged onto the NumberUp layout as determined by the device. *XYZ*: Permutations of the letters XYZ and xyz so that exactly one of upper or lower case of x y and z define the order in which content pages are flowed along each axis with respect to the coordinate system of the front side of the sheet. X Specifies flowing left to right across a sheet surface. x Specifies flowing right to left across a sheet surface. Y Specifies flowing bottom to top vertically across a sheet surface. y Specifies flowing top to bottom vertically across a sheet surface. Z Specifies flowing bottom of stack to top of stack through the stack. z Specifies flowing top of stack to bottom of stack through the stack. If not present, the default value is *SystemSpecified:* The following table specifies how cells are ordered on simplex 4-up depending on XYZ.

XyZ		Zxy		xyz		XYZ	
1 2	5 6	4 2	3 1	2 1	6 5	3 4	7 8
3 4	7 8	8 6	7 5	4 3	8 7	1 2	5 6

Name	Data Type	Description
Rotate ?	enumeration	Orthogonal rotation including the implied translation to be applied to the grid of PageCells on the entire surface relative to the process coordinate system. One of: *Rotate0* *Rotate90* – 90° counterclockwise rotation *Rotate180* – 180° rotation *Rotate270* – 90°clockwise rotation For details of orthogonal rotations, refer to Table 2-3. If a **RotatePolicy** value other than "*NoRotate"* is specified in **FitPolicy**, the value specified in **Rotate** may be modified accordingly. Note: A rotation of the grid also rotates the gutters, i.e., it is applied after all other parameters have been evaluated and applied. Default = *Rotate0*
Sides ?	enumeration	Indicates whether the content layout should be imaged on one or both sides of the media. When a different value for the Sides attribute is encountered, it must force a new sheet. However, when the same value for the Sides attribute is restated for consecutive pages, it is the same as if that re-statement was not present. Possible values are: *OneSidedBackFlipX*– Page content is imaged on the back side of media

Name	Data Type	Description
		so that the corresponding page cells back up to a blank front cell when flipping around the X axis. Equivalent to '*WorkAndTumble*' with a blank front side.
		OneSidedBackFlipY– Page content is imaged on the back side of media so that the corresponding page cells back up to a blank front cell when flipping around the Y axis. Equivalent to '*WorkAndTurn'* with a blank front side.
		OneSidedFront – Page content is imaged on the front side of media. The default.
		TwoSidedFlipX – Page content is imaged on both the front and back sides of media sheets so that the corresponding page cells back up to each other when flipping around the X axis. Equivalent to '*WorkAnd-Tumble*''.
		TwoSidedFlipY– Page content is imaged on both the front and back sides of media sheets so that the corresponding page cells back up to each other when flipping around the Y axis. Equivalent to '*WorkAnd-Turn*'.
StackDepth ?	integer	The number of sheets in a stack that are processed when imposing down the Z access.
		If not specified, the entire job defines one stack.
StepDocs ?	IntegerList	A list of two integers that species the number of instance documents to impose on one sheet. The first value specifies the repeats along the X axis, the second value specifies the repeats along the Y axis. Default="1 1". Each entry of *NumberUp* must be an integer multiple of *StepRepeat * StepDocs.* Positive values define grouped step and repeat whereas negative values define alternating step and repeat. The following examples have *NumberUp*="4 4" and StepRepeat="2 2" and *StepDocs*=:

"2 1"	"1 2"
A1 A1 B1 B1	A1 A1 A2 A2
A1 A1 B1 B1	A1 A1 A2 A2
A2 A2 B2 B2	B1 B1 B2 B2
A2 A2 B2 B2	B1 B1 B2 B2

Name	Data Type	Description
StepRepeat ?	IntegerList	A list of three integers that specifies the number of identical pages to impose. The first value specifies the repeats along the X axis, the second value specifies the repeats along the Y axis and the 3rd value specifies the repeats down the stack – the Z axis. Default="1 1 1". Each entry of *NumberUp* must be an integer multiple of *StepRepeat * StepDocs.* Positive values define grouped step and repeat whereas negative values define alternating step and repeat. Note that negative values are illegal for the 3rd component, since the total depth of the stack may be unknown. The following examples have *NumberUp*="4 4" and *StepRepeat*=:

"2 2 1"	"-2 2 1"	"-2 –2 1"	"2 –2 1"	"1 4 1"
1 1 2 2	1 2 1 2	1 2 1 2	1 1 2 2	1 2 3 4
1 1 2 2	1 2 1 2	3 4 3 4	3 3 4 4	1 2 3 4
3 3 4 4	3 4 3 4	1 2 1 2	1 1 2 2	1 2 3 4
3 3 4 4	3 4 3 4	3 4 3 4	3 3 4 4	1 2 3 4

Name	Data Type	Description
SurfaceContentsBox ? **Modified in JDF 1.1A**	rectangle	This box, specified in surface-coordinate space, defines the area into which PageCells are distributed. The lower left corner of the rectangle specified by the value of this attribute establishes the coordinate system into which the content is mapped onto the surface. Note: *SurfaceContentsBox* does not imply clipping. Clipping is defined by PageCell::*ClipBox*. If *SurfaceContentsBox* is not specified, a device may supply a *SurfaceContentsBox* that corresponds to the imagable area for the **Media** used by the device. Otherwise a rectangle with the origin at "0 0" and the dimensions of the **Media** defined in this resource is assumed. If no **Media** *Dimension* can be determined, the *SurfaceContentsBox* is assumed to have its origin at the lower left corner and be unbounded in X and Y.
VerticalCreep ?	IntegerList	Specifies which vertical gutters creep. The allowed values are zero-based indexes that reference vertical gutters formed by multiple columns of pages in a multi-up page layout specified by the second value of the *NumberUp* attribute. The value for an entry in this list must be between zero and 1 less then the second value of the *NumberUp* attribute. An index value outside of this range is ignored. If not specified then no vertical gutters will creep.
ImageShift ?	element	Details how to place the grid of PageCells onto the media. The coordinate system is defined so that the "X" dimension is the first number of the Media *Dimension* attribute; "Y" is the second number. **ImageShift** must be applied before any transformations of the grid of PageCells as specified by *Rotate* or **FitPolicy.**
InsertSheet *	refelement	Additional sheets to be inserted before, after, or within a job.
DeviceMark ?	refelement	Details how device dependent marks should be generated. If not specified, the marks are device dependent.
FitPolicy ?	refelement	Details how to fit the grid of PageCells onto the media.
JobField *	refelement	Specific information about this kind of mark object.
Media ?	refelement	Specific information about the media.
PageCell ? **Modified in JDF 1.1A**	refelement	PageCell elements describe how page contents will be imaged onto individual page cells. Only one page cell size may be specified and is applied to all cells on both **Surface**s of a **Sheet**.

Structure of the PageCell Subelement

PageCell elements describe how page contents will be imaged onto individual page cells. Only one page cell size may be specified and is applied to all cells on both **Surface**s of a **Sheet**.

Name	Data Type	Description
Border ? **Modified in JDF 1.1A**	number	A number indicating the width in points of a drawn border line, that appears around the trim region specified by the explicit or implied value of *TrimSize.* A value of 0 specifies no border. If this attribute is not present, its default value is 0. If the value of this attribute is non zero and positive, then a border of that specified width will be drawn to the outside of the page cell whose inside dimension is the same as the explicit or implied value of the *TrimSize* attribute. The border marks must not overwrite the page contents of the trimmed page. Note that when the page cells are distributed

Name	Data Type	Description
		evenly over the area of the *SurfaceContentsBox*, the page cells position and/or size may be adjusted to accommodate the border.
		If the value of this attribute is non zero and negative, then a border of a width specified by this attribute's absolute value will be drawn to the inside of the page cell whose outside dimension is the same as the explicit or implied value of the *TrimSize* attribute. The border marks may overwrite the page contents of the trimmed page.
		The rectangle defined by the inside edge of the border defines a ClipBox beyond which no content will be imaged.
ClipBox ?	rectangle	Defines a rectangle with an origin relative to the lower left corner of the page cell rectangle defined by the explicit or implied value of the *TrimSize* attribute. Page content data imaged outside of the region defined by this rectangle will be clipped. If ClipBox is larger than *TrimSize*, it is used to specify a bleed region. If not specified, its default value is "0 0 X Y" where X and Y are the explicit or implied values of *TrimSize*.
MarkList ?	NMTOKENS	List of Marks that should be marked on each PageCell. The appearance of the marks are defined by the process implementation. Values include: *CIELABMeasuringField* *ColorControlStrip* *ColorRegisterMark* *CutMark* *DensityMeasuringField* *IdentificationField* *JobField* *PaperPathRegisterMark* *RegisterMark* *ScavengerArea*
Rotate ?	enumeration	Orthogonal rotation to be applied to the contents in the PageCells. One of*:* *Rotate0* *Rotate90* – 90° counterclockwise rotation. – *Rotate180* – 180° rotation *Rotate270* – 90°clockwise rotation For details of orthogonal rotation, refer to Table 2-3. If a *RotatePolicy* value other than "*NoRotate"* is specified in **FitPolicy**, the value specified in *Rotate* may be modified accordingly. Default = " *Rotate0*"
TrimSize ? Modified in JDF 1.1A	XYPair	Defines the dimensions of the PageCell. The lower left corner of the rectangle specified by the value of this attribute establishes the coordinate system into which the page content is mapped. FitPolicy defines the default *TrimSize* in the absence of an explicit *TrimSize*. If not specified, *TrimSize* is calculated by subtracting the gutters from the **LayoutPreparationParams**:*SurfaceContentsBox* and dividing by the appropriate *NumberUp* value.

Name	Data Type	Description
Color ?	refelement	Color of the border. If not present, the default color is system specified.
DeviceMark ?	refelement	Details how device dependent marks should be generated. Defaults to the value of DeviceMark in the parent **LayoutPreparationParams**.
FitPolicy ?	refelement	Details how page content is fit into the PageCells. If the dimensions of the page contents vary, FitPolicy is applied to the contents of each cell individually.
ImageShift ?	element	Element which describes how content should be placed into the PageCells. X and Y are specified in the coordinate system of the PageCell.

Structure of the ImageShift Subelement

ImageShift elements describe how the grid of page cells will be imaged onto media, when ImageShift is specified in the context of **LayoutPreparationParams**. When ImageShift is specified in the context of a PageCell, it specifies how content is imaged into the respective page cells.

Name	Data Type	Description
PositionX ?	enumeration	Indicates how images should be positioned horizontally . *ShiftBack* and *ShiftFront* are applied after *PositionX* and *PositionY*. Values are: *Center* – Center the images horizontally without regard to limitations of the printable area. *Left* – Position the left edge of the images so they are coincident with the left edge of the printable area. *Right* – Position the right edge of the images so they are coincident with the right edge of the printable area. *None* – Place the images wherever the print data specifies. The default.
PositionY ?	enumeration	Indicates how images should be positioned vertically. *ShiftBack* and *ShiftFront* are applied after *PositionX* and *PositionY*. Values are: *Bottom* – Position the bottom edge of the images so they are coincident with the bottom edge of the printable area. *Center* – Center the images horizontally without regard to limitations of the printable area. *Top* – Position the top edge of the images so they are coincident with the top edge of the printable area. *None* – Place the images wherever the print data specifies. The default.
ShiftBack ?	XYPair	The amount in X and Y direction by which the image is to be shifted on the back side.
ShiftFront ?	XYPair	The amount in X and Y direction by which the image is to be shifted on the front side. Default = "0 0"

7.2.86 LongitudinalRibbonOperationParams

Deprecated in JDF 1.1.

This resource provides the parameters of the *LongitudinalRibbonOperation* process. It is defined as a list of abstract *LROperation* elements.

Resource Properties

Resource class: Parameter
Resource referenced by: -

Example Partition: *RibbonName, SheetName, SignatureName, WebName*
Input of processes: *LongitudinalRibbonOperations*
Output of processes: -

Resource Structure

Name	Data Type	Description
LROperation +	element	Abstract element which is a placeholder for a longitudinal ribbon operation.

Structure of LongitudinalRibbonOperationParams Elements

LROperation

Deprecated in JDF 1.1.

LROperation is an abstract element that describes the ***LongitudinalRibbonOperation*** process. The defined instances (subclasses) of LROperation are LongFold, LongGlue, LongPerforate, and LongSlit. All instances of LROperation have the following common contents.

Name	Data Type	Description
WorkingList ?	NumberList	List of lengths of the *Operation* to be performed in point. Entries with an odd position (first, third, etc.) in the list define an offset where the tool is inactive. Entries with an even position define a working length where the tool is on. The start position is the leading edge of the plate. If the sum of all entries is higher than the circumference of the press cylinder, the values exceeding the circumference are cropped. Counting always restarts at the leading edge. Default = 0 1000000000, i.e., always on.
XOffset	double	Position of the tool for longitudinal action along the cylinder axis.

LongFold

Deprecated in JDF 1.1.

LongFold is derived from the abstract element LROperation and describes a longitudinal fold operation and has no further contents in addition to those of LROperation.

LongGlue

Deprecated in JDF 1.1.

LongGlue is derived from the abstract element LROperation and describes a longitudinal gluing operation and has the following contents in addition to those of LROperation.

Name	Data Type	Description
GlueBrand ?	string	Glue brand. Use only when *Operation* = Glue
GlueType ?	Enumeration	If *Operation* = Glue, the following values can be used: *ColdGlue* *Hotmelt* *PUR* – Polyurethane
LineWidth ?	double	Width of the *Operation* line.
MeltingTemperature ?	integer	Required temperature for melting the glue (in degrees centigrade). Use only when *GlueType* = *Hotmelt* and *Operation* = Glue

LongPerforate
Deprecated in JDF 1.1.

LongPerforate is derived from the abstract element LROperation and describes a longitudinal gluing operation and has the following contents in addition to those of LROperation.

Name	Data Type	Description
TeethPerDimension ?	integer	If *Operation* = Perforate, the number of teeth in a given perforation extent is defined in teeth/point. *MicroPerforation* is defined by specifying a large number of teeth (n>1000).

LongSlit
Deprecated in JDF 1.1.

LongSlit is derived from the abstract element LROperation and describes a longitudinal cut operation and has no further contents in addition to those of LROperation.

7.2.87 ManualLaborParams
New in JDF 1.1

This resource describes the parameters to qualify generic manual work within graphic arts production. Additional Comment elements will generally be needed to describe the work in human readable form.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ManualLabor*
Output of processes: -

Resource Structure

Name	Data Type	Description
LaborType	NMTOKENS	List of types of manual labor that are performed.

7.2.88 Media

This resource describes a physical element that represents a raw, unexposed printable surface such as sheet, film, or plate.

Resource Properties
Resource class: Consumable
Resource referenced by: **ExposedMedia, DigitalPrintingParams, InsertSheet, LayoutElementProduction, LayoutPreparationParams, RenderingParams, Sheet, Tile,**
Example Partition: *SheetName, Side, TileID, WebName*
Input of processes: *ConventionalPrinting, ContactCopying, Cutting*, *DigitalPrinting*, *ImageSetting*, *Proofing*,
Output of processes: -

Resource Structure

Name	Data Type	Description
BackCoatings ?	enumeration	Identical to *FrontCoatings*, but applied to the back surface of the media. Default = value of *FrontCoatings*.
Brightness ?	double	Reflectance percentage.
ColorName ? New in JDF 1.1	string	Link to a definition of the color specifics. The value of *ColorName* color should match the *Name* attribute of a Color defined in a **ColorPool** resource that is linked to the process using this **Media** resource.

Name	Data Type	Description
Dimension ? Modified in JDF 1.1	XYPair	The X and Y dimensions of the chosen medium. Measured in points. The X,Y values of *Dimension* establishes the user coordinate system into which content is mapped, i.e., the origin is in the lower left corner of the rectangle defined by 0 0 X Y. In case of *Roll* media, the X-coordinate specifies the reel width and the Y-coordinate specifies the length of the web in points. If a *Dimension* coordinate is unknown, the value must be zero. Default = 0 0, i.e., unknown. If either or both X or Y is 0, i.e., unknown, the default orientation is assumed to be portrait, i.e., Y>X.
FrontCoatings ?	enumeration	What preprocess coating has been applied to the front surface of the media. Possible values are: *None* – The default. *Glossy* *HighGloss* *Matte* *Satin* *Semigloss*
Grade ? Modified in JDF1.1A	integer	The intended *Grade* of the media on a scale of 1 through 5. *Grade* is ignored if *MediaType* is not "*Paper*". *Grade* of paper material is defined in accordance with ISO 12647-2 ff. Offset printing grades are defined as the following integer values: 1: Gloss-coated paper 2: Matt-coated paper 3: Gloss-coated, web paper 4: Uncoated, white paper 5: Uncoated, yellowish paper
GrainDirection ? New in JDF 1.1	enumeration	Direction of the grain in the coordinate system defined by *Dimension*. Possible values are: *ShortEdge*: Along the shorter axis as defined by *Dimension.* *LongEdge*: Along the longer axis as defined by *Dimension.* If not specified the direction is unknown.
HoleCount ? Deprecated in JDF 1.1	integer	The number of holes that should be punched in the media (either pre- or post-punched). Default = 0. In JDF/1.1, use *HoleType* Hole or HoleLine, which includes the number of holes.
HoleType ? New in JDF 1.1	enumerations	Predefined hole pattern. Multiple hole patterns are allowed, e.g, 3-hole ring binding and 4-hole ring binding holes on one piece of media. For details of the hole types, refer to Appendix L JDF/CIP4 Hole Pattern Catalog. Allowed values are:

Name	Data Type	Description	
		None – The default.	*R6-generic*
		R2-generic	*R6m-4h2s*
		R2m-DIN	*R6m-DIN-A5*
		R2m-ISO	*R7-generic*
		R2i-US-a	*R7i-US-a*
		R2i-US-b	*R7i-US-b*
		R3-generic	*R7i-US-c*
		R3i-US	*R11m-7h4s*
		R4-generic	*P12m-rect-0t*
		R4m-DIN-A4	*P16_9i-rect-0t*
		R4m-DIN-A5	*W2_1i-round-0t*
		R4m-swedish	*W2_1i-square-0t*
		R4i-US	*W3_1i-square-0t*
		R5-generic	*C9.5m-round-0t*
		R5i-US-a	*Explicit* – Holes are defined in an array of Hole or HoleLine
		R5i-US-b	
		R5i-US-c	
ImagableSide ?	enumeration	Side of the chosen medium that may be marked. Possible values are: *Front* *Back* *Both* – Default value. *Neither*	
MediaColorName ? Modified in JDF 1.1	NamedColor	A name for the color. Allowed values are defined in Appendix A.2.8 NamedColor.	
MediaSetCount ?	integer	When the input media is grouped in sets, identifies the number of pieces of media in each set. For example, if the *MediaTypeDetails* is "*PreCutTabs*", a *MediaSetCount* of 5 would indicate that each set includes 5 tab sheets.	
MediaType ?	enumeration	Describes the medium being employed. Possible values are: *EmbossingFoil* *Film* *Foil* *LaminatingFoil* *Paper* *Plate* *ShrinkFoil* *Transparency* *Unknown:* the default.	
MediaTypeDetails ?	NMTOKEN	Additional details of the chosen medium. If *MediaTypeDetails*is specified, *MediaType* must be specified with a value other than "*Unknown*". Possible values include: *Aluminum* – Conventional press plate *Cardboard* *DryFilm*	

Name	Data Type	Description
		Continuous – Continuously connected sheets of an opaque material. Which edge is connected is not specified.
		ContinuousLong – Continuously connected sheets of an opaque material connected along the long edge.
		ContinuousShort – Continuously connected sheets of an opaque material connected along the short edge.
		CtPVisiblePhotoPolymer – Visible light CtP plate with photo polymer process.
		CtPVisibleSilver – Visible light CtP plate with silver halide process.
		CtPThermal: – Thermal CtP plate
		Envelope – Envelopes that can be used for conventional mailing purposes.
		EnvelopePlain -- Envelopes that are not preprinted and have no windows.
		EnvelopeWindow -- Envelopes that have windows for addressing purposes.
		FullCutTabs – Media with a tab that runs the full length of the medium so that only one tab is visible extending out beyond the edge of non-tabbed media.
		ImageSetterPaper – Contact paper as replacement for film.
		Labels – Label stock, e.g., a sheet of peel-off labels.
		Letterhead – Separately cut sheets of an opaque material including a letterhead.
		MultiLayer – Form medium composed of multiple layers which are preattached to one another, e.g., for use with impact printers.
		MultiPartForm – Form medium composed of multiple layers not preattached to one another; each sheet may be drawn separately from an input source.
		Paper – Proof or product component paper
		Photographic – Separately cut sheets of an opaque material to produce photographic quality images.
		PlateUV – Press plate for the UV process
		Polyester – CtP press plate.
		PreCutTabs – Media with tabs that are cut so that more than one tab is visible extending out beyond the edge of non-tabbed media.
		Stationery – Separately cut sheets of an opaque material.
		TabStock – Media with tabs, either precut or full-cut.
		Transparency – Separately cut sheets of a transparent material.
		WetFilm – Conventional photographic film
		Unknown – The default.
MediaUnit ?	enumeration	Describes the format of the media as it is delivered to the device. Possible values are:
		Roll
		Sheet – Default value.

Name	Data Type	Description
Opacity ?	enumeration	The opacity of the media. Possible values are: *Opaque* – The media is opaque. The default. *Transparent* – The media is transparent
Polarity ?	enumeration	Polarity of the chosen medium. Possible values are: *Positive* – Default value. *Negative*
PrePrinted ?	boolean	Indicates whether the media is preprinted. Default = *false*
Recycled ?	boolean	If *true*, recycled media is requested. Default = *false*
RollDiameter ?	double	Specifies diameter of a roll in points.
ShrinkIndex ? New in JDF 1.1	XYPair	Specifies the ratio of the media linear dimension after shrinking to prior shrinking. The X-Value specifies index in the major shrink axis, whereas the Y-Value specifies the index in the minor shrink axis. Used to describe shrink wrap media. Default = 1.0 1.0, i.e., no shrinking.
StockType ? New in JDF 1.1	NMTOKEN	Strings describing the available stock. Examples include: *Bristol* *Cover* *Bond* *Newsprint* *Index* *Offset* – This includes book stock. *Tag* *Text*
Texture ? New in JDF 1.1	NMTOKEN	The intended texture of the media. Examples include: *Antique* – Rougher than vellum surface. *Calendared* – Extra-smooth or polished uncoated paper. *Linen* – Texture of coarse woven cloth. *Smooth* *Stipple* – Fine pebble finish. *Vellum* – Slightly rough surface .
Thickness ?	double	The thickness of the chosen medium. Measured in micron [µm].
UserMediaType ? Deprecated in JDF 1.1	NMTOKEN	A human-readable description of the type of media. The value can be used by an operator to select the correct media to load. The semantics of the values will be site-specific. *UserMediaType* has been merged into *MediaTypeDetails* in JDF 1.1. Possible values include: *Continuous* – Continuously connected sheets of an opaque material. Which edge is connected is not specified. *ContinuousLong* – Continuously connected sheets of an opaque material connected along the long edge. *ContinuousShort* – Continuously connected sheets of an opaque material connected along the short edge. *Envelope* – Envelopes that can be used for conventional mailing purposes. *EnvelopePlain* – Envelopes that are not preprinted and have no win-

Name	Data Type	Description
		dows.
		EnvelopeWindow – Envelopes that have windows for addressing purposes.
		FullCutTabs – Media with a tab that runs the full length of the medium so that only one tab is visible extending out beyond the edge of non-tabbed media.
		Labels – Label stock, e.g., a sheet of peel-off labels.
		Letterhead – Separately cut sheets of an opaque material including a letterhead.
		MultiLayer – Form medium composed of multiple layers which are preattached to one another, e.g., for use with impact printers.
		MultiPartForm – Form medium composed of multiple layers not preattached to one another; each sheet may be drawn separately from an input source.
		Photographic – Separately cut sheets of an opaque material to produce photographic quality images.
		PreCutTabs – Media with tabs that are cut so that more than one tab is visible extending out beyond the edge of non-tabbed media.
		Stationery – Separately cut sheets of an opaque material.
		TabStock – Media with tabs, either precut or full-cut.
		Transparency – Separately cut sheets of a transparent material.
Weight ?	double	Weight of the chosen medium. Measured in grams per square meter [g/m²].
Color ? Deprecated in JDF 1.1	refelement	A **Color** resource that provides the color of the chosen medium.

7.2.89 MediaSource

Deprecated in JDF 1.1

This resource describes the source and physical orientation of the media to be used in DigitalPrinting or IDPrinting.

Resource Properties

Resource class: Parameter
Resource referenced by: DigitalPrintingParams, IDPrintingParams, InsertSheet, Layout, Sheet, Tile
Example Partition: -
Input of processes: *DigitalPrinting*, *IDPrinting*
Output of processes: -

Resource Structure

Name	Data Type	Description
LeadingEdge ?	number	Specifies the size, in points, of the edge of the media that represents the scanline direction. If this attribute is absent, the scanline direction is assumed to be along the x-axis of the *Dimension* parameter for the Media.
MediaLocation ?	String	Identifies the location, such as a slot name or ID, of the media in the device. If the media resource is partitioned by **Location** (see also Section 3.9.2.2 Locations of Physical Resources) there should be a match between one **Location** partition key and this **MediaLocation** value.

Name	Data Type	Description
ManualFeed ?	boolean	Indicates whether the media will be fed manually. Default = *false*
SheetLay ? New in JDF 1.1	enumeration	Lay of input media. Reference edge of where paper is placed in feeder. Possible values are: *Left* *Right* *Center* *Default* = The device-specific machine default. *SystemSpecified* = The device-specific machine default Default = *SystemSpecified*
Component ? New in JDF 1.1	refelement	A **Component** resource which identifies the preprinted media to be used. Only one of **Component** or **Media** should be specified.
Media ?	refelement	A **Media** resource which identifies the media to be used. Only one of **Component** or **Media** should be specified.

7.2.90 NumberingParams

This resource describes the describes the parameters of stamping or applying variable marks in order to produce unique components, for items such as lottery notes or currency. One NumberingParams element must be defined per numbering machine.

Resource Properties
Resource class:　　　Parameter
Resource referenced by: -
Example Partition:　　-
Input of processes:　　*Numbering*
Output of processes:　 -

Resource Structure

Name	Data Type	Description
NumberingParam *	element	Set of parameters for one numbering machine

Structure of NumberingParam Subelement

Name	Data Type	Description
StartValue ?	string	First value of the numbering machine.
XPosition	number	Position of the numbering machine along the printer axis.
YPosition	NumberList	List of stamp positions, in points, starting from the leading edge.
Orientation	number	Rotation of the numbering machine in degrees. If *Orientation* = 0, the top of the numbers is along the leading edge.
Step ?	integer	Number that specifies the difference between two subsequent numbers of the numbering machine. Default = 1

7.2.91 ObjectResolution

ObjectResolution defines a resolution depending on *SourceObject* data types.

Resource Properties
Resource class:　　　Parameter
Resource referenced by: **InterpretingParams, RenderingParams, TrappingDetails**
Example Partition:　　-
Input of processes:　　-
Output of processes:　 -

Resource Structure

Name	Data Type	Description
Resolution	XYPair	Horizontal and vertical output resolution in DPI.
SourceObjects ?	enumerations	Identifies the class(es) of incoming graphical objects to render at the specified resolution. Possible values are: *All* – Default value. *ImagePhotographic* – Contone images. *ImageScreenShot* – Images largely comprised of rasterized vector art. *LineArt* *SmoothShades* – Gradients and blends. *Text*

7.2.92 OrderingParams

Attributes of the ***Ordering*** process, which results in an acquisition.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Ordering*
Output of processes: -

Resource Structure

Name	Data Type	Description
Amount	double	Amount of the ordered resource.
Unit	string	Unit of measurement for *Amount.*
Comment	telem	**OrderingParams** require a Comment element that contains a human-readable description of what to order.
Company ? Deprecated in JDF 1.1	refelement	Address and further information of the **Company** responsible for this order. Replaced with **Contact** in JDF 1.1.
Contact * New in JDF 1.1	refelement	Address and further information of the **Contact** responsible for this order.

7.2.93 PackingParams

Deprecated in JDF 1.1

The PackingParams resource has been deprecated in version 1.1 and beyond. It is replaced by the individual resources used by the processes defined in Section 6.5.45.4 Numbering and 6.5.45.5 Packaging Processes.

This resource specifies the box packing parameters for a JDF job, using information that identifies the type of package, the wrapping used, and the shape of the package. Note that this specifies packing for shipping only, not packing of items into custom boxes etc. Boxes are convenience packaging, and are not envisioned to be protection for shipping. Cartons perform this function. All quantities are specified as finished pieces per wrapped/boxed/carton or palletized package.

The model for packaging is that products are *wrapped* together, wrapped packages are placed in *boxes,* boxes are placed in *cartons*, and cartons are stacked on *pallets*.

Resource Properties

Resource class: Parameter
Resource referenced by: -

Example Partition: -
Input of processes: *Packing*
Output of processes: -

Resource Structure

Name	Data Type	Description
BoxedQuantity ?	integer	How many units of *product* in a box.
BoxShape ?	shape	Describes the length, width and height of the box in points.
CartonQuantity ?	integer	How many units of *product* in a carton.
CartonShape ?	shape	Describes the length, width and height of the carton in points, e.g., 288 544 1012.
CartonMaxWeight ?	double	Maximum weight of an individual carton in kilograms.
CartonStrength ?	double	Strength of the carton in Newtons per square meter.
PalletQuantity ?	integer	Number of *product* per pallet
PalletSize ?	XYPair	Describes the length and width of the pallet in points, e.g., 3500 3500
PalletMaxHeight ?	double	Maximum height of a loaded pallet in points.
PalletMaxWeight ?	double	Maximum weight of a loaded pallet in kilograms.
PalletType ?	enumeration	Type of pallet used. Examples include: *2Way* – Two-way entry *4Way* – Four-way entry *Euro* – Standard 1*1 m Euro pallet
PalletWrapping ?	enumeration	Wrapping of the completed pallet. Examples include: *StretchWrap* *Banding* *None* – The default.
WrappedQuantity ?	integer	Number of units of *product* per wrapped package.
WrappingMaterial ?	name	Examples include: *RubberBand* *ShrinkWrap* *PaperBand* *Polyethylene* *None* – The default.

7.2.94 PalletizingParams

New in JDF 1.1

PalletizingParams defines the details of *Palletizing*. Details of the actual palette used for *Palletizing* can be found in the **Pallet** resource that is also an input of the *Palletizing* process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Palletizing*
Output of processes: -

Resource Structure

Name	Data Type	Description
Pattern ?	string	Name of the palletizing pattern. Used to store a predefined pattern that defines the layers and positioning of individual component on the palette. Default = equipment-specific pattern.
MaxHeight ?	number	Maximum height of a loaded pallet in points. Default = equipment-specific value.
MaxWeight ?	number	Maximum weight of a loaded pallet in grams.

7.2.95 Pallet
New in JDF 1.1
A Pallet represents the palette used in packing goods.

Resource Properties
Resource class: Consumable
Resource referenced by: -
Example Partition: -
Input of processes: *Palletizing*
Output of processes: -

Resource Structure

Name	Data Type	Description
PalletType	NMTOKEN	Type of pallet used. Examples include: *2Way* – Two-way entry *4Way* – Four-way entry *Euro* – Standard 1*1 m Euro pallet
Size ?	XYPair	Describes the length and width of the pallet in points, e.g., 3500 3500. Default = 0 0 which specifies the size defined by *PalletType.*

7.2.96 PDFToPSConversionParams

This resource specifies a set of configurable options that may be used by processes that generate PostScript files from PDF files. Font controls are applied in the following order:

1. IncludeBaseFonts
2. IncludeEmbeddedFonts
3. IncludeType1Fonts
4. IncludeType3Fonts
5. IncludeTrueTypeFonts
6. IncludeCIDFonts

For example, an embedded Type-1 font follows the rule for embedded fonts, not the rule for Type-1 fonts. In other words, if *IncludeEmbeddedFonts* is *true*, and *IncludeType1Fonts* is *false*, embedded Type-1 fonts would be included in the PostScript stream.

Resource Properties
Resource class: Parameter
Resources referenced: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *PDFToPSConversion*
Output of processes: -

Resource Structure

Name	Data Type	Description
BinaryOK ?	boolean	If true, binary data are to be included in the PostScript stream. Default = *true*
BoundingBox ?	rectangle	If all zeroes, this attribute is ignored. Otherwise, it is used for *BoundingBox* DSC comment, in *CenterCropBox* calculations and for *SetPageDevice*. Default = 0 0 0 0
CenterCropBox ?	boolean	If *true*, CropBox output is centered on the page when the CropBox < MediaBox. Default = *true*
GeneratePageStreams ?	boolean	If *true*, the process emits individual streams of data for each page in the **RunList**. Default = *false*
IgnoreAnnotForms ?	boolean	If *true*, ignores annotations that contain an XObject form. Default = *false*
IgnoreBG ? New in JDF 1.1	boolean	Ignores the BG,BG2 parameters in the PDF ExtGState dictionary. Default=*true*
IgnoreColorSeps ?	boolean	If *true*, ignores images for Level-1 separations. Default = *false*.
IgnoreDeviceExtGState ? Deprecated in JDF 1.1	boolean	If *true*, ignores all device-dependent extended graphic state parameters. This overrides *IgnoreHalftones*. The following parameters should be ignored: op OP – Overprint parameter OPM – Overprint mode BG, BG2 – Black generation UCR, UCR2 – Undercolor removal TR, TR2 – Transfer functions HT – Halftone dictionary FL – Flatness tolerance SA – Automatic stroke adjustment Default = *true*
IgnoreDSC ?	boolean	If *true*, ignores DSC (Document Structuring Conventions). Default = *true*
IgnoreExternSreamRef ?	boolean	If an image resource uses an external stream and *IgnoreExternStreamRef* = *true*, ignores code that points to the external file. Default = *false*
IgnoreHalftones ?	boolean	If *true*, ignores any halftone screening in the PDF file. Default = *false*
IgnoreOverprint ? New in JDF 1.1	boolean	Ignores the OP, op parameters in the PDF ExtGState dictionary. Default=*true*
IgnorePageRotation ?	boolean	If *true*, ignores a concat provided at the beginning of each page that orients the page so that it is properly rotated. Used when emitting EPS. Default = *false*
IgnoreRawData ?	boolean	If *true*, no unnecessary filters should be added when emitting image data. Default = *false*
IgnoreSeparableImages-Only ?	boolean	If *true*, and if emitting EPS, ignores only CMYK and gray images. Default = *false*

Name	Data Type	Description
IgnoreShowPage ?	boolean	If *true*, ignores save-and-restore showpage in PostScript files. Default = *false*
IgnoreTransfers ? New in JDF 1.1	boolean	Ignores the TR,TR2 parameters in the PDF ExtGState dictionary. Default = *true*
IgnoreTTFontsFirst ?	boolean	If *true*, ignores TrueType fonts before any other fonts. Default = *false*
IgnoreUCR ? New in JDF 1.1	boolean	Ignores the UCR, UCR2 parameters in the PDF ExtGState dictionary. Default=true
IncludeBaseFonts ?	enumeration	Determines when to embed the base fonts. Possible values are: *IncludeNever* – Default value *IncludeOncePerDoc* *IncludeOncePerPage*
IncludeCIDFonts ?	enumeration	Determines when to embed CID fonts. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeEmbeddedFonts ?	enumeration	Determines when to embed fonts in the document that are embedded in the PDF file. This attribute overrides the *IncludeType1Fonts*, *IncludeTrueTypeFonts*, and *IncludeCIDFonts* attributes. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeOtherResources ?	enumeration	Determines when to include all other types of resources in the file. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeProcSets ?	enumeration	Determines when to include ProcSets in the file. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeTrueTypeFonts ?	enumeration	Determines when to embed TrueType fonts. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeType1Fonts ?	enumeration	Determines when to embed Type-1 fonts. Possible values are: *IncludeNever* *IncludeOncePerDoc* – Default value. *IncludeOncePerPage*
IncludeType3Fonts ?	enumeration	Determines when to embed Type-3 fonts. Must always be set to *IncludeOncePerPage*. It is included here to complete the precedence hierarchy.

Name	Data Type	Description
OutputType	enumeration	Describes the kind of output to be generated. Possible values are: *PostScript* – Default value *EPS*
PSLevel ?	integer	Number that indicates the PostScript level.. Default = 2
Scale ?	Number	Number that indicates the wide-scale factor of documents. Full-size = 100. Default = 100
SetPageSize ?	boolean	(PostScript Level 2 only) If *true*, sets page size on each page automatically. Use media box for outputting PostScript files and crop box for EPS. Default = *false*.
SetupProcsets ?	boolean	If *true*, indicates that if procsets are included, the init/term code is also included. Default = *true*
ShrinkToFit ?	boolean	If *true*, the page is scaled to fit the printer page size. This field overrides scale. Default = *false*
SuppressCenter ?	boolean	If *true*, suppresses automatic centering of page contents whose crop box is smaller than the page size. Default = *false*
SuppressRotate ?	boolean	If *true*, suppresses automatic rotation of pages when their dimensions are better suited to landscape orientation. More specifically, the application that generates the PostScript compares the dimensions of the page. If the width is greater than the height, then pages are not rotated if **SupressRotate** is *true*. On the other hand, if **SupressRotate** is *false*, the value of the PDF Rotate key for each page is honored, regardless of the dimensions of the pages (as defined by the **MediaBox** attribute). Default = *false*
TTasT42 ?	boolean	If including TrueType fonts, converts to Type-42 instead of Type-1 fonts when **TTasT42** = *true*. Default = *false*
UseFontAliasNames ?	boolean	If *true*, font alias names are used when printing with system fonts. Default = *false*

7.2.97 PDLResourceAlias

This resource provides a mechanism for referencing resources that occur in files, or that are expected to be provided by devices. Prepress and printing processes have traditionally used the word "resource" to refer to reusable data structures that are needed to perform processes. Examples of such resources include fonts, halftones, and functions. The formats of these resources are defined within PDLs, and instances of these resources may occur within PDL files, or may be provided by devices.

JDF does not provide a syntax for defining such resources directly within a job. Instead, resources continue to occur within PDL files and continue to be provided by devices. However, since it is necessary to be able to refer to these resources from JDF jobs, the **PDLResourceAlias** resource is provided to fulfill this need.

Resource Properties
Resource class: Parameter
Resource referenced by: ColorantControl
Example Partition: -
Input of processes: *Interpreting*
Output of processes: -

Resource Structure

Name	Data Type	Description
ResourceType	string	The type of PDL resource that is referenced. The semantic of this attribute is defined by the PDL.
SourceName ?	string	The name of the resource in the file referenced by the FileSpec element or by the device.
FileSpec ?	refelement	Location of the file containing the PDL resource. If FileSpec is absent, the device is expected to provide the resource defined by this **PDLResourceAlias** resource.

7.2.98 PerforatingParams

New in JDF 1.1

PerforatingParams define the parameters for perforating a sheet .

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Perforating*
Output of processes: -

Resource Structure

Name	Data Type	Description
Perforate *	element	definition of one or more Perforate lines.

Structure of the Perforate element

Perforate describes one perforated line.

Name	Data Type	Description
StartPosition	XYPair	Starting position of the tool.
WorkingPath	XYPair	Relative working path of the tool. Since the tools can only work parallel to the edges, one coordinate must be zero.
WorkingDirection	enumeration	Direction from which the tool is working. Possible values are: *Top* – From above *Bottom* – From below
TeethPerDimension ?	number	Number of teeth in a given perforation extent in teeth/point. *MicroPerforation* is defined by specifying a large number of teeth (n>1000).

7.2.99 Person

This resource provides detailed information about a person. It also has the ability to specify different communication channels to this person. The structure of the resource is derived from the vCard format. It contains all of the same name subtypes (N:) of the identification and the title of the organizational properties. The corresponding XML types of the vCard are quoted in the description field of the table below.

Resource Properties

Resource class: Parameter
Resource referenced by: Contact, Employee
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
AdditionalNames ?	string	Additional names of the contact person (vCard: N:other).
FamilyName ?	string	The family name of the contact person (vCard: N:family).
FirstName ?	string	The first name of the contact person (vCard: N:given).
JobTitle ?	string	Job function of the person in the company or organization (vCard: title).
NamePrefix ?	string	Prefix of the name, may include title (vCard: N:prefix).
NameSuffix ?	string	Suffix of the name (vCard: N:suffix).
ComChannel *	element	Communication channels to the person.

7.2.100   PlaceHolderResource

This resource is used to link *ProcessGroup* nodes when the exact nature of interchange resources is still unknown. In this way, a skeleton of process networks can be constructed, with the **PlaceHolderResource** resources serving as place holders in lieu of the appropriate resources. This resource needs no structure besides that provided in an abstract Resource element, as it has no inherent value except as a stand-in for other resources.

Resource Properties

Resource class: PlaceHolder
Resource referenced by: -
Example Partition: -
Input of processes: any *ProcessGroup* nodes
Output of processes: any *ProcessGroup* nodes

Resource Structure

The resource has no additional structure.

7.2.101   PlasticCombBindingParams

This resource describes the details of the ***PlasticCombBinding*** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *PlasticCombBinding*
Output of processes: -

Resource Structure

Name	Data Type	Description
Brand ?	string	The name of the comb manufacturer and the name of the specific item. Default =system specified.
Color ?	NamedColor	Determines the color of the plastic comb. Default =system specified.
Diameter ?	double	The comb diameter is determined by the height of the block of sheets to be bound. Default =system specified.
Thickness ?	double	The material thickness of the comb. Default =system specified.

Name	Data Type	Description
Type ? Modified in JDF 1.1	enumeration	The distance between the "teeth" and the distance between the holes of the prepunched sheets must be the same. The following values from the hole type catalog in Appendix L exist: *P12m-rect-02*: Distance = 12 mm; Holes = 7 mm x 3 mm *P16_9i-rect-0t*: Distance = 14.28 mm; Holes = 8 mm x 3 mm The following values are deprecated in JDF 1.1. *Euro* (Distance = 12 mm; Holes = 7 mm x 3 mm) *USA1* (Distance = 14.28 mm; Holes = 8 mm x 3 mm)

7.2.102   PlateCopyParams

Deprecated in JDF 1.1

This resource specifies the parameters of the **FilmToPlateCopying** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *FilmToPlateCopying*
Output of processes: -

Resource Structure

Name	Data Type	Description
Cycle ?	integer	Number of exposure light units to be used. The amount depends on the subject to be exposed.
Diffusion ?	enumeration	The diffusion foil setting. Possible values are: *On* *Off*
Vacuum ?	double	Amount of vacuum pressure to be used. Measured in bars.

7.2.103   PreflightAnalysis

Note that the resources for Preflight are under development and subject to major changes in a future release of this specification.

PreflightAnalysis resources record the results of a **Preflight** process. The semantics for results are specific to the FileType of the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value for each of these elements is an array of PreflightResultsDetail and PreflightInstance subelements. Within the PreflightInstance subelements, results are further broken down into PreflightInstanceDetails.

Each PreflightResultsDetail and PreflightInstance subelement in the **PreflightAnalysis** hierarchy describes the results of a comparison of the properties of the file against one PreflightConstraint in the **PreflightProfile**.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: -
Output of processes: *Preflight*

Resource Structure

Name	Data Type	Description
ColorsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about color.
DocumentResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about documents.
FontsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about fonts.
FileTypeResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about file types.
ImagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about images.
PagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides analysis about pages.

Structure of PreflightDetail Subelement

PreflightDetail subelements are used to describe one property within the **PreflightAnalysis** category in which they occur. This subelement is also used by **PreflightInventory** resource.

Name	Data Type	Description
PageRefs	IntegerRangeList	Identifies the set of pages in a **RunList** resource that exhibit the characteristic identified by the combination of the *Property* attribute and the Value element.
Property ?	string	Identifies the property described by this element.
Status ?	enumeration	Possible values are: *Error* – Value violates the ConstraintValue specified in the associated Preflight Constraint element. The constraint was flagged as an Error in the profile. *Warning* – Value violates the ConstraintValue specified in the associated PreflightConstraint element. The constraint was flagged as a Warning in the profile. *Ignore* – The constraint is ignored, and no PreflightDetail or PreflightInstanceDetail elements are created for this constraint. *IgnoreValue* – No comparison was made against a ConstraintValue. In other words, either the *Status* for the PreflightConstraint was *Ignore* or *IgnoreValue*, or this PreflightDetail is part of a **PreflightInventory** hierarchy.
Value ?	element	Identifies the value of the property. The semantics are PDL-specific.

Structure of PreflightInstance Subelement

PreflightInstance subelements are used to collect PreflightInstanceDetail elements for one instance of some object which occurs in the PDL files referenced by a run list. For example, there might be one PreflightInstance element for each font that occurs in the pages of a run list. This subelement is also used by **PreflightInventory** resources.

Name	Data Type	Description
Identifier ?	string	Identifies the instance this element collects PreflightInstanceDetail elements.
PageRefs Modified in JDF 1.1	Integer-RangeList	Identifies the set of pages in a **RunList** on which the instance occurs.
PreflightInstanceDetail *	element	A pool of PreflightInstanceDetail elements that describe the properties for this instance

Name	Data Type	Description
Modified in JDF 1.1		

Structure of PreflightInstanceDetail Subelement

PreflightInstanceDetail subelements describe one property of one instance of some object type that occurs in a PDL file. For example, several PreflightInstanceDetail elements might describe the properties of a single font. This subelement is also used by **PreflightInventory** resources.

Name	Data Type	Description
Status ?	enumeration	Specifies the results of the comparison between the value of the property for this instance with the ConstraintValue for the associated PreflightConstraint element. Possible values are: *Error* – Value violates the ConstraintValue specified. The constraint was flagged as an Error in the profile. *Warning* – Value violates the ConstraintValue specified. The constraint was flagged as a Warning in the profile. *IgnoreValue* – No comparison was made against a Constraint-Value. In other words, either the *Status* for the Constraint was *Ignore* or *IgnoreValue*, or this PreflightInstanceDetail is part of a **PreflightInventory** hierarchy.
Property ?	string	Identifies the property described by this element.
Value ?	element	Identifies the value of the property. The semantics are PDL-specific.

7.2.104 PreflightInventory

Note that the resources for Preflight are under development and subject to major changes in a future release of this specification.

PreflightInventory resources, like **PreflightAnalysis** resources, record the results of a *Preflight* process. The semantics for results are specific to the FileType of the for the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value of each of these elements is an array of PreflightResultsDetail and PreflightInstance subelements. Within the PreflightInstance subelements, results are further broken down into PreflightInstanceDetails.

Each PreflightResultsDetail or PreflightInstance subelement in the **PreflightInventory** hierarchy describes the results of a comparison of the properties of the file against one PreflightConstraint in the **PreflightProfile**.

Resource Properties
Resource class:　　　　　Parameter
Resource referenced by: -
Example Partition:　　　　-
Input of processes:　　　*Preflight*
Output of processes:　　*Preflight*

Resource Structure

Name	Data Type	Description
ColorsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides a color inventory.
DocumentResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides a document inventory.
FontsResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that

Name	Data Type	Description
		provides a font inventory.
FileTypeResultsPool ?	element	A PreflightDetail and PreflightInstance subelement that provides a file-type inventory.
ImagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides an image inventory.
PagesResultsPool ?	element	A pool of PreflightDetail and PreflightInstance subelements that provides a page inventory.

7.2.105   PreflightProfile

Note that the resources for Preflight are under development and subject to major changes in a future release of this specification.

PreflightProfile resources specify a set of constraints against which a file may be tested. The semantics for constraints are specific to the FileType of the for the file. The elements in this resource, detailed in the table below, place the results in specific categories. The value for each of these elements is an array of PreflightConstraint subelements. Within the PreflightConstraint resources, the ConstraintValue element indicates allowable values and the *Status* attribute indicates the error level (if any) to be flagged when exceptions to the constraints are identified.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Preflight*
Output of processes: -

Resource Structure

Name	Data Type	Description
ColorsContraintsPool ?	element	A pool of PreflightConstraint subelements. Each element in this pool identifies a specific constraint concerning colors against which to test the file
DocumentConstraintsPool ?	element	A pool of PreflightConstraint subelements. Each element in this pool identifies a specific constraint concerning documents against which to test the file
FontsConstraintsPool ?	element	A pool of PreflightConstraint subelements. Each element in this pool identifies a specific constraint concerning fonts against which to test the file
FileTypeConstraintsPool ?	element	A *Preflight* constraint. The *Type* attribute must have a value of *array* and must contain string objects that identify the allowable types of data in the file. The strings in the *Value* array must be MIME-file types as recorded by the Internet Assigned Numbers Authority (IANA). IANA has procedures for registering new file types if needed.
ImagesConstraintsPool ?	element	A pool of PreflightConstraint subelements. Each element in this pool identifies a specific constraint concerning images against which to test the file
PagesConstraintsPool ?	element	A pool of PreflightConstraint subelements. Each element in this pool identifies a specific constraint concerning pages against which to test the file

Structure of PreflightConstraint Subelement

Name	Data Type	Description
AttemptFixupErrors ?	boolean	If *true*, the device performing preflight should attempt to fix errors that are identified during preflight. Errors that are corrected are not given a **Status** attribute. Default = *false*
AttemptFixupWarnings ?	boolean	If *true*, the device performing preflight should attempt to fix warnings that are identified during preflight. Warnings that are corrected are not given a **Status** attribute. Default = *false*
Constraint ?	string	Describes the specific file characteristic to be checked.
Status	enumeration	Possible values are: *Error* – Values that violate the ConstraintValue specified are flagged as Errors in PreflightDetail and PreflightInstanceDetail elements. *Warning* – Values that violate the ConstraintValue specified are flagged as Warnings in PreflightDetail and PreflightInstanceDetail elements. *Ignore* – The constraint is ignored, and no PreflightDetail or PreflightInstanceDetail elements are created for this constraint. *IgnoreValue* – No comparison is made against the ConstraintValue.
ConstraintValue ?	element	Provides a value against which to test occurrences of the characteristic in the file. Note: The semantics of the ConstraintValue element depend on the PDL characteristic in question.

7.2.106   Preview

The preview of the content of a surface. It can be used for the calculation of the ink coverage (*PreviewType* = *Separation*) or as a preview of what is currently processed in a device (*PreviewType* = *Viewable*). When the preview is of *Type* = *Separation*, a gray value of *0* represents full ink, while a value of *255* represents no ink (for more information, see DeviceGray color model chapter 4.8.2. of the *PostScript Language Reference Manual*).

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *PreviewType, Separation, SheetName, Side, TileID, WebName, RibbonName*
Input of processes: *InkZoneCalculation*
Output of processes: *PreviewGeneration*

Resource Structure

Name	Data Type	Description
Compensation ?	enumeration	Compensation of the image to reflect the application of transfer curves to the image. Possible values are: *Unknown* – Default value. *None* – No compensation. *Film* – Compensated until film exposure. *Plate* – Compensated until plate exposure. *Press* – Compensated until press.
CTM ? New in JDF 1.1	matrix	Orientation of the Preview w.r.t. the coordinate system of the device that is defined in **Compensation.** Default = identity matrix 1 0 0 1 0 0

Name	Data Type	Description
Directory ? New in JDF 1.1	URL	Defines a directory where the files that are associated with this Preview should be copied to or from. If *Directory* is not specified, the **URL** must be completely specified.
PreviewType	enumeration	Type of the preview. Possible values are: *Separation* – Separated preview in medium resolution. *SeparatedThumbNail* – Very low resolution separated preview. *ThumbNail* – Very low resolution RGB preview. *Viewable* – RGB preview in medium resolution.
URL	URL	URL identifying the PNG image file that represents this **Preview**. This is a normally a URL to a MIME subpart (see Section A.4.1). Note: A preview will generally be partitioned by separation, unless it represents an RGB viewable image or thumbnail.

7.2.107   PreviewGenerationParams

Parameters specifying the size and the type of the preview.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Example Partition:	*PreviewType, Separation, SheetName, Side, TileID, WebName, RibbonName*
Input of processes:	*PreviewGeneration*
Output of processes:	-

Resource Structure

Name	Data Type	Description
AspectRatio ? New in JDF 1.1	enumeration	Policy that defines how to define the preview size if the aspect ratio of the source and preview are different. Note that *AspectRatio* only has an effect if *Size* is specified. One of: *CenterMax* – Keep the aspect ratio and preview *Size* and center the image so that the preview has missing pixels at both sides of the larger dimension. *CenterMin* – Keep the aspect ratio and preview *Size* and center the image so that the preview has blank pixels at both sides of the smaller dimension. *Crop* – Keep the aspect ratio and modify the preview size so that the image fits into a bounding rectangle defined by *Size*. *Expand* – Keep the aspect ratio and modify the preview size so that the smaller image dimension is defined by *Size*. *Ignore* – Fill the preview completely, keeping *Size*, even if this requires modifying the aspect ratio. The default.
PreviewType ? Deprecated in JDF 1.1	enumeration	The kind of preview to be generated. Possible values are: *Separation* *Viewable*
PreviewUsage New in JDF 1.1 *?*	enumeration	The kind of preview to be generated. Possible values are: *Separation*: separated preview in medium resolution. *SeparatedThumbNail*: Very low resolution separated preview. *ThumbNail*: Very low resolution rgb preview. *Viewable:* rgb preview in medium resolution.

Name	Data Type	Description
Resolution ?	XYPair	Resolution of the preview, in DPI. Default = 50.8 50.8 dpi.
Size ?	XYPair	Size of the preview, in pixels. If this attribute is present, the *Resolution* attribute evaluated according to the policy defined in *AspectRatio*. If *Size* is not specified, it defaults to "0 0" and must be calculated using the *Resolution* attribute and the input image size.
ImageSetterParams ? New in JDF 1.1	refelement	Details of the ImageSetting process. Needed for accessing information about coordinate transformations that are performed by the image setter hardware.

7.2.108   ProofingParams

This resource specifies the settings needed for all proofing operations, including both "hard" or "soft" proofing, of color and imposition proofs.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *Proofing*, *SoftProofing*
Output of processes: -

Resource Structure

Name	Data Type	Description
ColorType ?	Enumeration	Color quality of the proof. Possible values are: *Monochrome* – Black and white. *BasicColor* – Color does not match precisely. This implies the absence of a color matching system. *MatchedColor* – Color is matched to the output of the press using a color matching system.
DisplayTraps ?	boolean	If *true*, the trap networks are shown in the proof. Default = *false*
HalfTone ?	boolean	Specifies whether the proof should emulate halftone screens. Default = *false*
ImageViewingStrategy ?	string	Identifies which images will be displayed during the **SoftProofing** process. Possible values are: *NoImages* – Default value. *OmitReference* – Displays only images actually embedded in the file. *UseProxies* – Displays images embedded in the file and proxy versions of referenced data. *UseReplacements* – Displays embedded images plus the full resolution version of referenced images.
ManualFeed ? New in JDF 1.1	boolean	Indicates whether the media will be fed manually. Default = *false*

Name	Data Type	Description
ProofRenderingIntent ? New in JDF 1.1	enumeration	Identifies the rendering intents associated with the proof. Possible ICC-defined rendering intent values are: *Saturation* *Perceptual* – The default. *RelativeColorimetric* *AbsoluteColorimetric*
ProofType ?	enumeration	Describes the type of the proof. Possible values are: *None* – Default value. Not a proof or the type is unknown. *Page* – Page proof *Imposition* – Imposition proof.
Resolution ?	XYPair	Resolution of the output.
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the proofer device. The *ResourceUsage* attribute of the FileSpec must be "*ProoferProfile*".
Media ?	refelement	Describes the media to be used.

7.2.109 PSToPDFConversionParams

This resource contains the parameters that control the conversion of PostScript streams to PDF pages.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: Preflight
Output of processes: -

Resource Structure

Name	Data Type	Description
ASCII85EncodePages ?	boolean	If *true*, binary streams such as page contents streams, sampled images, and embedded fonts are ASCII85-encoded, resulting in a PDF file that is almost pure ASCII. If *false*, they are not, resulting in a PDF file that may contain substantial amounts of binary data. Default = *false*
AutoRotatePages ?	enumeration	Allows the device to try to orient pages based on the predominant text orientation. Only used if the file does not contain "%%ViewingOrientation", "%%PageOrientation", or "%%Orientation" DSC comments. If the file does contain such DSC comments, it honors them. "%%ViewingOrientation" takes precedence over others, then "%%PageOrientation", then "%%Orientation". Possible values are: *None* – Turns *AutoRotatePages* off. *All* – Takes the predominant text orientation across all pages and rotates all pages the same way. *PageByPage* – Does the rotation on a page-by-page basis, rotating each page individually. Useful for documents that use both portrait and landscape orientations. Default = *None*
Binding ?	enumeration	Determines how the printed pages would be bound. Specify *Left* for left binding or *Right* for right binding. Default = *Left*

Name	Data Type	Description
CompressPages ?	boolean	Enables compression of pages and other content streams like forms, patterns and Type 3 fonts. If *tru*e, use Flate compression.
DefaultRenderingIntent ?	enumeration	Selects the rendering intent for the current job. Possible values are: *Default* – The default. *Perceptual* *Saturation* *RelativeColorimetric* *AbsoluteColorimetric* See the *Portable Document Format Reference Manual* for more information on rendering intent.
DetectBlend ?	boolean	Enables or disables blend detection. If *true*, and if **PDFVersion** is 1.3 or higher, then blends will be converted to smooth shadings. Default = *true*
DoThumbnails ?	boolean	If *tru*e, thumbnails are created. Default = *true*
EndPage ?	integer	Number that indicates the last page that is displayed when the PDF file is viewed. **EndPage** must equal be to anything less than **StartPage** or be greater than or equal to 1. If not, then it must be greater than or equal to **StartPage**. When combined with **Start-Page**, **EndPage** selects a range of pages to be displayed. The entire file may or may not be distilled, but only **StartPage** to **EndPage** pages, inclusive, are opened and viewed in Acrobat.
ImageMemory ?	integer	Number of bytes in the buffer used in sample processing for color, grayscale, and monochrome images. Its contents are written to disk when the buffer fills up.
InitialPageSize ? New in JDF 1.1	XYPair	Defines the initial page dimensions assumed by the PS-to-PDF converter in points. This will be overridden by any PageSize page device parameter found in the PostScript stream. The use of this attribute is strongly encouraged if the PS-to-PDF converter may be used to process Encapsulated PostScript files. Default = system specific
InitialResolution ? New in JDF 1.1	XYPair	Defines the initial horizontal and vertical resolution of the PS-to-PDF converter in DPI. This will be overridden by any HWResolution page device parameter found in the PostScript stream. The use of this attribute is strongly encouraged if the PS-to-PDF converter may be used to process Encapsulated PostScript files. Default = system specific
OverPrintMode ?	integer	Controls the overprint mode strategy of the job. Set to 0 for full overprint or 1 for non-zero overprint. For more information, see http://partners.adobe.com/asn/developer/PDFS/TN/5044.ColorSep_Conv.pdf
Optimize ?	boolean	If *tru*e, the PS-to-PDF converter optimizes the PDF file. See the *Portable Document Format Reference Manual* for more information on optimization. Default = *true*
PDFVersion ?	double	Specifies the version number of the PDF file produced. Possible values include all legal version designators, e.g., 1.2, 1.3, 1.4.

Name	Data Type	Description
StartPage ?	integer	Sets the first page that is be displayed when the PDF file is opened with Acrobat. *StartPage* must be greater than or equal to 1. If *EndPage* is not -1, then it must be greater than or equal to *Start-Page*.
AdvancedParams ?	element	Advanced parameters which control how certain features of Post-Script are handled.
ThinPDFParams ?	element	Parameters that control the optional content or form of PDF files that will be created.

Structure of AdvancedParams Subelement

Name	Data Type	Description
AutoPositionEPSInfo ? Modified in JDF1.1A	boolean	If *true*, the process automatically resizes and centers EPS information on the page. Default = *true*
EmitDSCWarnings ?	boolean	If *true*, warning messages about questionable or incorrect DSC comments appear during the distilling of the PS file. Default = *false*
LockDistillerParams ?	boolean	If *true*, the incoming PS content that specifies any of the **PSToPDFConversionParams** settings is used. If *false*, any **PSToPDFConversionParams** settings configured by the PS content are ignored. Default = *true*
ParseDSCComments ?	boolean	If *true*, the process parses the DSC comments for any information that might be helpful for converting the file or for information that must be stored in the PDF file. If *false*, the process treats the DSC comments as pure PS comments and ignores them. Default = *true*
ParseDSCComment-ForDocInfo ?	boolean	If *true*, the process parses the DSC comments in the PS file and extracts the document information. This information is recorded in the Info dictionary of the PDF file. Default = *true*
PreserveCopyPage ?	boolean	If *true*, the copypage operator of PostScript Level 2 is maintained. If *false*, the PostScript Level 3 definition of copypage operator is used. In PostScript Levels 1 and 2, the copypage operator transmits the page contents to the current output device (similar to showpage). However, copypage does not perform many of the reinitializations that showpage does. Many PostScript Level 1 and 2 programs used the copypage operator to perform such operations as printing multiple copies and implementing forms. These programs produce incorrect results when interpreted using the Level 3 copypage semantics. This attribute provides a mechanism to retain Level 2 compatibility for this operator. Default = *true*
PreserveEPSInfo ?	boolean	If *true*, preserves the EPS information in the PS file and stores it in the resulting PDF file. Default = *true*
PreserveHalftoneInfo ? New in JDF 1.1	boolean	If *true*, passes halftone screen information (frequency, angle, and spot function) into the PDF file. If *false*, halftone information is not passed in. Default = *false*
PreserveOverprint-Settings ? New in JDF 1.1	boolean	If *true*, Distiller passes the value of the setoverprint operator through to the PDF file. Otherwise, overprint is ignored. Default = *true*

Name	Data Type	Description
PreserveOPIComments ?	boolean	If *true*, encapsulates Open Prepress Interface (OPI) low resolution images as a form and preserves information for locating the high resolution images. Default = *true*
TransferFunctionInfo ? New in JDF 1.1	enumeration	Determines how transfer functions are handled. Possible values are: *Preserve* – Transfer functions are passed into the PDF file. *Remove* – Transfer functions are ignored. They are neither applied to the color values nor passed into the PDF file. *Apply* – Transfer functions are used to modify the data that is written to the PDF file, instead of writing the transfer function itself to the file. Default = *Preserve*
UCRandBGInfo ? New in JDF 1.1	enumeration	Determines whether the arguments to the PostScript commands "setundercolorremoval" and "setblackgeneration" are passed into the PDF file. Possible values are: *Preserve* – The arguments are passed into the PDF file. *Remove* – The arguments are ignored. Default = *Preserve*
UsePrologue ?	boolean	If *true*, the process must prepend a PostScript prologue file to the job and append a PostScript epilog file to the job. Such files are used to control the PostScript environment for the conversion process. The expected location and allowable contents for these files is defined by the process implementation. Default = *false*

Structure of ThinPDFParams Subelement

Name	Data Type	Description
FilePerPage ?	boolean	If *true*, the process generates 1 PDF file per page. Default = *false*
SidelineFonts ?	boolean	If *true*, font data are stored in external files during PDF generation. Default = *false*
SidelineImages ?	boolean	If *true*, image data are stored in an external stream during the PDF Generation phase. This prevents large amounts of image data from having to be passed through all phases of the code generation process. Default = *false*

7.2.110   RegisterMark

Defines a register mark, which can be used for setting up and monitoring color registration in a printing process. It can also be used to synchronize the paper position in a paper path. The position and rotation of each register mark can be specified with the help of the following attributes. It is important that the register marks are defined in such a way that their centers are on the point of origin of the coordinate system, as otherwise they are not positioned properly.

Resource Properties

Resource class: Parameter
Resource referenced by: Surface
Example Partition: -
Input of processes: Any printing process
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the register mark in the coordinates of the MarkObject that contains this mark.
MarkType ?	NMTOKEN	Type of register mark. Possible values include: *Arc* *Circle* *Cross*
MarkUsage ? New in JDF 1.1	enumerations	Specifies the usage of the RegisterMark. Allowed values are: *Color* – The mark is used for separation color registration. *PaperPath* – The mark is used for paper path synchronization.
Rotation ?	double	Rotation in degrees. Positive graduation figures indicate counter-clockwise rotation; negative figures indicate clockwise rotation.
SeparationSpec *	element	Set of separations to which the register mark is bound.

7.2.111   RegisterRibbon

New in JDF 1.1

Description of register ribbons. For the register ribbon the length should be given. There are two parameters:
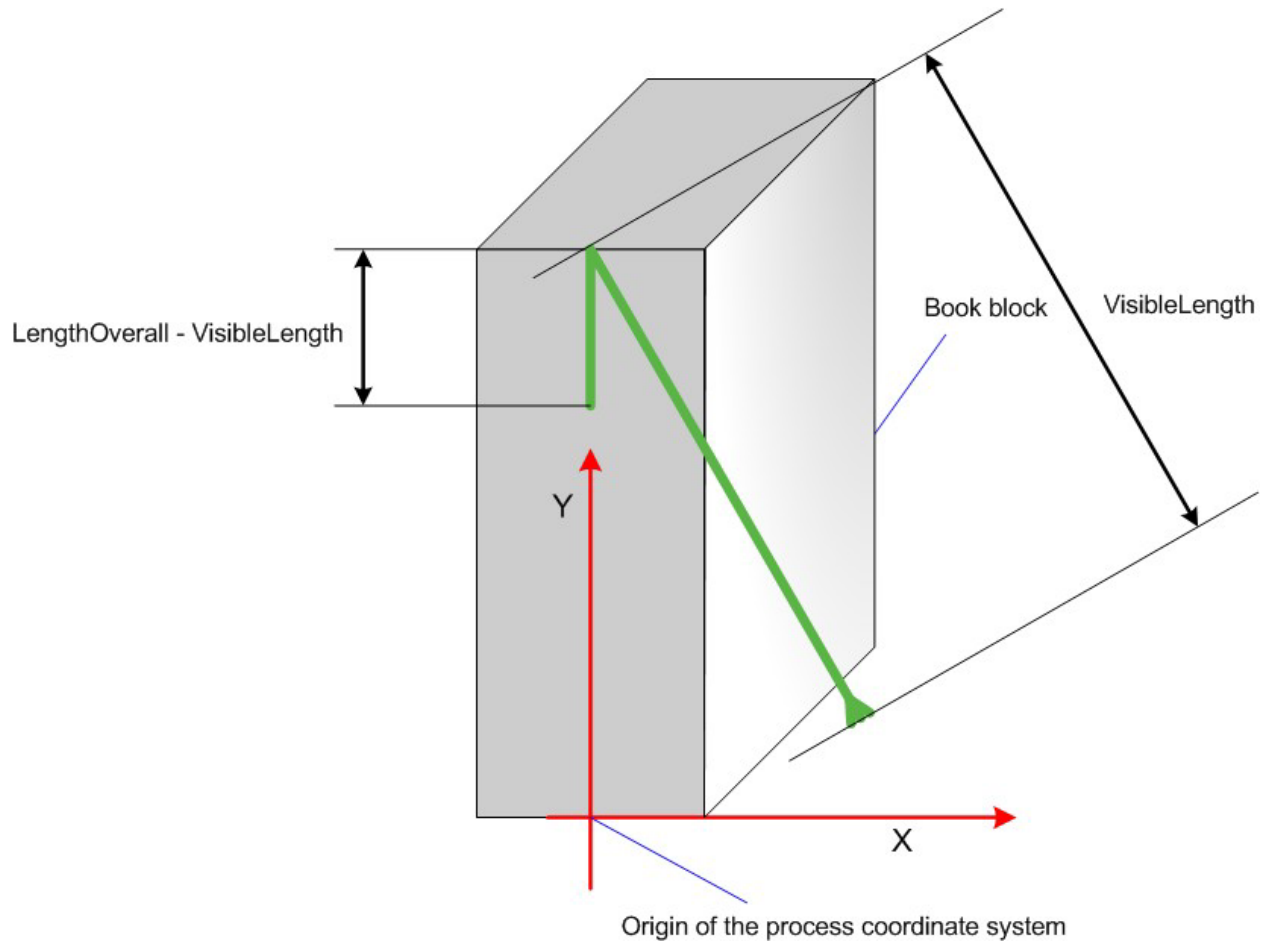
Figure 7.17Parameters and Coordinate System for BlockPreparation

Resource Properties

Resource class: Consumable
Resource referenced by: *BlockPreparationParams*
Example Partition: -
Input of processes: -

Resource Structure

Name	Data Type	Description
LengthOverall	number	Overall length of the register ribbon, i.e., 1+2 in the picture above.
Material ?	string	Material of the register ribbon. Default =system specified.
RibbonColor ?	NamedColor	Color of the ribbon. Default =system specified.
RibbonEnd ?	NMTOKEN	End of the Ribbon. Values include: *Cut* *CutSealed* *Knot* *SealedOffset* – The ribbon is sealed a distance from the cut. Default =system specified.
VisibleLength	number	Length of the register ribbon which will be seen when opening the book, i.e., 2 in picture above.

7.2.112   RenderingParams

This set of parameters identifies how the *Rendering* process should operate. Specifically, these parameters define the expected output of the **ByteMap** resource that the *Rendering* process creates.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *Rendering*
Output of processes: -

Resource Structure

Name	Data Type	Description
BandHeight ?	integer	Height of output bands expressed in lines. For a frame device, the band height is simply the full height of the frame. Default = device specific
BandOrdering ?	enumeration	Indicates whether output buffers are generated in *BandMajor* or *ColorMajor* order. Possible values are: *ColorMajor* – Only an option when dealing with non-interleaved data. Default = device specific
BandWidth ?	integer	Width of output bands expressed in pixels. Default = device specific
ColorantDepth ?	integer	Number of bits per colorant. Determines whether the output is bitmaps or bytemaps. A value of 1 implies that a bitmap is used and that halftone screening is performed by the interpretation process. Default = device specific
Interleaved ?	boolean	If *true*, the resulting colorant values are interleaved and *BandOrdering* is ignored. Default = device specific
AutomatedOverprint-Params ?	refelement	Optional controls for overprint substitutions. Defaults to no automated overprint generation.
ObjectResolution +	refelement	Elements which define the resolutions to render the contents at. More than one element may be used to specify different resolutions for different SourceObject types. Default = device specific
Media ? New in JDF 1.1	refelement	This resource provides a description of the physical media which will be marked. The physical characteristics of the media may affect decisions made during *Rendering*.

7.2.113   ResourceDefinitionParams

This set of parameters identifies how the *ResourceDefinition* process should operate. Specifically, these parameters define how default parameters of applications and the input resource should be combined.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: **ResourceDefinition**
Output of processes: -

Resource Structure

Name	Data Type	Description
DefaultID ? Deprecated in JDF 1.1	NMTOKEN	JDF ID of the default resource. If missing, it is assumed that the file specified by *DefaultJDF* contains only a JDF resource element, not a complete JDF.
DefaultJDF ?	URL	Link to a JDF resource that defines preset values.
DefaultPriority ?	enumeration	Defines whether preset values of the application or of the Resource specified in *DefaultJDF* have priority. Possible values are: *Application* – The application default settings are used to fill the resource. *DefaultJDF* – The Settings specified in *DefaultJDF* are applied. The default.
ResourceParam + New in JDF 1.1	refelement	Specification of the definition parameters of one individual resource.

Structure of the ResourceParam Subelement

New in JDF 1.1

Name	Data Type	Description
DefaultID ?	NMTOKEN	JDF ID of the default resource. If missing, it is assumed that the file specified by *DefaultJDF* contains only a JDF resource element, not a complete JDF.
DefaultJDF ?	URL	Link to a JDF resource that defines preset values. Defaults to the *DefaultJDF* specified in **ResourceDefinitionParams**.
DefaultPriority ?	enumeration	Defines whether preset values of the application or of the Resource specified in *DefaultJDF* have priority. Possible values are: *Application* *DefaultJDF* Defaults to the *DefaultPriority* specified in **ResourceDefinition-Params**.

7.2.114   RingBindingParams

This resource describes the details of the *RingBinding* process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *RingBinding*
Output of processes: -

Resource Structure

Name	Data Type	Description
BinderColor ?	NamedColor	Color of the ring binder.

Name	Data Type	Description
BinderMaterial ?	NMTOKEN	The following describe **RingBinding** binder materials used. Values include: *Cardboard* – Cardboard with no covering. *ClothCovered* – Cardboard with cloth covering. *PVC* – Solid PVC. *PVCCovered* – Cardboard with PVC covering.
BinderName ?	string	The name of the binder manufacturer and the name of the specific item.
RingDiameter ?	double	Diameter of the rings in points.
RingMechanic ?	boolean	If *true*, a hand lever is available for opening. Default = *false*
RingShape ?	NMTOKEN	The following **RingBinding** shapes are used: *Round* – The default. *Oval* *D-shape* *SlantD*
RingSystem ? Deprecated in JDF 1.1	enumeration	The following ring binding systems are used: *2HoleEuro* – in Europe *3HoleUS* – in North America *4HoleEuro* – in Europe
RivetsExposed ?	boolean	The following **RingBinding** choice describes mounting of ring mechanism in binder case. If *true*, the heads of the rivets are visible on the exterior of the binder. If *false*, the binder covering material covers the rivet heads. Default = *true*
SpineColor ?	NamedColor	Color of the binders spine.
SpineWidth ?	double	The spine width is determined by the final height of the block of sheets to be bound.
ViewBinder ?	NMTOKEN	The following **RingBinding** clear vinyl outer-wrap types are used on top of a colored base wrap: *Embedded* – Printed material is embedded by sealing between the colored and clear vinyl layers during the binder manufacturing. *Pocket* – Binder is designed so that Printed material may be inserted between the color and clear vinyl layers after the binder is manufactured.

7.2.115   RunList

RunList resources describe an ordered set of LayoutElement or ByteMap elements. Ordering and structure are defined using the generic partitioning mechanisms as described in 3.9.2 Description of Partitionable Resources.

   RunList resources are used whenever an ordered set of page descriptions elements are required. Depending on the process usage of a **RunList**, only certain *Types* of LayoutElement may be valid. For example, a pre-RIP imposition process requires LayoutElement elements of *Type* page or *document*, whereas a post-RIP imposition process requires ByteMap elements. The usage is detailed in the descriptions of the processes that use the **RunList** resource.

   RunList resources allow structuring of multiple *Pages* into *Documents*. Multiple *Documents* that have a joint context may be grouped into *Sets*.

Resource Properties

Resource class:	Parameter
Resource referenced by:	-
Example Partition:	*DocIndex, PartVersion, Run, RunPage, Separation*
Input of processes:	RunLists are used as input resources by most processes that act on content data
Output of processes:	RunLists are used as output resources by most processes that act on content data

Resource Structure

Name	Data Type	Description
Directory ?	URL	Defines a directory where the files that are associated with this **Runlist** should be copied to or from. If *Directory* is not specified, all FileSpec elements in the RunList must be completely specified.
DocCopies ? New in JDF 1.1	integer	Number of instance document copies that this **RunList** represents. Specifying *DocCopies* is equivalent to repeating the sequence of **RunList** leaves between *EndOfDocument* = *true* for a total of *DocCopies* times. Default = 1. Note: It is illegal to specify *DocCopies* with different values of in various leaves of a **RunList** representing the same instance document.
DocNames ?	NameRange-List	A list of named documents in a multi-document file that supports named access to individual documents. *DocNames* defaults to all documents. If *DocNames* occurs in the RunList, *Docs* is ignored if it is also present.
Docs ?	Integer-RangeList	0-based list of document indices in a multi-document file specified by the LayoutElement element.
EndOfDocument ?	boolean	If *true*, the last page in the RunList is the last page of an instance document. The precise handling of instance-document changes is defined in the **InsertSheet** resource. Default = *false*
EndOfSet ? New in JDF 1.1	boolean	If *true*, the last page in the RunList is the last page of a set of instance documents. The precise handling of instance-document boundaries is defined in the **InsertSheet** resource. Default = *false*
FirstPage ?	integer	First page in the document that is described by this RunList. This attribute is generally used to describe preseparated files. Default = 0
IsPage ?	boolean	If *true*, the individual RunList element defines one or more page slots, e.g., for filling PlacedObjects. If *false*, the first parent partitioned RunList element with IsPage = *true* defines the page level. Defaults to *true*. In general, *IsPage* will be *false* for separations of a preseparated RunList.
LogicalPage ? Modified in JDF 1.1	integer	The logical page number of the first page in a **RunList**. This attribute may be used to retain logical page indices when a partitioned **RunList** is spawned. It defaults to 1 plus the last page of the previous sibling RunList partition. If the RunList element is the first partition *LogicalPage* defaults to 0. Note that is an error to specify *LogicalPage* to be less than the number of previously defined logical pages, since this defines overlapping pages within the **RunList**.

Name	Data Type	Description
NDoc ? New in JDF 1.1	integer	Total number of instance documents that are defined by the **RunList**. If *NDoc* is not specified, it defaults to all instance documents in the partitioned **RunList** elements that make up the **RunList**.
NPage ?	integer	Total number of pages (placed object slots or **RunList** elements with *IsPage* = *true*) that are defined by the **RunList**. If *NPage* is not specified, it defaults to all pages in the partitioned **RunList** elements that make up the **RunList**. If the **RunList** describes multiple instance documents, *NPage* refers to the total number of pages in all instance documents.
NSet ? New in JDF 1.1	integer	Total number of instance document sets that are defined by the **RunList**. If *NSet* is not specified, it defaults to all instance document sets in the partitioned **RunList** elements that make up the **RunList**.
PageCopies ? New in JDF 1.1	integer	Number of page copies that this **RunList** represents. Specifying *PageCopies* is equivalent to repeating the **RunList** leaves representing each page for a total of *PageCopies* times. Default = 1. Note that pages specified by PageCopies are always assumed uncollated when calculating the index in the logical **RunList**, e.g., *PageCopies* = 2 would result in a logical page sequence of 0 0 1 1 2 2, etc.
PageNames ?	NameRange-List	A list of named pages in a multi-page file that supports named access to individual pages. *PageNames* defaults to all pages. If *PageNames* occurs in the RunList, *FirstPage, Npage, SkipPage* and *Pages* are ignored if any of them is also present.
Pages ? Modified in JDF 1.1A	Integer-RangeList	0-based list of indices in the documents specified by the LayoutElement element and the *Docs* and *Sets* attribute. If *Pages* is present, *FirstPage*, and *SkipPage* are ignored.
RunTag ? New in JDF 1.1	NMTOKEN	Tag of a partition of a resource other than the **RunList** which is partitioned by *RunTags.* The partition matches if any of the entries in the *RunTags* list matches *RunTag.* Multiple entries in a **RunList** may have the same *RunTag*.
SetCopies ? New in JDF 1.1	integer	Number of instance document set copies that this **RunList** represents. Specifying Set*Copies* is equivalent to repeating the sequence of **RunList** leaves between *EndOfSet* = "true" for a total of *SetCopies* times. Default=1. Note that it is illegal to specify *SetCopies* with different values of in various leaves of a **RunList** representing the same instance document.
SetNames ? New in JDF 1.1	NameRange-List	A list of named documents in a multi-document file that supports named access to individual documents. *SetNames* defaults to all documents. If *SetNames* occurs in the RunList, *Sets* is ignored if it is also present.
Sets ? New in JDF 1.1	Integer-RangeList	0-based list of document set indices in a multi-document file specified by the LayoutElement element.

Name	Data Type	Description
SkipPage ?	integer	Used when the **RunList** comprises every Nth page of the file. *SkipPage* indicates the number of pages to be skipped between each of the pages that comprise the RunList element. This is generally used to describe preseparated files, or to select only even or odd pages. Default = 0 Note: SkipPage is therefore 3 (4 Separations -> skip 3) in a CMYK separated file.
Sorted ?	boolean	Specifies whether the elements in the RunList are sorted in the document reader order. Default = *true*.
ByteMap ?	refelement	Describes the page or stream of pages. Only one of ByteMap or LayoutElement may be specified in one RunList element. If neither ByteMap nor LayoutElement are specified, the RunList entry specifies empty content.
DynamicInput *	element	Replacement text for a DynamicField element. This information defines the contents of a dynamic mark on the Layout for automated page layout. The mark must be filled using information from the document runlist, such as the bar code of the recipient. This information varies with the document content. DynamicInput elements have one optional *Name* attribute that, when linked to the *ReplaceField* attribute of the DynamicField element, defines the string that should be replaced.
InsertSheet *	refelement	Describes how Sheets and Surfaces may be completed and optional media which may be inserted at the beginning or end of this RunList element.
LayoutElement ?	refelement	Describes the document, page or image. Only one of ByteMap or LayoutElement may be specified in one RunList element. If neither ByteMap nor LayoutElement are specified, the RunList entry specifies empty content.

Structure of a DynamicInput Subelement

DynamicInput defines the contents of a dynamic mark on a **Surface** resource for automated page layout. The mark must be filled using information from the document runlist, such as the bar code of the recipient. This information varies with the document content. For details on dynamic marks, see the DynamicField element description in Section 7.2.133 Surface.

Name	Data Type	Description
Name ?	string	Label that must match the *ReplaceField* attribute of the appropriate DynamicField element
-	text	Defines the text string that should be inserted as a replacement for the text defined in *ReplaceField* of a DynamicField element.

Examples of partitioning of a RunList

The following examples illustrate how a RunList can be structured using partitioning Mechanisms. Note that the partitioning of a RunList often generates the values necessary to evaluate the partitioning of other resources, e.g., the RunIndex into the RunList. Thus, the order in which the RunLists appear in the XML document is significant. It is interesting to note that the "Run" partitioning key has a string value, and is not required to be numeric.

Simple unstructured Single-File Runlist

This example specifies all pages contained in "in/colortest.pdf".

```
<RunList ID="Link0003" Pages="0~-1" Class="Parameter" Status="Available">
  <LayoutElement>
```

```
      <FileSpec URL="File://in/colortest.pdf"/>
   </LayoutElement>
</RunList>
```

Simple Multi-File unseparated RunList using RunList::Directory

This example specifies all pages contained in "File1.pdf" and "File2.pdf", which are located in the directory "//Dir" that is specified in **RunList**::*Directory*.

```
<RunList ID="Link0003" Class="Parameter" Status="Available" PartIDKeys="Run" Direc-
tory="File://Dir/">
   <RunList Run="1" Pages="0~-1">
     <LayoutElement>
       <FileSpec URL="File1.pdf"/>
     </LayoutElement>
   </RunList>
   <RunList Run="2" Pages="0~-1">
     <LayoutElement>
       <FileSpec URL="File2.pdf"/>
     </LayoutElement>
   </RunList>
</RunList>
```

Simple Multi-File unseparated RunList with independent spawning

This example specifies the first five pages contained in File1.pdf and File2.PDF. File2.pdf has been spawned and is being processed individually.

```
<RunList ID="Link0003" Class="Parameter" Status="Available" PartIDKeys="Run">
   <RunList Run="1" Pages="0~4">
     <LayoutElement>
       <FileSpec URL="File://File1.pdf"/>
     </LayoutElement>
   </RunList>
   <RunList Run="2" SpawnStatus="SpawnedRW" Pages="0~-1">
     <LayoutElement>
       <FileSpec URL="File://File2.pdf"/>
     </LayoutElement>
   </RunList>
</RunList>
```

This is the corresponding spawned RunList. Note the *LogicalPage* attribute, which specifies the number of skipped pages.

```
<RunList ID="Link0003" Class="Parameter" Status="Available" PartIDKeys="Run" Run="2"
   LogicalPage="5" Pages="0~-1">
   <LayoutElement>
     <FileSpec URL="File://File2.pdf"/>
   </LayoutElement>
</RunList>
```

Simple Multi-File separated RunList

This example specifies all pages contained in Presep.pdf and following that, pages 1, 3, and 5 of each preseparated file.

```
<RunList ID="Link0003" Class="Parameter" Status="Available" PartIDKeys="Run Separation">
   <RunList Run="1" SkipPage="3">
     <LayoutElement>
       <FileSpec URL="File://Presep.pdf"/>
     </LayoutElement>
     <RunList Separation="Cyan" FirstPage="0" IsPage="false"/>
     <RunList Separation="Magenta" FirstPage="1" IsPage="false"/>
     <RunList Separation="Yellow" FirstPage="2" IsPage="false"/>
     <RunList Separation="Black" FirstPage="3" IsPage="false"/>
   </RunList>
   <RunList Run="2" Pages="1 3 5" IsPage="true">
     <RunList Separation="Cyan" IsPage="false">
       <LayoutElement>
         <FileSpec URL="File://Cyan2.pdf"/>
       </LayoutElement>
```

```
    </RunList>
    <RunList Separation="Magenta" IsPage="false">
      <LayoutElement>
        <FileSpec URL="File://Magenta2.pdf"/>
      </LayoutElement>
    </RunList>
    <RunList Separation="Yellow" IsPage="false">
      <LayoutElement>
        <FileSpec URL="File://Yellow2.pdf"/>
      </LayoutElement>
    </RunList>
    <RunList Separation="Black" IsPage="false">
      <LayoutElement>
        <FileSpec URL="File://Black2.pdf"/>
      </LayoutElement>
    </RunList>
  </RunList>
</RunList>
```

7.2.116   SaddleStitchingParams

This resource provides the parameters of the *SaddleStitching* process.
Deprecated in JDF 1.1

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *SaddleStitching*
Output of processes: -

Resource Structure

Name	Data Type	Description
NumberOfStitches	integer	The number of stitches that will be made.
StitchPositions ?	NumberList	Array containing the stitch positions along the saddle. The center of the stitch must be specified, and the number of entries must match the number given in the *NumberOfStitches* attribute.
StapleShape ?	enumeration	Shape of staples. Possible values are: *Crown* *Overlap* *Butted* *ClinchOut* *Eyelet* These values are displayed in Figure 7.18, below.
StitchWidth ?	double	Width of each stitch.
WireGauge ?	double	Gauge of the wire being used.
WireBrand ?	string	Brand of wire being used.
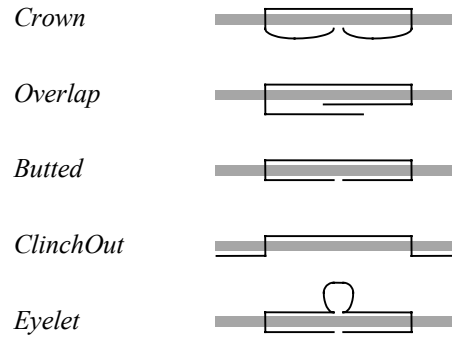
Figure 7.18 Staple shapes

The process coordinate system is defined as follows — The Y-axis is aligned with the binding edge, and increases from the registered edge to the edge opposite the registered edge. The X-axis, meanwhile, is aligned with the registered edge. It increases from the binding edge to the edge opposite the binding edge, which is the product front edge.

7.2.117   **ScanParams**

This resource provides the parameters for the **Scanning** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *RunIndex*
Input of processes: *Scanning*
Output of processes: -

Resource Structure

Name	Data Type	Description
BitDepth	integer	Bit depth of a one-color separation.
CompressionFilter ?	enumeration	Specifies the compression filter to be used. Possible values include: *CCITTFaxEncode* – Used to select CCITT Group 3 or 4 facsimile encoding. *DCTEncode* – Used to select JPEG compression. *FlateEncode* – Used to select ZIP compression. *WaveletEncode* – Used to select Wavelet compression. *JBIG2Encode* – Used to select JBIG2 monochrome compression.
DCTQuality ?	number	A value between 0 and 1 that indicates "how much" the process should compress images. 0.0 means "do as loss-less compression as possible." 1.0 means "do the maximum compression possible."
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes color corrections. The *ResourceUsage* attribute of the FileSpec must be "*CorrectionProfile*".
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that defines the target output device for a device specific scan, such as the profile of a CMYK press. The *ResourceUsage* attribute of the FileSpec must be "*TargetProfile*".
FileSpec ?	refelement	A **FileSpec** resource pointing to an ICC profile that describes the scanner. The *ResourceUsage* attribute of the FileSpec must be "*ScanProfile*".

Name	Data Type	Description
InputBox ?	rectangle	Rectangle that describes the image section to be scanned, in points. The origin of the coordinate system is the lower left corner of the physical item to be scanned.
Magnification ?	XYPair	Size of the output/size of the input for each dimension. Default = 1.0.
MountID ?	string	ID of the drum or other mounting device upon which the media should be mounted.
Mounting ?	enumeration	Specifies how to mount originals. Possible values are: *Unfixed* – Original lies unfixed on the scanner tray/drum. *Fixed* – Original is fixed on the scanner tray/drum with transparent tape. *Wet* – Original is put in gel or oil and fixed on the scanner tray/drum. *Registered* – Original is fixed with registration holes. This value is used for copix.
OutputColorSpace	enumeration	Color space of the output images. Possible values are: *LAB* *RGB* *CMYK* *GrayScale*
OutputResolution	XYPair	X and Y resolution of the output bitmap in DPI.
OutputSize ?	XYPair	X-,Y-dimension of the intended output image in points.
SplitDocuments ?	integer	A number representing how many images are scanned before a new file is created.

7.2.118   ScavengerArea

New in JDF 1.1

This resource describes a scavenger area for removing excess ink from printed sheets. It is defined within a MarkObject of a **Surface**.

Resource Properties

Resource class: Parameter
Resource referenced by: Surface
Example Partition: -
Input of processes: *Any printing process*
Output of processes: -

Resource Structure

Name	Data Type	Description
Center	XYPair	Position of the center of the scavenger area in the coordinates of the MarkObject that contains this mark.
Rotation ?	double	Rotation in degrees. Positive graduation figures indicate counter-clockwise rotation; negative figures indicate clockwise rotation.
Size	XYPair	Size of the scavenger area.
SeparationSpec *	element	Set of separations to which the scavenger area is bound.

7.2.119   ScreeningParams

This resource specifies the parameter of the screening process. Since screening is, in most cases, very OEM specific, the following parameters are generic enough that they can be mapped onto a number of OEM controls.

Resource Properties

Resource class: Parameter
Resource referenced by: ExposedMedia
Example Partition: *Separation, SheetName, Side, SignatureName*
Input of processes: *Screening*, *ColorCorrection*
Output of processes: -

Resource Structure

Name	Data Type	Description
IgnoreSourceFile ?	boolean	Specifies whether to ignore the screen settings (such as setscreen, setcolorscreen, and sethalftone) specified in the source files. Default = *true* Note: In some cases, Halftones are used to create patterns. In these cases, the halftone in the source PDL file will not be over-ridden.
AbortJobWhenScreen-MatchingFails ?	boolean	Specifies what happens when the device can not fulfill the screening requests. If *true*, it flushes the job. If *false*, it ignores matching errors using the default screening. Default = *false*
ScreenSelector * Modified in JDF 1.1	element	List of screen selectors. A screen selector is included for each separation, including a default specification.

Structure of ScreenSelector Subelement

Description of screening for a selection of source object types and separations.

Name	Data Type	Description
Angle ?	double	Specifies the angle of the screen when AM screening is used. Only one of *Angle* or *AngleMap* may be specified. If neither *Angle* or *AngleMap* are specified, the angle is determined by the default of the selected *ScreeningFamily.*

AngleMap ?
 New in JDF 1.1
	string	Specifies the mapping of the angle of the screen to the angle of a different separation when AM screening is used, e.g., a spot color that has the same screening angle as the cyan separation is specified by *AngleMap* = *Cyan*. Only one of *Angle* or *AngleMap* may be specified. This mapping is not transitive, so, when *Separation* already specifies a color with a known default[4], it specifies the angle of the separation defined by *AngleMap* prior to that separation being mapped, e.g., the following example specifies that *Black* should be mapped to the *Cyan* default separation and *Cyan* to the *Black* default separation.
		
		 `<ScreenSelector AngleMap="Black" Separa-`
		`tion="Cyan"/>`
		
		 `<ScreenSelector AngleMap="Cyan" Separa-`
		`tion="Black"/>`

DotSize ? New in JDF 1.1	double	Specifies the dot size of the screen in micron[μm] when FM screening is used.
Frequency ?	double	Specifies the line frequency in lpi of the screen when AM screening is used.

---

[4] In general this will be a CMYK process color, but it can also be another process color, e.g., HexaChrome™

Name	Data Type	Description
ScreeningFamily ?	string	Vendor specific screening family name. Possible values include: *Rational Tangent* *Adobe Accurate* *Agfa Balanced* *Soft-IS* *ErrorDiffusion*
ScreeningType ?	enumeration	General type of screening. One of *AM:* the default. *FM* *Adaptive*
Separation ?	string	The name of the separation. If *Separation* = *All*, the ScreenSelector should be applied to all separations that are not specified explicitly. Default = *All*
SourceFrequency ?	double	Specifies the line frequency of screens which should be matched from the source file when screen matching is to be done for AM screening.
SourceObjects ?	enumerations	Identifies the class(es) of incoming graphical objects on which to use the selected screen. Possible values are: *All* – Default value. *ImagePhotographic* – Contone images. *ImageScreenShot* – Images largely comprised of rasterized vector art. *Text* *LineArt* *SmoothShades* – Gradients and blends.
SpotFunction ?	NMTOKEN	Specifies the spot function of the screen when AM screening is used. These names are the same as the spot function names defined in PDF: *Round* *Diamond* *Ellipse* *EllipseA* *InvertedEllipseA* *EllipseB* *EllipseC* *InvertedEllipseC* *Line* *LineX* *LineY* *Square* *Cross* *Rhomboid* *DoubleDot* *InvertedDoubleDot* *SimpleDot*

Name	Data Type	Description
		InvertedSimpleDot
		CosineDot
		Double
		InvertedDouble

7.2.120   SeparationControlParams

This resource provides the controls needed to separate composite color files.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Separation*
Output of processes: -

Resource Structure

Name	Data Type	Description
AutomatedOverprint-Params ?	refelement	Optional controls for overprint substitutions. Default = no automated overprint generation.
TransferFunctionControl ?	refelement	Controls whether the device performs transfer functions and what values are used when doing so.

7.2.121   SeparationSpec

This resource specifies a specific separation, and is usually used to define a list or sequence of separations.

Resource Properties

Resource class: ResourceElement
Resource referenced by: **ColorantControl**, **LayoutElement**, **RegisterMark**, **TransferFunctionControl**
Example Partition: -
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Name	string	Name of one specific separation.

7.2.122   ShapeCuttingParams

New in JDF 1.1

ShapeCuttingParams defines the details of the ShapeCutting process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ShapeCutting*
Output of processes: -

Resource Structure

Name	Data Type	Description
Shape *	element	details of each individual shape

Structure of Shape Subelement

Name	Data Type	Description
CutBox ?	rectangle	Specification of a rectangular window.
CutOut ?	boolean	If *true*, the inside of a specified shape will be removed. If *false*, the outside of a specified shape will be removed. An example of an inside shape is a window, while an example of an outside shape is a shaped greeting card. Default = *false*
CutPath ?	path	Specification of a complex path. This may be an open path in the case of a single line.
Material ?	string	Transparent material that fills a shape, such as an envelope window, that was cut out when *CutOut* = *true*.
CutType ?	enumeration	Type of cut or perforation used. Possible values are: *Cu* – Full cut. *Perforate* – Interrupted perforation that does not span the entire sheet
ShapeDepth ?	double	Depth of the shape cut. Measured in micron[µm]. If not specified, the shape is completely cut.
ShapeType	enumeration	Describes any precision cutting other than hole making. Possible values are: *Rectangular* *Round* *Path*
TeethPerDimension ?	number	Number of teeth in a given perforation extent in teeth/point. *MicroPerforation* is defined by specifying a large number of teeth (n>1000).

7.2.123   Sheet

This resource provides a description of a sheet, as well as the marks on that sheet.

Resource Properties

Resource class: Parameter
Resource referenced by: InsertSheet, Layout
Example Partition: *SheetName*. Otherwise it is strongly discouraged to partition the **Layout** tree, including **Sheet**.
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
LockOrigins ?	boolean	Determines the relationship of the coordinate systems for front and back surfaces.
		When *false*, all contents for all surfaces are transformed into the first quadrant, in which the origin is at the lower left corner of the surface.
		When *true*, contents for the front surface are imaged into the first quadrant (as above), but contents for the back surface are imaged into the second quadrant, in which the origin is at the lower right. This allows the front and back origins to be aligned even if the exact media size is unknown.
		Default = *false*

Name	Data Type	Description
Name ?	string	Unique name of the sheet. *Name* is used for external reference to a sheet in, for example, a Part element.
SurfaceContentsBox ?	rectangle	This box, specified in surface-coordinate space, defines the area into which contents and marks will occur for all Surfaces in the **Sheet**. CTMs for MarkObjects or ContentObjects transform page contents or marks into this rectangle.
InsertSheet *	refelement	Specifies how to complete a sheet in an automated printing environment.
Media? New in JDF 1.1	refelement	Describes the media to be used.
MediaSource ? Deprecated in JDF 1.1	refelement	Describes the media to be used. Replaced by **Media** in JDF 1.1.
Surface (Front) ?	refelement	Describes the front surface to be used. Two surfaces may be attached: one front surface and one back surface. The surface is defined by the *Side* attribute of the **Surface** resource. The *Side* attribute of this Surface element must be *Front*.
Surface (Back) ?	refelement	Describes the back surface to be used. The *Side* attribute of this Surface element must be *Back*.

7.2.124　ShrinkingParams

New in JDF 1.1

This resource provides the parameters for the **Shrinking** process in shrink wrapping.

Resource Properties

Resource class:　　　　Parameter
Resource referenced by: -
Example Partition:　　　-
Input of processes:　　*Shrinking*
Output of processes:　　-

Resource Structure

Name	Data Type	Description
Duration ?	duration	Shrinking time. Default = equipment-specific value.
ShrinkingMethod ?	enumeration	Specifics of the shrinking method for shrink wrapping. *ShrinkCool* *ShrinkHot* – The default.
Temperature ?	number	Oven temperature in ° Centigrade. Default = equipment-specific value.

7.2.125　SideSewingParams

Deprecated in JDF 1.1

This resource provides the parameters for the **SideSewing** process. SideSewing is a special case of *ThreadSewing*. The process coordinate system is defined in the following way: the Y-axis is aligned with the binding edge. It then increases from the registered edge to the edge opposite to the registered edge. The X-axis is aligned with the registered edge, which then increases from the binding edge to the edge opposite to the binding edge, i.e., the product front edge.

Figure 7.19 Parameters and coordinate system used for side sewing

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *SideSewing*
Output of processes: -

Resource Structure

Name	Data Type	Description
NumberOfNeedles	integer	Specifies the number of needles to be used.
NeedlePositions ?	NumberList	Array containing the Y-coordinates of the needle positions. The number of entries must match the number given in *NumberOf-Needles*.
Offset	double	Specifies the distance between the stitch and the binding edge.
SewingPattern ?	enumeration	Specifies the sewing pattern to be used. Possible values are: *Normal* *Staggered* *CombinedStaggered*
ThreadMaterial ?	enumeration	Specifies the thread material to be used. Possible values are: *Cotton* *Nylon* *Polyester*
ThreadThickness ?	double	The thickness of the thread to be used.
ThreadBrand ?	string	The brand of thread to be used.

7.2.126   SpinePreparationParams

New in JDF 1.1

SpinePreparationParams describes the preparation of the spine of book blocks for hard and soft cover book production, e.g., milling and notching.

Resource Properties

Resource class: Parameter

Resource referenced by: -
Example Partition: -
Input of processes: *SpinePreparation*
Output of processes: -

Resource Structure

Name	Data Type	Description
FlexValue ?	double	Flex quality parameter given in [N/cm].
MillingDepth	double	Milling depth in points. This describes the total cut-off of the spine, regardless of the technology used to achieve this goal.
NotchingDistance ?	double	Notching distance in points.
NotchingDepth ?	double	Notching depth relative to the leveled spine in pt. Default = 0, i.e., no notching.
Operations ?	NMTOKENS	List of operations to be applied to the spine. Duplicate entries are allowed to specify a sequence of identical operations. The order of operations is significant. Possible values include: *Brushing* – Brushes away dust from the spine to improve the binding quality. *FiberRoughing* – The fibers of the paper on the spine are exposed without the risk of glazing the paper coating. This optimizes the spine preparation considering paper and adhesive types. *Leveling* – After milling the spine, any uneven areas are leveled to achieve an even surface. *Milling* – Cuts of a part of the spine in a way that the spine is not to evenly. A rough texture of the fibers is assured. This creates ideal conditions for stable anchoring of the sheets in the glue. *Notching* – This gives a clamping effect on the spine which is de-sirable for some products. *Sanding* – Is used for voluminous book papers. *Shredding* – Produces a relatively smooth surface. Further opera-tions like *Notching*, *Leveling*, *FiberRoughing*, *Sanding* or *Brush-ing* are necessary.
PullOutValue ?	double	Pull out quality parameter given in [N/cm].
StartPosition ?	double	Starting position of milling tool (along the Y-axis of the operation coordinate system). Default = 0
WorkingLength ?	double	Working length of milling operation. If specified larger than the Spine length, the complete Spine is prepared. If not specified, the complete spine is prepared.

Figure 7.20 Parameters and coordinate systems for the SpinePreparation process

7.2.127   SpineTapingParams

New in JDF 1.1

SpineTapingParams define the parameters for taping a strip tape or kraft paper to the spine of a book block.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *SpineTaping*
Output of processes: -

Resource Structure

Name	Data Type	Description
HorizontalExcess ?	double	Taping spine excess on each side. The tape is assumed to be centered between left and right. Default =system specified.
StripBrand ?	string	Strip brand. Default =system specified.
StripColor ?	NamedColor	Color of the strip. Default =system specified.
StripLength ?	double	Length of strip material along binding edge. If not defined, default = spine length.

Name	Data Type	Description
StripMaterial ?	enumeration	Strip material. Possible values are: *Calico* *Cardboard* *CrepePaper* *Gauze* *Paper* *PaperlinedMules* *Tape* Default =system specified.
TopExcess ?	double	Top spine taping excess. This value may be negative. Default = 0
GlueApplication *	refelement	Describes where and how to apply glue to the book block.
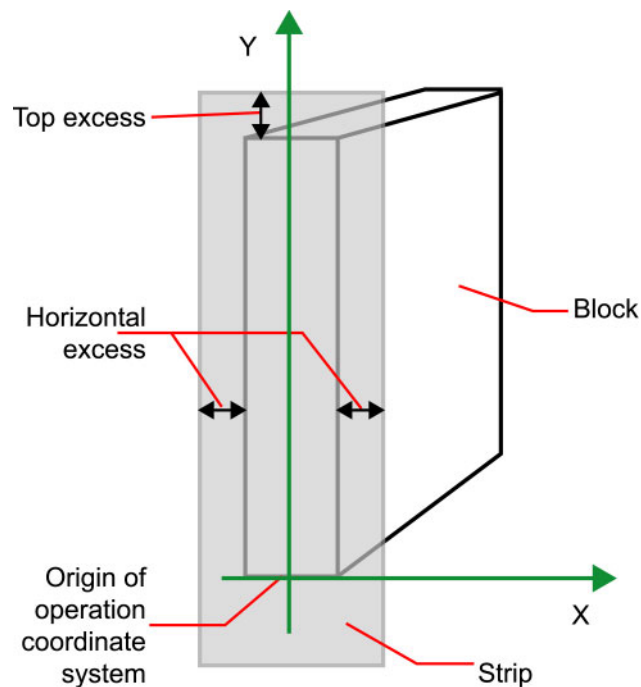
Figure 7.21 Parameters and coordinate system for the SpineTaping process

7.2.128   StackingParams

New in JDF 1.1

Settings for the *Stacking* process. A stack of components may be uneven and unstable, due to variations in thickness across each component. The thickness variations may be caused by folding, binding, or inserted components. A stack may be split into layers, with successive layers rotated by 180° to compensate for the unevenness.
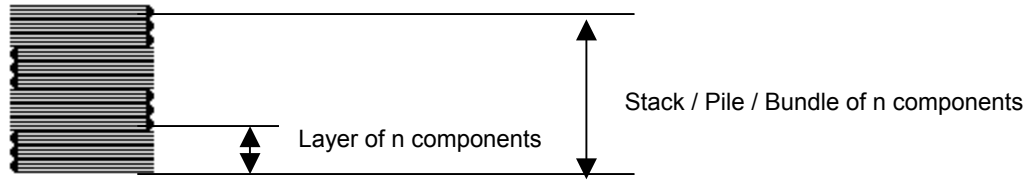
If the thickest part is on an edge, e.g., a book binding, the components may be offset to separate the thick parts. Layer compensation and offsetting may be combined as in the following examples.
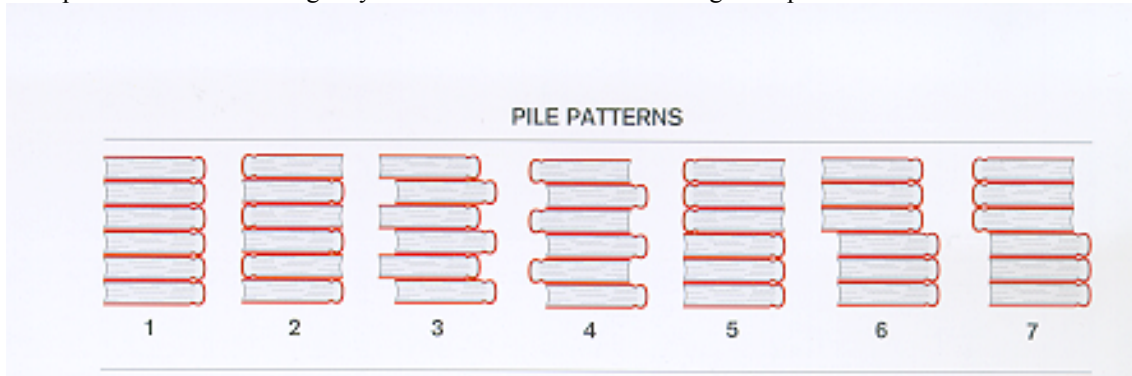
Table 7-3 Parameters in Stacking

Pile Pattern	StandardAmount	LayerAmount (Default = StandardAmount)	Compensate (Default = true)	Offset (Default = false)
1	6	6	*true*	*false*
2	6	1	*true*	*false*
3	6	1	*false*	*true*
4	6	1	*true*	*true*
5	6	3	*true*	*false*
6	6	3	*false*	*true*
7	6	3	*true*	*true*

If the number of components is not evenly divisible by standard stack size (StandardAmount) or the number of components in a bundle is not evenly divisible by layer size (LayerAmount), there will be a remainder, yielding one or more odd-count stacks or layers. By default, the odd-count stack or layer size may contain as few as one component. This may exceed equipment cycle times, and flimsy components (newspapers) may cause problems with downstream equipment such as strappers. MinAmount and MaxAmount control the minimum and maximum size of odd-count stacks and layers. The following figures show the odd count handling for bundles and layers.
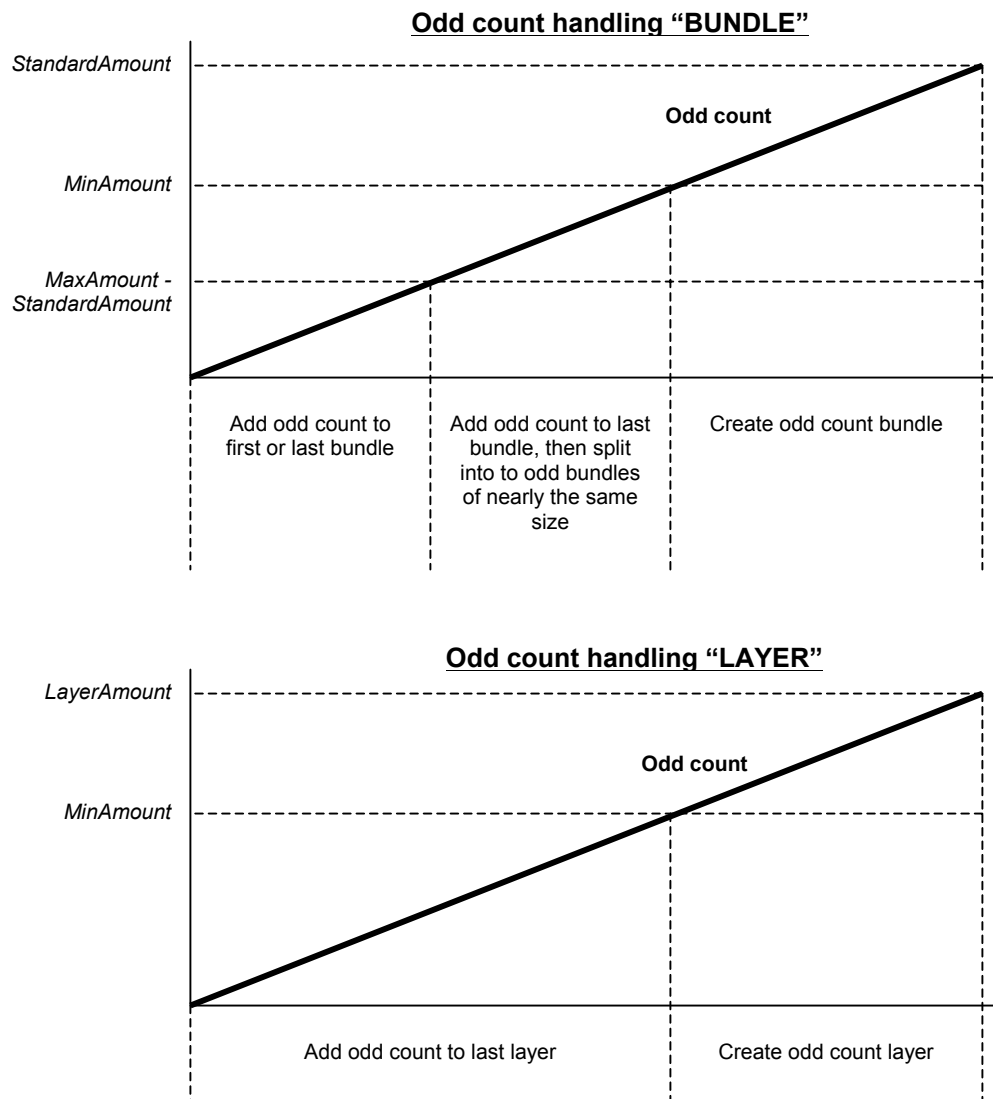
Odd count handling "BUNDLE"

Add odd count to first or last bundle	Add odd count to last bundle, then split into to odd bundles of nearly the same size	Create odd count bundle

Odd count handling "LAYER"

Add odd count to last layer	Create odd count layer

Resource Properties

Resource class: Parameter
Resource referenced by:
Example Partition: -
Input of processes: *Stacking*
Output of processes: -

Resource Structure

Name	Data Type	Description
LayerAmount?	integer	Number of products in a layer, typically an even divisor of *StandardAmount.* Default = *StandardAmount*
StandardAmount	integer	Number of products in a standard stack
MaxAmount ?	integer	Maximum number of products in a stack, *MaxAmount >= StandardAmount.* Default = *StandardAmount*

Name	Data Type	Description
MinAmount ?	integer	Minimum number of products in a stack or layer, *(MaxAmount – StandardAmount) <= MinAmount < StandardAmount* and *MinAmount < LayerAmount*. Defaults to *(MaxAmount – Standar-dAmount)*
MaxWeight ?	number	Maximum weight of a stack in grams. Default = infinity
Compensate?	boolean	180 degree rotation applied to successive layers to compensate for un-even stacking. Default = *true*. If *LayerAmount = StandardAmount*, there is one layer, and effectively no compensation.
Offset?	boolean	Offset or Shift applied to successive layers to separate the thicker por-tions of components, for example, offsetting the spines of hardcover books. Default = *false*

7.2.129   StitchingParams

This resource provides the parameters for the ***Stitching*** process. The process coordinate system is defined as follows:

A: The y-axis increases from the (first) registered edge to the edge opposite to the registered edge. The X-axis is aligned with the (second) registered edge. It increases from the binding edge (or first registered edge) to the edge opposite to the binding edge (or first registered edge).

B.   When the ***ReferenceEdge*** attribute is present, then the process coordinate system is defined relative to the layout coordinate system: *Left* is defined to be along the sheet layout Y-axis, *Bottom* is defined to be along the document content X-axis, *Right* is opposite *Left*, and *Top* is opposite *Bottom*. To position a staple in the top left corner of a portrait document, the ***ReferenceEdge*** value must be *Left*. For details on coordinate systems, see 2.5.3 Coordinate Systems of Resources and Processes.

Note that the stitches are applied from the front in the figures describing the stitching coordinate system.

Crown

Overlap

Butted

ClinchOut

Eyelet
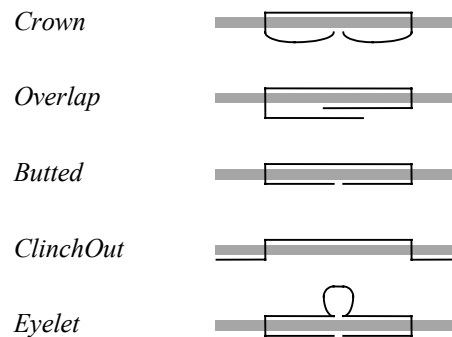
Figure 7.22 Staple shapes
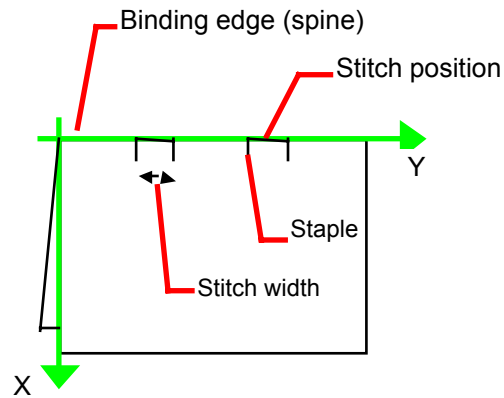
Figure 7.23 Parameters and coordinate system used for saddle stitching
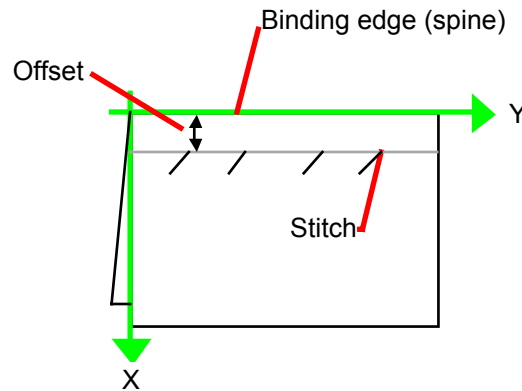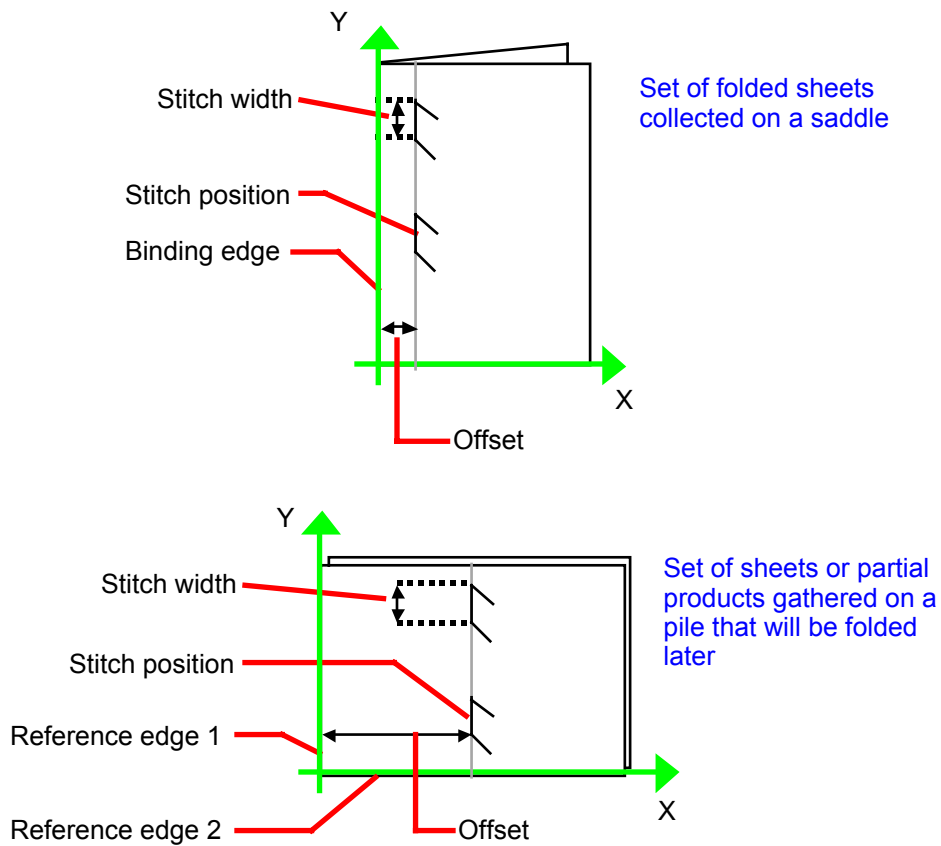
Figure 7.24 Parameters and coordinate system used for stitching

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -

Input of processes: *Stitching*
Output of processes: -

Resource Structure

Name	Data Type	Description
Angle ?	double	Angle of stitch in degree. The angle increases in a counterclockwise direction. 0 = horizontal, which means that it is parallel to the X-axis of the operation coordinate system. Defaults to the system specified value which may vary depending on other attributes set in this resource. If *StitchType = Saddle, Angle* must be ignored
NumberOfStitches ?	integer	Number of stitches. If not present or -1, means use the system specified number of stitches which may vary depending on other attributes set in this resource.
ReferenceEdge ? New in JDF 1.1	enumeration	The edge or corner of the component to be stitched for the process coordinate system (see description above). This attribute is intended for use when the *Stitching* process is combined with other processes, such as *DigitalPrinting*, where, when combined, there is no input **Component** to be stitched. Possible values are: Top Left Right Bottom *SystemSpecified* – Default to the process coordinate system defined in A above. The default.
Offset ?	double	Distance between stitch and binding edge. If *StitchType = Saddle, Offset* must be ignored. Note that it is possible to describe saddle stitching with an offset by defining *StitchType = Side* with a large *Offset* value.
StapleShape ?	enumeration	Specifies the shape of the staples to be used. Possible values are: *Crown* *Overlap* *Butted* *ClinchOut* *Eyelet* Representations of these values are displayed in Figure 7.18. Default = equipment-specific setting
StitchFromFront ?	boolean	If *true*, *Stitching* is done from front to back. Otherwise it is done from back to front. Default =system specified.
StitchPositions ?	NumberList	Array containing the stitch positions. The center of the stitch must be specified, and the number of entries must match the number given in *NumberOfStitches*.
StitchType ?	enumeration	Specifies the type of the *Stitching* operation. One of: *Corner* – Stitch in the corner that is at the clockwise end of the reference edge. *Saddle* – Stitch the on the middle fold which is on the saddle. *Side* – Stitch along the reference edge. *SystemSpecified* – The system specified value. The default.

Name	Data Type	Description
StitchWidth ?	double	Width of the stitch to be used. If not present or 0, means use the system specified width of stitches which may vary depending on other attributes set in this resource.
WireGauge ?	double	Gauge of the wire to be used. If not present or 0, means use the system specified wire gauge which may vary depending on other attributes set in this resource.
WireBrand ?	string	Brand of the wire to be used.

7.2.130   Strap

New in JDF 1.1

Resource Properties
Resource class: Consumable
Resource referenced by:
Example Partition: -
Input of processes: *Strapping*
Output of processes: -

Resource Structure

Name	Data Type	Description
StrapColor ?	NamedColor	Color of the string or strap. defaults to "*any*".
Material	enumeration	Strap material. AdhesiveTape Strap String

7.2.131   StrappingParams

New in JDF 1.1

StrappingParams defines the details of *Strapping*.

Resource Properties
Resource class: Parameter
Resource referenced by:
Example Partition: -
Input of processes: *Strapping*
Output of processes: -

Resource Structure

Name	Data Type	Description
StrappingType	enumeration	Can be: *Single* – One strap *Double* – Two parallel single straps *Cross* – Two crossed straps *DoubleCross* – Two cross straps that strap each side of a box. Default = equipment-specific setting

7.2.132   StripBindingParams

New in JDF 1.1

This resource describes the details of the **StripBinding** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: **StripBinding**
Output of processes: -

Resource Structure

Name	Data Type	Description
Brand ?	string	The name of the comb manufacturer and the name of the specific item. Default =system specified.
Distance ?	double	The distance between the pins and the distance between the holes of the prepunched sheets must be the same. Default =system specified.
Length ?	double	The length of the pin is determined by the height of the pile of sheets to be bound. Default =system specified.
StripColor ?	NamedColor	Determines the color of the strip. Default =system specified.

7.2.133   Surface

This resource describes the marks on a sheet surface. Up to two **Surface** resources may be defined for a **Sheet**.

Resource Properties

Resource class: Parameter
Resource referenced by: **Sheet**
Example Partition: *Side*. Otherwise it is strongly discouraged to partition the **Layout** tree, including **Surface**.
Input of processes: -
Output of processes: -

Resource Structure

Name	Data Type	Description
Side	enumeration	The side of the **Sheet** that the **Surface** describes. Possible values are: *Front* *Back*
SurfaceContentsBox ?	rectangle?	This rectangle provides the region of the surface into which the

Name	Data Type	Description
		contents of ContentObjects and MarkObjects are to be imaged.
		Note: The SurfaceContentBox also provides a translation for an object's CTM.
PlacedObject *	element	Provides a list of the ContentObject and MarkObject elements to be placed on to the surface. Contains the marks on the surface in rendering order. See the description that follows.
		Note: PlacedObject is not a container but an abstract type.

Structure of the Abstract PlacedObject Subelement

The marks that may be placed on the designated **Surface** come in two varieties: ContentObject or MarkObject elements. Both inherit characteristics from the abstract PlacedObject element type, and both are described below.

Name	Data Type	Description
ClipBox ?	rectangle	Clipping rectangle in the coordinates of the *SurfaceContentsBox*.
CTM	matrix	Transformation matrix of the object in the *SurfaceContentsBox*.
HalfTonePhaseOrigin ?	XYPair	Location of the origin for screening of this ContentObject. Specified in the coordinate systems of *SurfaceContentBox*. Default = 0 0
LayerID ? New in JDF 1.1	integer	If a layout supports layering, e.g., for versioning, *LayerID* may be used to identify the layer that a ContentObject belongs to, e.g., the language layer version. The details of the layers are optionally specified in the **Layout**::LayerList::LayerDetails key.
OrdID ? New in JDF 1.1	integer	If a layout supports layering, e.g., for versioning, *OrdID* may be used to identify ContentObjects that belong to the same final page. These will have a matching Ord*ID.*
SourceClipPath ?	path	Clip path for the PlacedObject in the coordinates of the source page.
TrimCTM ? New in JDF 1.1	matrix	The transformation matrix of the object's trim box in the *SurfaceContentsBox*. *TrimCTM* and *CTM* are identical if the TrimBox and dimension of the object in the PlacedObject are identical. Defaults to the value of *CTM.*
Type Deprecated in JDF 1.1	enumeration	Describes the kind of PlacedObject. Possible values are: *Content* *Mark*

Structure of ContentObject Subelement

ContentObject elements describe containers for page content on a surface. They are filled from the Content **RunList** of the *Imposition* process. For print applications where page count varies from Instance Document to Instance Document, imposition templates can automatically assign pages to the correct **Surface** and PlacedObject position.

Name	Data Type	Description
DocOrd ? New in JDF 1.1	integer	Reference to an index of an instance document in the content **RunList**. This references an instance document with an index module. **Layout**::*MaxDocOrd* equals *DocOrd* in an automated layout scenario. The index may either be known explicitly from a variable **Runlist** or implicitly from the index within an indexable content definition language, e.g., PPML.
Ord ? Modified in JDF 1.1	integer	A non-negative zero-based reference to an index in the content **RunList**. The index is incremented for every page of the **RunList** with *IsPage* = *true*. The *Ord* value of the first page of a **RunList**

Name	Data Type	Description
		has the value 0.
OrdExpression ?	string	Function to calculate an Ord value dynamically, using a value of *s* for signature number and *n* for total number of pages in the instance document. *Ord* or *DocOrd* and *OrdExpression* are mutually exclusive in one PlacedObject.
SetOrd ? New in JDF 1.1	integer	A non-negative zero-based reference to an index of a document set in the content **RunList**. This references an instance document with an index module. **Layout**::*MaxSetOrd* equals *SetOrd* in an automated layout scenario. The index may either be known explicitly from a variable **Runlist** or implicitly from the index within an indexable content definition language, e.g., PPML.

Using Ord to reference elements in RunLists
New in JDF 1.1A

The *Ord* attribute in ContentObject or MarkObject elements represents a reference to a *logical* element in a **RunList**. The reference is not changed by repartitioning the **RunList**. The content and marks **RunList** are referenced independently. The following examples illustrate the usage of *Ord*.

Simple Multi-File unseparated RunList

This example specifies all pages contained in "File1.pdf" and "File2.pdf". File 1 has 6 pages, file 2 has an unknown number of pages.

```
<RunList ID="L3" Class="Parameter" Status="Available" PartIDKeys="Run">
  <RunList Run="1" NPage="6" Pages="0~5">
    <LayoutElement>
       <FileSpec URL=" File://File1.pdf"/>
    </LayoutElement>
  </RunList>
  <RunList Run="2" Pages="0~-1">
    <LayoutElement>
       <FileSpec URL="File://File2.pdf"/>
    </LayoutElement>
  </RunList>
</RunList>
```

Table 7-4 Example 1 of Ord in PlacedObjects

Ord	File	Page	Ord	File	Page
0	File1	0	1	File1	1
2	File1	2	3	File1	3
4	File1	4	5	File1	5
6	File2	0	7	File2	1
8	File2	2	(n)	File2	(n-6)

Simple Multi-File separated RunList

This example specifies 2 pages contained in Presep.pdf and following that, pages 1, 3, and 5 of each preseparated file.

```
<RunList ID="Link0003" Class="Parameter" Status="Available" PartIDKeys="Run Separation">
  <RunList Run="1" SkipPage="3" NPage="2">
    <LayoutElement>
       <FileSpec URL="File://Presep.pdf"/>
    </LayoutElement>
    <RunList Separation="Cyan" FirstPage="0" IsPage="false"/>
    <RunList Separation="Magenta" FirstPage="1" IsPage="false"/>
    <RunList Separation="Yellow" FirstPage="2" IsPage="false"/>
    <RunList Separation="Black" FirstPage="3" IsPage="false"/>
```

```
    </RunList>
    <RunList Run="2" Pages="1 3 5" IsPage="true">
      <RunList Separation="Cyan" IsPage="false">
        <LayoutElement>
          <FileSpec URL="File://Cyan2.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Magenta" IsPage="false">
        <LayoutElement>
          <FileSpec URL="File://Magenta2.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Yellow" IsPage="false">
        <LayoutElement>
          <FileSpec URL="File://Yellow2.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Black" IsPage="false">
        <LayoutElement>
          <FileSpec URL="File://Black2.pdf"/>
        </LayoutElement>
      </RunList>
    </RunList>
  </RunList>
</RunList>
```

Table 7-5 Example 2 of Ord in PlacedObjects

Ord	File	Page	Separation	Ord	File	Page	Separation
0	PreSep	0	Cyan	0	Presep	1	Magenta
0	PreSep	2	Yellow	0	Presep	3	Black
1	PreSep	4	Cyan	1	Presep	5	Magenta
1	PreSep	6	Yellow	1	Presep	7	Black
2	Cyan2	1	Cyan	2	Magenta2	1	Magenta
2	Yellow2	1	Yellow	2	Black2	1	Black
3	Cyan2	3	Cyan	3	Magenta2	3	Magenta
3	Yellow2	3	Yellow	3	Black2	3	Black
4	Cyan2	5	Cyan	4	Magenta2	5	Magenta
4	Yellow2	5	Yellow	4	Black2	5	Black

Using Expressions in the OrdExpression Attribute

Expressions can use the operators $+$, $-$, $*$, $/$,$\%$ and parentheses, operating on integers and two variables: s for signature number (starting at 0) and n for number of pages to be imposed in one document. Signature number denotes the number of times that a complete set of placed objects has been filled with content from the run list. The operators have the same meaning as in the C programming language. Expressions are evaluated with normal "C" operator precedence. Multiplication must be expressed by explicitly including the $*$ operator, i.e., use "2*s", not "2 s". Remainders are discarded.

OrdExpression Examples

a.) Saddlestitched booklet for variable page length documents
The following describes the OrdExpressions for a booklet with varying page lengths. The example page assignments are for a book of 13-16 pages.

Front:
OrdExpression = "2*s"	0	2	4	6
OrdExpression = "4*((n+3)/4) –(s*2)-1"	15	13	11	9

Back:
OrdExpression = "2*s+1"	1	3	5	7
OrdExpression = "4*((n+3)/4) –(s*2)-2"	14	12	10	8

DocOrd Usage Examples

b.) Two-sided business cards 4/sheet
The following describes the Ord + DocOrd usage for a 4-up step + repeat business card
MaxDocOrd=4

Front:
Ord=0   DocOrd=0
Ord=0   DocOrd=1
Ord=0   DocOrd=2
Ord=0   DocOrd=3
Back:
Front:
Ord=1   DocOrd=0
Ord=1   DocOrd=1
Ord=1   DocOrd=2
Ord=1   DocOrd=3

Structure of MarkObject Elements

MarkObject elements describe containers for page marks on a surface. They are filled from the Marks **RunList** of the *Imposition* process. An individual MarkObject represents the content data of the Marks. The content data in individual MarkObjects may contain multiple logical marks: CIELABMeasuringField, ColorControlStrip, Cut-Mark, DensityMeasuringField, IdentificationField, RegisterMark, and ScavengerArea.

Name	Data Type	Description
LayoutElementPageNum ? New in JDF 1.1	integer	Page number to use from the PDL file described by the *LayoutElement* attribute. Default = 0
Ord ? Modified in JDF 1.1A	integer	A non-negative reference to an index in the marks **RunList**. The index is incremented for every page of the **RunList** with *IsPage* = *true*. The first page of a **RunList** has the value 0.
CIELABMeasuringField*	refelement	Specific information about this kind of mark object.
ColorControlStrip * Modified in JDF 1.1	refelement	Specific information about this kind of mark object.
CutMark * Modified in JDF 1.1	refelement	Specific information about this kind of mark object.
DensityMeasuringField * Modified in JDF 1.1	refelement	Specific information about this kind of mark object.
DeviceMark ? New in JDF 1.1	refelement	If neither *Ord* nor LayoutElement are specified, it is assumed that the device can independently generate the mark. DeviceMark defines a set of formatting parameters for the mark.
DynamicField *	refelement	Definition of text replacement for a MarkObject.
IdentificationField *	refelement	Specific information about this kind of mark object.
JobField * New in JDF 1.1	refelement	Specific information about this kind of mark object.
LayoutElement ?	refelement	PDL description of the mark. LayoutElement and *Ord* are mutually exclusive within one MarkObject.
RegisterMark* Modified in JDF 1.1	refelement	Specific information about this kind of mark object.
ScavengerArea * New in JDF 1.1	refelement	Specific information about this kind of mark object

Structure of the DeviceMark Subelement

New in JDF 1.1

Name	Data Type	Description
Font ?	NMTOKEN	The name of the font that should be used for the DeviceMark. Values include *Courier* *Helvetica* *Helvetica-Condensed* *Times-Roman* If not specified, the result is device dependent.
FontSize ?	integer	The size of the font that should be used for the DeviceMark, in points ≥ 0.
MarkOrientation ?	enumeration	Description of the preferred DeviceMark orientation One of: *Vertical* *Horizontal* If not specified, the result is device dependent.

Structure of DynamicField Subelement

Name	Data Type	Description
Format	string	Format string in C printf format that defines the replacement.
InputField ? Deprecated in JDF 1.1	string	String that must be replaced by the DynamicInput element in the Contents **RunList** referenced by *Ord* or *OrdExpression*.
Ord ?	integer	Reference to an index in the Contents **RunList** that contains DynamicInput elements. Only one of *Ord* or *OrdExpression* may be specified.
OrdExpression ?	string	Expression to calculate the reference to an index in the Contents **RunList** that contains DynamicInput fields. For details, see the definition of *OrdExpression* in the description of the PlacedObject element. Only one of *Ord* or *OrdExpression* may be specified.
ReplaceField ?	string	String that must be replaced by the instantiated text expression as defined by the *Format* and *Template* attributes in the file referenced by *Ord*, *OrdExpression* . If *ReplaceField* is not specified, the Device that processes the DynamicField must format the DynamicField.
Template	string	Template to define a sequence of variables consumed by *Format*. A list of predefined values is found in the description of the **FileSpec** resource. In addition, the *Name* attribute of DynamicInput elements of a **RunList** define further variables.
DeviceMark ? New in JDF 1.1	refelement	DeviceMark defines the formatting parameters for the mark. If not specified, the DeviceMark settings defined in **LayoutPreparationParams** or in the **Layout** tree are assumed.

DynamicField Subelement Properties

DynamicField provides a description of dynamic text replacements for MarkObjects. This element should be used for production purposes, such as defining bar codes for variable data printing. DynamicField elements are not intended as a placeholders for actual content such as addresses. Rather, they are marks with dynamic data such as time stamps and database information. Dynamic objects are MarkObjects with optional additional DynamicField elements that define text replacement.

Example usage of a DynamicField Element:
```
<!—The RunList entry: -->
```

```
<RunList … >
  <DynamicInput Name="i1">Joe</DynamicInput>
  <DynamicInput Name="i2">John</DynamicInput>
  <LayoutElement Type="Graphics">
    <FileSpec URL="File://Variable.pdf"/>
</LayoutElement>
</RunList>

…

<!—The MarkObject in the Layout hierarchy: -->
<MarkObject CTM=… (…)>
  <LayoutElement Type="Graphics">
    <FileSpec URL="File://MyReplace.pdf"/>
  </LayoutElement >
  <DynamicField ReplaceField="___xxx___"
    Format="Replacement Text for %s and %s go in here at %s on %s"
    Template="i1,i2,Time,Date" Ord="0"/>
</MarkObject>
```

In the example above, the text "___xxx___" in the file MyReplace.pdf would be replaced by the sentence "Replacement Text for Joe and John go in here at 14:00 on Mar-31-2000".

MyReplace.pdf is placed at the position defined by the CTM of the MarkedObject and Variable.pdf is placed at the position defined by the CTM of the PlacedObject.

7.2.134   ThreadSealingParams

New in JDF 1.1

This resource provides the parameters for the **ThreadSealing** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ThreadSealing*
Output of processes: -

Resource Structure

Name	Data Type	Description
BlindStitch ?	boolean	If *true*, a blind stitch after last stitch is required. Default = *false*
ThreadMaterial ?	enumeration	Thread material. Possible values are: *Cotton* *Nylon* *Polyester*
ThreadPositions	NumberList	Array containing the Y-coordinate of the center positions of the thread.
ThreadLength	double	Length of one thread.
ThreadStitchWidth	double	Width of one stitch.
SealingTemperature ?	integer	Temperature needed for sealing thread and sheets together in degrees centigrade.

7.2.135   ThreadSewingParams

This resource provides the parameters for the **ThreadSewing** process. It may also specify a gluing application, which would be used principally between the first and the second or the last and the last sheet but one. A gluing application might also be necessary if different types of paper are used.

The process coordinate system is defined as follows: The Y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The X-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge, i.e., the product front edge.
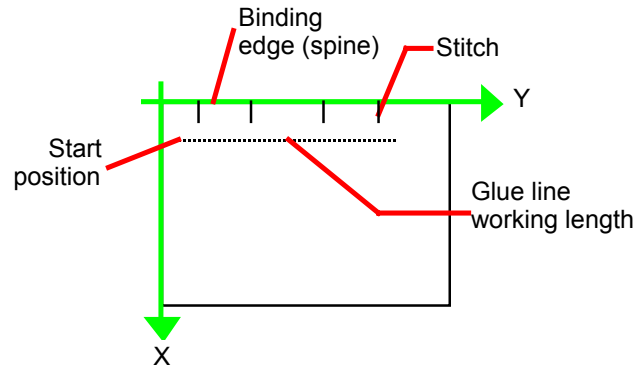
Figure 7.25 Parameters and coordinate system used for thread sewing
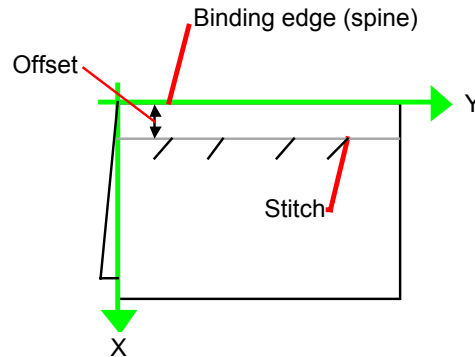
Figure 7.26 Parameters and coordinate system used for side sewing

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *ThreadSewing*
Output of processes: -

Resource Structure

Name	Data Type	Description
BlindStitch ?	boolean	If *true*, a blind stitch after last stitch is required. Default = *false*
CastingMaterial ?	enumeration	Casting material of the thread being used. Possible values are: *Cotton* *Nylon* *Polyester*

Name	Data Type	Description
CoreMaterial ?	enumeration	Core material of the thread being used. This attribute must be used to define the thread material if there is no casting. Possible values are: *Cotton* *Nylon* *Polyester*
GlueLineRefSheets	IntegerList	This entry is only required if GlueLine is defined. It contains the indices of the loose parts of the input component after which gluing should be applied. The index starts with 0.
Offset ? New in JDF 1.1	double	Specifies the distance between the stitch and the binding edge. Used only for side stitching. Default = 0
NumberOfNeedles	integer	Specifies the number of needles to be used. Default = equipment-specific setting.
NeedlePositions ?	NumberList	Array containing the Y-coordinate of the needle positions. The number of entries must match the number given in *NumberOfNeedles*. Default = equipment-specific setting.
Sealing ?	boolean	If *true*, thermo-sealing is required. Default *false*
SewingPattern ?	enumeration	Sewing pattern. Possible values are: *Normal* – The default. *Staggered* *CombinedStaggered* *Side* – Side sewing.
ThreadThickness ?	double	Thread thickness.
ThreadBrand ?	string	Thread brand.
GlueLine *	element	Gluing parameters.

7.2.136   Tile

Each **Tile** resource defines how content from a **Surface** resource will be imaged onto a piece of media that is smaller than the designated surface. Tiling occurs in some production environments when pages are imaged on to an intermediate medium, and the resulting image of the surface is larger than the media. In this case, instructions are needed to determine how the intermediate media (tiles) will be assembled to achieve the desired output, e.g., a single plate for the surface. For example, a device might require that four pieces of film be assembled to create the image for the plate.

In general, a **Tile** resource will be partitioned (see Section 3.9.2 Description of Partitionable Resources) by *TileID*. Individual tiles are selected and matched by specifying the appropriate *TileID* attribute, which is described in Table 3-25 Contents of the Part element.

Resource Properties
Resource class: Parameter
Resource referenced by: -
Example Partition: *TileID*
Input of processes: *Tiling*
Output of processes: -

Resource Structure

Name	Data Type	Description
ClipBox	rectangle	A rectangle that defines the bounding box of the **Surface** contents which will be imaged on this **Tile**. The *ClipBox* is defined in the coordinate system of the **Surface**.

Name	Data Type	Description
CTM	matrix	A coordinate transformation matrix mapping the *ClipBox* for this **Tile** to the rectangle 0 0 X Y, where X and Y are the extents of the media that the **Tile** will be imaged onto.
MediaSource?	refelement	Describes the media to be used.

7.2.137　　　Tool

New in JDF 1.1

A **Tool** resource defines a generic tool that is customized for needed for a given job, e.g., an embossing stamp. The manufacturing process for the tool is not described within JDF.

Resource Properties
Resource class:　　　　Handling
Resource referenced by: -
Example Partition:　　 -
Input of processes:　　 *Embossing*, *ShapeCutting*
Output of processes:　　-

Resource Structure

Name	Data Type	Description
ToolAmount ?	Integer	Number of identical instances of the tool that the tool contains, e.g., the number of cut forms in a die cutting die.
ToolID	string	ID of the tool. This is a unique name within the workflow.
ToolType ?	NMTOKEN	Type of the tool. Possible values include: *EmbossingCalendar* *EmbossingStamp* *CutDie*

7.2.138　　　TransferCurve

TransferCurve elements specify the characteristic curve of transfer of densities between systems. For more details on transfer curves and their usage, refer to the CIP3 PPF specification at:
http://www.cip4.org/documents/technical_info/cip3v3_0.pdf

Resource Properties
Resource class:　　　　Parameter
Resource referenced by: Color, TransferCurvePool
Example Partition:　　 *RibbonName, SheetName, Side, WebName*
Input of processes:
Output of processes:　　-

Resource Structure

Name	Data Type	Description
Curve	TransferFunction	The density mapping curve for the separation defined by Separation.
Separation ?	string	The name of the separation. If *Separation* = *All*, this curve should be applied to all separations that are not explicitly defined.

7.2.139   TransferCurvePool

A transfer curve pool is a collection of TransferCurveSet elements that each contains information about a Trans-
ferCurve. Multiple TransferCurvesSets may exist at one time. For example, one may exist for the laser calibra-
tion of the imagesetter, one for the **ContactCopying** process and one for the printing process. Each
TransferCurveSet consists of one or more TransferCurve elements. A TransferCurve element should be applied
to the appropriate correlative *Separation*, or to all *Separation*s when *Separation = All*. The TransferCurveSets
should be concatenated in the following order:

> Film -> Plate -> Press -> Paper.
> and
> Proof.

Resource Properties

Resource class:	Parameter
Resource referenced by:	TransferFunctionControl
Example Partition:	-
Input of processes:	*InkZoneCalculation*
Output of processes:	-

Resource Structure

Name	Data Type	Description
TransferCurveSet *	element	The set of transfer curves.

Structure of TransferCurveSet Subelement

TransferCurveSet elements describe both the characteristic curve of transfer and the relation between the various
process coordinate systems.

TransferCurveSet

Name	Data Type	Description
CTM ? New in JDF 1.1	matrix	Defines the transformation of the coordinate system in the device as defined by *Name*. Default = identity matrix: "1 0 0 1 0 0"
Name Modified in JDF 1.1	NMTOKEN	The name of the TransferCurveSet. Possible values include: *Film* – The transformation from the **Layout** system to the *Film*. In a CtP envi-ronment, this defaults to the identity matrix and the identity TransferCurve. *Plate* – The transformation from the *Film* system to the *Plate*. *Paper* – The transformation from the *Press* system to the *Paper*. *Press* – The transformation from the *Plate* system to the *Press* *Proof* – The transformation from the **Layout** system to the *Proof*.
TransferCurve * Modified in JDF 1.1	refelement	List of TransferCurve entries.

7.2.140   TransferFunctionControl

Resource class:	Parameter
Resource referenced by:	SeparationControlParams
Example Partition:	-
Input of processes:	-
Output of processes:	-

Resource Structure

Name	Data Type	Description
TransferFunctionSource	enumeration	Identifies the source of transfer curves which should be applied during separation.
		Document – Use the transfer curves provided in the document.
		Device – Use transfer functions provided by the output device. When **Separation** is being performed pre-RIP, this may mean that no transfer curves will be applied.
		Custom – Use the transfer curves provided in the TransferCurve-Pool element of this element.
TransferCurvePool *?*	refelement	Provides a set of transfer curves to be used by the process.

7.2.141 TrappingDetails

This resource identifies the root of the hierarchy of resources. This hierarchy controls the **Trapping** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *SoftProofing*
Output of processes: -

Resource Structure

Name	Data Type	Description
DefaultTrapping ?	boolean	If *true*, pages that have no defined TrapRegions are trapped using the set of TrappingParams. The BleedBox is used for the TrapZone. If *false*, only pages that have TrapRegions are trapped. Default = *false*
IgnoreFileParams ?	boolean	If *true*, any trapping controls provided within any source files used by this process are ignored. If *false*, trapping controls embedded in the source files are honored. Default = *true*
Trapping ?	boolean	If *true*, trapping is enabled. If omitted, the default setting for the device is used.
TrappingOrder ?	element	Trapping processes will trap colorants as if they are laid down on the media in the order specified in *TrappingOrder*. The colorant order may affect which colors to spread, especially when opaque inks are used. Default = system specified
TrappingType ?	integer	Identifies the trapping method to be used by the trapping process. The number identifies the minor (last three digits) and major (any digits prior to the last three) version of the trapping type requested. Default = system specified
TrappingParams ?	refelement	A **TrappingParams** resource that is used to define the default trapping parameters when *DefaultTrapping* = *true*.
ObjectResolution * New in JDF 1.1	refelement	Elements which define the resolutions to trap the contents at. More than one element may be used to specify different resolutions for different SourceObject types. Default = device specific
TrapRegion *	refelement	A set of **TrapRegion** resources that identify the pages to be trapped, the geometry of the areas to trap on each page, and the trapping settings to use for each area.

Structure of the TrappingOrder Subelement

Name	Data Type	Description
SeparationSpec*	element	An array of colorant names.

7.2.142   TrappingParams

This resource provides a set of controls that are used to generate traps. The values of the parameters are chosen based on the customer's trapping strategy, and depend largely on the content of the pages to be trapped and the characteristics of the output device (press). The attributes of this resource that are optional in the sense that each implementation decides a default value for them.

Resource Properties

Resource class: Parameter
Resource referenced by: TrapRegion, TrappingDetails
Example Partition: *DocIndex, RunIndex, RunTag, SheetName, Side, SignatureName*
Input of processes: *-*
Output of processes: *-*

Resource Structure

Name	Data Type	Description
BlackColorLimit ?	number	A number between 0 and 1 that specifies the lowest color value required for trapping a colorant according to the black trapping rule. This entry uses the subtractive notion of color, where 0 is white, or no colorant, and 1 is full colorant. Default = system specified
BlackDensityLimit ?	number	A positive number that specifies the lowest neutral density of a colorant for trapping according to the black trapping rule. Default = system specified
BlackWidth ?	number	A positive number that specifies the trap width for trapping according to the black trapping rule. *BlackWidth* is specified in *TrapWidth* units; a value of 1 means that the black trap width is one *TrapWidth* wide. The resulting black trap width is subject to the same device limits as *TrapWidth*. Default = system specified
Enabled ?	boolean	If *true*, trapping is enabled for zones that are defined with this parameter set. Default = system specified
HalftoneName ?	string	A name that identifies a halftone object to be used when marking traps. The name is the value of the *ResourceName* attribute of some **PDLResourceAlias** resource. If absent, the halftone in effect just before traps are marked will be used, which may cause unexpected results.
ImageInternalTrapping ?	boolean	If *true*, the planes of color images are trapped against each other. If *false*, the planes of color images are not trapped against each other. Default = system specified
ImageResolution ?	integer	A positive integer indicating the minimum resolution, in dots per inch, for downsampled images. Images can be downsampled by a power of 2 before traps are calculated. The downsampled image is used only for calculating traps, while the original image is used when printing the image. Default = system specified

Name	Data Type	Description
ImageMaskTrapping ?	boolean	Controls trapping when the *TrapZone* contains a stencil mask.
		A stencil mask is a monochrome image in which each sample is represented by a single bit. The stencil mask is used to paint in the current color: image samples with a value of 1 are marked, samples with a value of 0 are not marked.
		When *false*, none of the objects covered by the clipped bounding box of the stencil mask are trapped. No traps are generated between the stencil mask and objects that the stencil mask overlays. No traps are generated between objects that overlay the stencil mask and the stencil mask. For all other objects, normal trapping rules are followed. Two objects on top of the stencil mask that overlap each other may generate a trap, regardless of the value of this parameter. When *true*, objects are trapped to the stencil mask, and to each other. Default = system specified
ImageToImageTrapping ?	boolean	If *true*, traps are generated along a boundary between images. If *false*, this kind of trapping is not implemented.
		Default = system specified
ImageToObjectTrapping ?	boolean	If *true*, images are trapped to other objects. If *false*, this kind of trapping is not implemented. Default = system specified
ImageTrapPlacement ?	enumeration	Controls the placement of traps for images. Possible values are:
		Center – Trap is centered on the edge between the image and the adjacent object.
		Choke – Trap is placed in the image.
		Normal – Trap is based on the colors of the areas.
		Spread –Trap is placed in the adjacent object.
		Default = system specified
MinimumBlackWidth ?	number	Specifies the minimum width, in points, of a trap that uses black ink. Allowable values are those greater than or equal to zero.
		Default = 0
SlidingTrapLimit ?	number	A number between 0 and 1. Specifies when to slide traps towards a center position. If the neutral density of the lighter area is greater than the neutral density of the darker area multiplied by the *SlidingTrapLimit*, then the trap slides. This applies to vignettes and non-vignettes. No slide occurs at 1. Default= system specified
StepLimit ?	number	A number between 0 and 1. Specifies the smallest step required in the color value of a colorant to trigger trapping at a given boundary.
		If the higher color value at the boundary exceeds the lower value by an amount that is equal or greater than the larger of 0.05 or *StepLimit* times the lower value (low + max (*StepLimit* * low, 0.05)), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. This entry is used when not specified explicitly by a *ColorantZoneDetails* subelement for a colorant. Default = system specified

Name	Data Type	Description
TrapColorScaling ?	number	A number between 0 and 1. Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. 1 means the trap has the combined color values of the darker and the lighter area. 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. This entry is used when not specified explicitly by a *ColorantZoneDetails* subelement for a colorant. Default = system specified
TrapEndStyle ?	NMTOKEN	Instructs the trap engine how to form the end of a trap that touches another object. Possible values include: *Miter* *Overlap* Other values may be added later as a result of customer requests. Default = *Miter*
TrapJoinStyle ?	NMTOKEN	Specifies the style of the connection between the ends of two traps created by consecutive segments along a path. Possible values include: *Bevel* *Miter* *Round* Default = *Miter*
TrapWidth ?	number	A positive number. Specifies the trap width in points. Also defines the unit used in trap width specifications for certain types or objects, such as *BlackWidth*. Default = system specified
ColorantZoneDetails *	element	ColorantZoneDetails subelements. Entries in this dictionary reflect the results of any named colorant aliasing specified. Each entry defines parameters specific for one named colorant. If the colorant named is neither listed in the ColorantParams array, nor implied by the *ProcessColorModel*, for the *ColorantControl* object in effect when these *TrappingParams* are applied, the entry is not used for trapping.

Structure of ColorantZoneDetails Subelement

Name	Data Type	Description
Colorant	string	The colorant name that occurs in the SeparationSpec::*Name* of the ColorantParams array of the *ColorantControl* object used by the process.
StepLimit ?	number	A number between 0 and 1. Specifies the smallest step required in the color value of a colorant to trigger trapping at a given boundary. If the higher color value at the boundary exceeds the lower value by an amount that is equal or greater than the larger of 0.05 or *StepLimit* times the lower value (low + max (*StepLimit* * low, 0.05)), then the edge is a candidate for trapping. The value 0.05 is set to avoid trapping light areas in vignettes. If omitted, the *StepLimit* attribute in the **TrappingParams** resource is used.

Name	Data Type	Description
TrapColorScaling ?	number	A number between 0 and 1. Specifies a scaling of the amount of color applied in traps towards the neutral density of the dark area. 1 means the trap has the combined color values of the darker and the lighter area. 0 means the trap colors are reduced so that the trap has the neutral density of the darker area. If omitted, the *TrapColorScaling* attribute in the **TrappingParams** resource is used.

7.2.143   TrapRegion

This resource identifies a set of pages to be trapped, an area of the pages to trap, and the parameters to use.

Resource Properties

Resource class: Parameter
Resource referenced by: TrappingDetails
Example Partition: -
Input of processes: *-*
Output of processes: -

Resource Structure

Name	Data Type	Description
TrapZone ?	path	Each element within *TrapZone* is one subpath of a complex path. The *TrapZone* is the area that results when the paths are filled using the non-zero winding rule. When absent, the *MediaBox* array for the **RunList** defines the *TrapZone*.
Pages	IntegerRangeL-ist	Identifies a set of pages from the **RunList** to trap using the specified geometry and trapping style.
TrappingParams ?	refelement	The set of TrappingParams which will be used when trapping in this region. Default = use system specified trapping parameters.

7.2.144   TrimmingParams

This resource provides the parameters for the *Trimming* process.

   The process coordinate system is defined as follows — The y-axis is aligned with the binding edge. It increases from the registered edge to the edge opposite to the registered edge. The x-axis is aligned with the registered edge. It increases from the binding edge to the edge opposite to the binding edge, i.e. the product front edge.
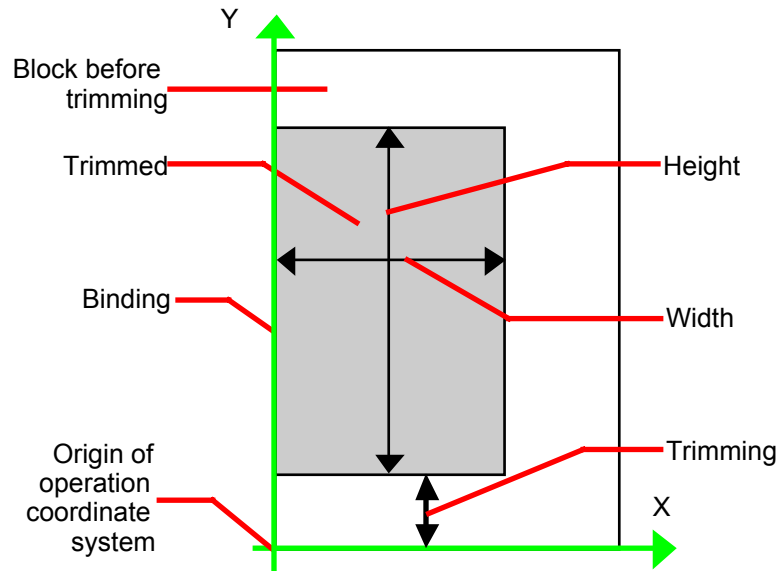
Figure 7.27 Parameters and coordinate system used for trimming

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Trimming*
Output of processes: -

Resource Structure

Name	Data Type	Description
Height ?	double	Height of the trimmed product. If not specified, the system specified *Height* is assumed.
TrimmingOffset ?	double	Amount to be cut at bottom side. If not specified, the system specified *TrimmingOffset* is assumed.
TrimmingType New in JDF 1.1	enumeration	Trimming operation to perform: Possible values are: *Detailed* – Cut the amount specified by *Height, Width* and *TrimmingOffset.* *SystemSpecified* – Cut the amount specified by the system.
Width ?	double	Width of the trimmed product. If not specified, the system specified *Width* is assumed.

7.2.145   VerificationParams

This resource provides the parameters of a *Verification* process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: *Verification*
Output of processes: -

Resource Structure

Name	Data Type	Description
FieldRange ?	Integer-RangeList	Zero based range list of integers that determines which characters of the data in **IdentificationField** should be applied to the field formatting strings. Defaults = 0~-1, which means first-to-last.
InsertError ?	string	Database insertion statement in C printf format defining how information read from the **IdentificationField** resource of the **Verification** process should be stored in case of verification errors. The database is defined by the **DBSelection** resource of the **Verification** process. This field must be specified if a database is selected.
InsertOK ?	string	Database insertion statement in C printf format defining how information extracted from the **IdentificationField** should be stored in case of verification success. The database is defined by the **DBSelection** resource of the verification node. This field must be specified if a database is selected.
Tolerance ?	double	Ratio of tolerated verification failures to the total number of tests. 0 = none allowed, 1.0 = all.

Usage of FieldRange and Format Strings.

A database field name can be calculated from the characters of the **IdentificationField** using standard C *printf* notation and the *FieldRange* attribute. Each range that is defined in *FieldRange* is passed to *printf* as one string that is applied to the format. The order is maintained. Note that SQL was chosen for illustrative purposes only. The mechanism is defined for any database interface.

Example

IdentificationField string: 1234:John Doe
FieldRange: 5~-1 0~3
FieldOK: Insert true into Va where Name = '%s' and ID = %s
Resulting string: Insert true into Va where Name = "John Doe" and ID = 1234

7.2.146   WireCombBindingParams

This resource describes the details of the ***WireCombBinding*** process.

Resource Properties

Resource class: Parameter
Resource referenced by: -
Example Partition: -
Input of processes: ***WireCombBinding***
Output of processes: -

Resource Structure

Name	Data Type	Description
Brand ?	string	The name of the comb manufacturer, e.g., *Wire-O®,* and the name of the specific item. Default =system specified.
Color ?	NamedColor	Determines the color of the comb. Default =system specified.
Diameter ?	double	The comb diameter is determined by the height of the block of sheets to be bound. Default =system specified.
Distance ?	double	The distance between the "teeth" and the distance between the holes of the prepunched sheets must be the same. Default =system specified.

Name	Data Type	Description
FlipBackCover ? New in JDF 1.1	boolean	The spine is typically hidden between the last page of the Component and the back cover. Flip the back cover after the wire was "closed" or keep it open. The latter makes sense, if further processing is required, e.g., inserting a CD, before closing the book. Default = *false*.
^Material ?	enumeration	The material used for forming the wire comb binding. Possible values are: *LaqueredSteel* *TinnedSteel* *ZincsSteel* Default =system specified.
Shape ?	enumeration	The shape of the wire comb binding. Possible values are: *Single* – Each "tooth" is made with one wire. The default. *Twin* – The shape of each "tooth" is made with a double wire
Thickness ?	double	The thickness of the comb material. Default =system specified.

7.2.147   WrappingParams

New in JDF 1.1

WrappingParams defines the details of *Wrapping*. Details of the material used for *Wrapping* can be found in the **Media** resource that is also an input of the *Wrapping* process.

Resource Properties
Resource class: Parameter
Resource referenced by:
Example Partition: -
Input of processes: *Wrapping*
Output of processes: -

Resource Structure

Name	Data Type	Description
WrappingKind	enumeration	*LooseWrap* – The wrap is loose around the component *ShrinkWrap* – The wrap is shrinked around the component Default = equipment-specific setting

7.3 Device Capability Definitions

New in JDF 1.1

The elements in this section are used to specify capabilities of devices. Note that only attributes and elements that are explicitly described within the device capabilities structure are supported by the device. For more details on using device capabilities, refer to section 4.8 Describing Device Capabilities with JDF.

7.3.1 Structure of the DeviceCap Subelement

New in JDF 1.1

The DeviceCap element describes the JDF Nodes and Resources that a device is capable of processing. Elements that are derived from the abstract State elements are used to describe ranges and lists of ranges of allowed parameters.

Name	Data Type	Description
CombinedMethod ?	enumeration	Specifies how the processes specified in *Types* may be specified.

Name	Data Type	Description
		One of:
		Combined – The list of processes in **Types** must be specified as a *Combined* process.
		ProcessGroup – The list of processes in **Types** must be specified as a *ProcessGroup* of individual processes.
		CombinedProcessGroup – The list of processes in **Types** may be specified either as a *Combined* process or as a *ProcessGroup* of individual processes.
		None – No support for *Combined* or *ProcessGroup*. Only one individual process type defined in **Types** is supported. The default.
GenericAttributes ?	NMTOKENS	List of generic attributes that are supported and unrestricted by the device implementation. Note that descriptions of attributes that appear in **State** elements (see the following section 7.3.5) overwrite the description in **GenericAttributes**.
OptionalCombined-Types ?	NMTOKENS	List of optional JDF Node types. The entries of the list must be a subset of **Types.**
		For example, a RIP with optional in-RIP trapping would specify **OptionalCombinedTypes** = *Trapping* if **Types** = *Trapping Interpreting Rendering*". Default = *none*, i.e. no optional Node type dependencies exist..
TypeOrder ?	enumeration	Ordering restriction for *Combined* or *ProcessGroup* nodes.
		Fixed – The order of process types specified in the **Types** attribute is ordered and each type can be specified only once, e.g., **Cutting, Folding.** Order does matter. The default. *Unordered* – The order of process types specified in the **Types** attribute is unordered, and each type can be specified only once, e.g., **DigitalPrinting, Screening, Trapping** Order does not matter.
		Unrestricted – The order of process types specified in the **Types** attribute is unordered, and each type can be specified multiply, e.g., **Cutting, Folding.** The device can do both processes, in any order and multiple times.
Type	NMTOKEN	JDF **Type** attribute of the supported process. Extension types may be specified by stating the namespace prefix in the value.
Types ?	NMTOKENS	If **Type** = *Combined,* or **Type** = *ProcessGroup* this attribute represents the list of combined processes. If any of the Services are in a namespace other than JDF, the namespace prefix should be included in this list. For details, see Section 3.2.3 Combined Process Nodes
DevCaps *	element	List of definitions of the accepted resources. The **DevCaps** elements are combined with a logical AND, i.e. A JDF must fulfill all restrictions defined by the set of **DevCaps**. Only resources that are specified within this list are honored by the device.
Performance *	element	Specification of a devices performance capabilities.

7.3.2 Structure of the Performance Subelement

New in JDF 1.1

The Performance element describes speed as the capability to consume or produce a JDF Resource.

Name	Data Type	Description
AverageAmount	number	Average amount produced/consumed per hour assuming an average job.
AverageCleanup ?	duration	Average time needed to clean the device after a job. Default = 0M
AverageSetup ?	duration	Average time needed to setup the device before a job. Default = 0M
MaxAmount ?	number	Maximum amount produced/consumed per hour assuming an ideal job. Default = 0 which translates to the value of *AverageAmount*.
MaxCleanup ?	duration	Maximum time needed to clean the device after a job assuming a worst case job. Default = 0M which specifies the value defined in *AverageCleanup.*
MaxSetup ?	duration	Maximum time needed to setup the device before a job assuming a worst case job. Default = 0M which specifies the value defined in *AverageSetup.*
MinAmount ?	number	Minimum amount produced/consumed per assuming a worst case job. Default = 0 which translates to the value of *AverageAmount*.
MinCleanup ?	duration	Minimum time needed to clean the device after a job assuming an ideal job. Default = 0M which specifies the value defined in *AverageCleanup.*
MinSetup ?	duration	Minimum time needed to setup the device before a job assuming an ideal job. Default = 0M which specifies the value defined in *AverageSetup.*
Name	NMTOKEN	Name of the input resource type that is processed by the device, e.g., "Media", "Ink", "RunList".
Unit ?	NMTOKEN	Unit of measure of resource consumption per hour. Defaults to the resources generic units as defined in **Table 1-3 Units used in JDF**

7.3.3 Structure of the DevCaps Subelement

New in JDF 1.1

The DevCaps element describes the valid parameter space of a JDF Resource, message or resource link that is consumed, honored, or produced by a device. Note: DevCaps not only describes the structure of the individual resources and resource links but also of the NodeInfo element within a JDF node. The DevCaps element may be used to model product intent resources as well as process definition resources.

Name	Data Type	Description
DevNS ?	URI	Namespace of the resource or message that is described. Default = the JDF namespace.
Name	NMTOKEN	Fully qualified name of the element that is described, *ResourceUsage* attribute or *ProcessUsage* of the respective resource within a JDF node . If *Name* = *NodeInfo*, it describes the structure of the NodeInfo information that is accepted by the device.
ResourceUpdate ?	NMTOKENS	Specifies the capability to handle partial updates defined in ResourceUpdate elements. Possible values include: *None* – ResourceUpdate is not supported. Must not be combined with any other value. The default. *JMFID* – JMF Resource messages that reference ResourceUpdates that have been previously loaded to the device are accepted. *PDLID* – References from PDL data, e.g., PPML TicketRef elements that reference ResourceUpdates that have been previously loaded to the device are accepted.

Name	Data Type	Description
Types ?	NMTOKENS	List of JDF Node types that a DevCaps applies to. Default = the *Types* attribute of the parent DeviceCap element. The value of *Types* must be a subset of *Types* in DeviceCap.
DevCap +	element	List of definitions of the accepted parameter space for resources and messages. The parameter spaces of multiple DevCap elements are combined as a superset of the individual DevCap elements. Only elements that are explicitly specified as DevCap elements within a DevCaps are supported.

7.3.4 Structure of the DevCap Subelement

New in JDF 1.1

The DevCap element describes the valid parameter space of a JDF resource, message or element that is consumed or produced by a Device. The structure of the DevCap is identical to that of the JDF resource, message, or element that it models. Individual attributes are replaced by the appropriate State elements. For more details on State elements, see Section 7.3.5. The *Name* attribute of the State element must match the attribute key that is described. If no State element exists for a given attribute, it is assumed to be unsupported. The restrictions of multiple attributes and elements are combined with a logical AND.

Subelements of resources are modeled by including nested DevCap with a *ResourceUsage* attribute equal to the subelements tag-name or *ResourceUsage* if the subelement is a **FileSpec**. Attributes of the resource link belonging to the resource, e.g., *Transformation* or the various pipe control parameters may also be restricted.

Name	Data Type	Description
DevNS ?	URI	Namespace of the element that is described by this DevCap. Default = the JDF namespace.
MaxOccurs ?	integer or "unbounded"[5]	Maximum number of occurrences of the element described by this DevCap. Default = 1.
MinOccurs ?	integer or "unbounded"	Minimum number of occurrences of the element described by this DevCap. Default = 1.
Name	NMTOKEN	Fully qualified name of the resource that is described. *ResourceUsage* attribute or *ProcessUsage* of the respective resource within a JDF node. Default = the value of *Name* of the parent DevCaps element.
DevCap *	element	Definition of the accepted parameter space for the messages or resources subelements.
State *	element	Abstract State elements that define the parameter space that is covered by device. One State element must be defined for each supported attribute or Intent Span element of the resource that is not specified DeviceCaps::*GenericAttributes*. If a resource attribute has no matching State element in DevCap, it is not supported.

7.3.5 Structure of the Abstract State Subelement

New in JDF 1.1

The following table describes the common, data type independent parameters of all State objects.

---

[5] This construct is built to be compatible with the XML schema recommendation of minOccurs, maxOccurs.

Name	Data Type	Description
DevNS ?	URI	Namespace of the attribute that is described by this **State** element. Default = the JDF namespace.
HasDefault ?	boolean	A flag that describes whether the parameter has a device default. If set, *DefaultValue* must be set. Default = *true*
Name ?	NMTOKEN	Name of the attribute that is described by this **State**. If *Name* is omitted this **State** describes the element's text, i.e., the text between the XML start and end tag.
Span ? New in JDF 1.1a	boolean	A flag that describes whether the parameter is an intent span data type. Default="*false*". For example a State element describing an XYPairSpan would have *DataType*="*XYPairState*" and *Span*="*true*".

The following types of **State** elements are defined:

Name	Data Type	Description
BooleanState	element	Describes a set of boolean values.
EnumerationState	element	Describes a set of enumeration values.
IntegerState	element	Describes a numerical range of integer values.
MatrixState	element	Describes a range of matrices. generally used to define valid orientations of Components.
NameState	element	Describes a set of NMTOKEN values.
NumberState	element	Describes a numerical range of values.
ShapeState	element	Describes a set of 3 value shape values.
StringState	element	Describes a set of string values.
XYPairState	element	Describes a set of XYPair values.

7.3.5.1 Structure of the BooleanState Subelement
New in JDF 1.1

This **State** subelement is used to describe ranges of boolean values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
AllowedValueList ? *Added in JDF 1.1A*	enumerations	A list of all legal values. Allowed list values are the booleans "*true*" and "*false*". Default = the empty list, which specifies an unrestricted range.
CurrentValue ?	boolean	Current value for the current running job set in the device. If not specified, the value is unknown.
DefaultValue ?	boolean	Expected, initial value. Must be set if *HasDefault* = true.

7.3.5.2 Structure of the EnumerationState Subelement
New in JDF 1.1

This **State** subelement is used to describe ranges of enumerative values. It inherits from the abstract **State** element described above. It is identical to the **NameState** element except for the fact that it describes a closed list of enumeration values.

Name	Data Type	Description
DefaultValue ?	enumeration	Expected, initial value. Must match the enumeration defined in the resource. Must be set if *HasDefault* = true.

Name	Data Type	Description
CurrentValue ?	enumeration	Current value for the current running job set in the device. Must match the enumeration defined in the resource. Default = unknown.
AllowedValueList ?	enumerations	A list of all potential legal values. Must match the enumeration defined in the resource. Default = the empty list, which specifies an unrestricted range.
PresentValueList ?	enumerations	A list of values that can be chosen without operator intervention. Must match the enumeration defined in the resource. If not specified, the value of *AllowedValueList* is applied.

7.3.5.3 Structure of the IntegerState Subelement
New in JDF 1.1

This State subelement is used to describe ranges of integer values. It inherits from the abstract State element described above.

Name	Data Type	Description
DefaultValue ?	integer	Expected, initial value. Must be set if *HasDefault* = true.
CurrentValue ?	integer	Current value for the current running job set in the device. Default = unknown.
AllowedValueList ?	IntegerList	A list of all legal values. Default = the empty list, which specifies an unrestricted range.
AllowedValueMax ?	integer	Inclusive maximum allowed value.
AllowedValueMin ?	integer	Inclusive minimum allowed value.
PresentValueList ?	IntegerList	A list of values that can be chosen without operator intervention. If not specified, the value of *AllowedValueList* is applied.
PresentValueMax ?	integer	Inclusive maximum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMax* is applied.
PresentValueMin ?	integer	Inclusive minimum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMin* is applied.

7.3.5.4 Structure of the MatrixState Subelement
New in JDF 1.1

This State subelement is used to describe ranges of matrix values. It inherits from the abstract State element described above. It is primarily intended to specify orientations and manipulation capabilities of physical resources, e.g. in finishing devices.

Name	Data Type	Description
DefaultValue ?	matrix	Expected, initial value. Must be set if *HasDefault* = true.
CurrentValue ?	matrix	Current value for the current running job set in the device. Default = unknown.
Value *	element	A list legal values.

Structure of the Value element

Name	Data Type	Description
AllowedValue	matrix	A legal value for a matrix variable.
PresentValue ?	matrix	A legal value for a matrix variable that can be chosen without operator intervention. If not specified, the value of *AllowedValue* is applied.

7.3.5.5   Structure of the NameState Subelement

New in JDF 1.1

This **State** subelement is used to describe ranges of NMTOKEN values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
DefaultValue ?	NMTOKEN	Expected, initial value. Must be set if *HasDefault* = true.
CurrentValue ?	NMTOKEN	Current value for the current running job set in the device. Default = unknown.
AllowedValueList ?	NMTOKENS	A list legal values. Default = the empty list, which specifies an unrestricted range.
PresentValueList ?	NMTOKENS	A list of values that can be chosen without operator intervention. If not specified, the value of *AllowedValueList* is applied.

7.3.5.6   Structure of the NumberState Subelement

New in JDF 1.1

This **State** subelement is used to describe ranges of integer values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
DefaultValue ?	number	Expected, initial value. Must be set if *HasDefault* = true.
CurrentValue ?	number	Current value for the current running job set in the device. Default = unknown.
AllowedValueList ?	NumberList	A list legal values. Defaults to the empty list, which specifies an unrestricted range.
AllowedValueMax ?	number	Inclusive maximum allowed value.
AllowedValueMin ?	number	Inclusive minimum allowed value.
PresentValueList ?	NumberList	A list of values that can be chosen without operator intervention. If not specified, the value of *AllowedValueList* is applied.
PresentValueMax ?	number	Inclusive maximum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMax* is applied.
PresentValueMin ?	number	Inclusive minimum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMin* is applied.

7.3.5.7   Structure of the ShapeState Subelement

New in JDF 1.1

This **State** subelement is used to describe ranges of Shape values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
DefaultValue ?	shape	Expected, initial value. Must be set if *HasDefault* = true.
CurrentValue ?	shape	Current value for the current running job set in the device. Default = unknown.
AllowedValueList ?	NumberList	A list of values that can be chosen. The NumberList must have a number of entries that is a multiple of three and three adjacent entries define one shape.
AllowedValueMax ?	shape	Inclusive maximum allowed value.
AllowedValueMin ?	shape	Inclusive minimum allowed value.

Name	Data Type	Description
PresentValueList ?	NumberList	A list of values that can be chosen without operator intervention. The NumberList must have a number of entries that is a multiple of three and three adjacent entries define one shape. If not specified, the value of **AllowedValueList** is applied.
PresentValueMax ?	shape	Inclusive maximum allowed value that can be chosen without operator intervention. If not specified, the value of **AllowedValueMax** is applied.
PresentValueMin ?	shape	Inclusive minimum allowed value that can be chosen without operator intervention. If not specified, the value of **AllowedValueMin** is applied.

7.3.5.8 Structure of the StringState Subelement
New in JDF 1.1

This **State** subelement is used to describe ranges of string values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
DefaultValue ?	string	Expected, initial value. Must be set if **HasDefault** = true.
CurrentValue ?	string	Current value for the current running job set in the device. Default = unknown.
Value +	element	A list legal values.

Structure of the Value element
New in JDF 1.1

Name	Data Type	Description
AllowedValue	string	A legal value for a string variable.
PresentValue ?	string	A legal value for a string variable that can be chosen without operator intervention. If not specified, the value of **AllowedValue** is applied.

7.3.5.9 Structure of the XYPairState Subelement
New in JDF 1.1

This **State** subelement is used to describe ranges of XYPair values. It inherits from the abstract **State** element described above.

Name	Data Type	Description
DefaultValue ?	XYPair	Expected, initial value. Must be set if **HasDefault** = true.
CurrentValue ?	XYPair	Current value for the current running job set in the device. Default = unknown.
AllowedValueList ?	NumberList	A list of values that can be chosen. The NumberList must have an even number of entries and two adjacent entries define one XY-Pair.
AllowedValueMax ?	XYPair	Inclusive maximum allowed value.
AllowedValueMin ?	XYPair	Inclusive minimum allowed value.
PresentValueList ?	NumberList	A list of values that can be chosen without operator intervention. The NumberList must have an even number of entries and two adjacent entries define one XYPair. If not specified, the value of **AllowedValueList** is applied.

Name	Data Type	Description
PresentValueMax ?	XYPair	Inclusive maximum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMax* is applied.
PresentValueMin ?	XYPair	Inclusive minimum allowed value that can be chosen without operator intervention. If not specified, the value of *AllowedValueMin* is applied.

7.3.6 Examples of Device Capabilities

New in JDF 1.1   Modified in JDF 1.1A

Device Description of a Scanner

Simple example of a Scanner description in a Device resource. The JMF based hand shaking is also illustrated. NodeInfo, ExposedMedia, and ScanParams are restricted.

Device Query:

```
<JMF xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1" TimeStamp="2002-04-
05T16:45:43+02:00" SenderID="Controller">
  <Query ID="DeviceQuery" Type="KnownDevices">
    <DeviceFilter DeviceDetails="Capability"/>
  </Query>
</JMF>
```

Device Response:

```
<?xml version='1.0' encoding='utf-8' ?>
<JMF xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1" TimeStamp="2002-06-
05T16:45:43+02:00" SenderID="Scanner">
  <Response ID="xyz" refID="DeviceQuery" Type="KnownDevices">
    <DeviceList>
      <DeviceInfo>
        <Device ID="IDXYZ" Class="Implementation"  Status="Available"
                DeviceID="Joe the Drum" ModelName="Bongo">
          <DeviceCap Type="Scanning"
            GenericAttributes="ID Class SettingsPolicy BestEffortExceptions
                               OperatorInterventionExceptions MustHonorExceptions
                               PartIDKeys DocIndex rRefs">
          <!— the scanner takes a minute to set up and scans an average of 2 sheets a min. -->
          <Performance Name="ExposedMedia" AverageSetup="P1T0H1M" AverageAmount="120"/>
          <DevCaps Name="NodeInfo">
            <DevCap>
              <!-- NodeInfo only supports the JobPriority and TargetRoute attributes -->
              <StringState Name="TargetRoute"/>
              <IntegerState Name="JobPriority"/>
            </DevCap>
          </DevCaps>
          <DevCaps Name="ExposedMedia">
            <DevCap>
              <!--  ExposedMedia restrictions -->
              <DevCap Name="Media">
                <NameState Name="MediaUnit" DefaultValue="Sheet"/>
                <XYPairState Name="Dimension" AllowedValueMax="600 1200"
                             AllowedValueMin="0 0"/>
              </DevCap>
            </DevCap>
          </DevCaps>
          <DevCaps Name="ScanParams">
            <DevCap>
              <!--    Black and white 1 bit mode -->
              <IntegerState Name="BitDepth" DefaultValue="1" AllowedValueList="1"/>
              <EnumerationState Name="CompressionFilter"
                                AllowedValueList="CCITTFaxEncode    None"/>
              <NumberState Name="Magnification" AllowedValueMax="100"
```

```
                                 AllowedValueMin="1.e-002"/>
                 <EnumerationState Name="OutputColorSpace" AllowedValueList="GrayScale"/>
                 <XYPairState Name="OutputResolution" DefaultValue="2400 2400"/>
                   </DevCap>
                   <DevCap>
                     <!--    Grayscale 12 bit mode -->
                     <IntegerState Name="BitDepth" DefaultValue="8" AllowedValueMax="12"
                                  AllowedValueMin="1"/>
                     <EnumerationState Name="CompressionFilter"
                                      AllowedValueList="FlateEncode DCTEncode None"/>
                     <NumberState Name="Magnification" AllowedValueMax="100"
                                  AllowedValueMin="1.e-002"/>
                     <EnumerationState Name="OutputColorSpace" AllowedValueList="GrayScale"/>
                     <XYPairState Name="OutputResolution" DefaultValue="600 600"
                                  AllowedValueMax="2400 2400" AllowedValueMin="100 100"/>
                   </DevCap>
                   <DevCap>
                 <!--    Color 10 bit mode -->
                 <IntegerState Name="BitDepth" DefaultValue="8" AllowedValueMax="10"
                              AllowedValueMin="1"/>
                 <EnumerationState Name="CompressionFilter"
                                  AllowedValueList="FlateEncode DCTEncode None"/>
                 <NumberState Name="Magnification" AllowedValueMax="10"
                              AllowedValueMin="1.e-002"/>
                 <EnumerationState Name="OutputColorSpace" AllowedValueList="CMYK RGB LAB"/>
                 <XYPairState Name="OutputResolution" DefaultValue="600 600"
                              AllowedValueMax="2400 2400" AllowedValueMin="100 100"/>
               </DevCap>
             </DevCaps>
           </DeviceCap>
         </Device>
       </DeviceInfo>
     </DeviceList>
   </Response>
</JMF>
```

JDF node that is accepted by the scanner of the previous example
All parameters of the following Scanning node are compliant with the device capabilities.

```
?xml version='1.0' encoding='utf-8' ?>
<JDF xmlns=http://www.CIP4.org/JDFSchema_1 ID="GoodScan" Type="Scanning"
Status="Waiting" Version="1.1">
  <ResourcePool>
    <ScanParams ID="Link0007" Class="Parameter" Status="Available" Bit-
Depth="8" OutputColorSpace="RGB" OutputResolution="600. 600."/>
    <ExposedMedia ID="Link0008" Class="Handling" Status="Available">
      <Media Dimension="425.196850394 566.929133858"/>
    </ExposedMedia>
  </ResourcePool>
  <ResourceLinkPool>
    <ScanParamsLink rRef="Link0007" Usage="Input"/>
    <ExposedMediaLink rRef="Link0008" Usage="Input"/>
  </ResourceLinkPool>
</JDF>
```

JDF node that is rejected by the scanner of the previous example
All parameters of the following Scanning node except **Magnification** are compliant with the device capabilities.
Therefore, the device can NOT execute the job.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF xmlns=http://www.CIP4.org/JDFSchema_1" ID="BadScan" Type="Scanning"
Status="Waiting" Version="1.1">
  <ResourcePool>
```

```
    <ScanParams ID="Link0012" Class="Parameter"  Status="Available" Bit-
Depth="8" Magnification="1000. 1000." OutputColorSpace="RGB" OutputResolu-
tion="600. 600."/>
    <ExposedMedia ID="Link0013" Class="Handling"  Status="Available">
      <Media Dimension="425.196850394 566.929133858"/>
    </ExposedMedia>
  </ResourcePool>
  <ResourceLinkPool>
    <ScanParamsLink rRef="Link0012" Usage="Input"/>
    <ExposedMediaLink rRef="Link0013" Usage="Input"/>
  </ResourceLinkPool>
</JDF>
```

Chapter 8 Building a System Around JDF

8.1 Implementation Considerations and Guidelines

JDF parsing: JDF devices must implement JDF parsing. At a minimum, a device must be able to search the JDF to find a node whose process type it is able to execute. In addition, a device must be able to consume the inputs and produce the outputs for each process type it is able to execute.

Test run: To reduce failures during processing, it is recommended that either individual devices or their controller support the testrun functionality. This prevents the case where a device begins processing a node that is incomplete or malformed.

8.2 JDF and JMF Interchange Protocol

A system of vendor independent elements must define a protocol that allows them to interchange information based on JDF and JMF.

8.2.1 File-Based Protocol (JDF only)

The file-based protocol is only a solution for JDF job tickets, not JMF messages. A JMF-compliant controller must implement the HTTP protocol. A file-based protocol is based on hot folders. Every processor must define an input hot folder and an output folder for JDF. In addition the "SubmitQueueEntry" message contains a URL attribute that allows specification of arbitrary JDF locators.

Implementation of JDF file-based protocol is simple, but it is important to note that the protocol does not support acknowledgement receipts for protocol error handling. It requires that the receiver polls the output folder of the processor. Finally, granting read/write access to your hot folder negates the security functions.

8.2.2 HTTP-Based Protocol (JDF + JMF)

HTTP is a stable, vendor-independent protocol, and it supports a variety of advantageous features. For example, it offers a wide availability of tools, it is already a common technology among vendors who use HTTP, and it has a well defined query-response mechanism (HTTP post message). It also offers widespread firewall support and secure connections via SSL when using HTTPS.

8.2.3   Protocol Implementation Details

JDF Messaging will not specify a standard port. We recommend that you use the standard HTTP port 80 in order to avoid firewall problems.

Implementation of Messages

Only HTTP servers may be targeted by Query or Command messages. This is done with a standard HTTP Post request. The JMF is the body of the HTTP post message. The Response is the body of the initiated HTTP post response. Signal and Acknowledge messages are also implemented as HTTP post messages. The body of the HTTP response to these messages is empty.

HTTP Push Mechanisms

Since HTTP is a stateless protocol, push mechanisms, such as regular status bar updates, are non-trivial when communicating with a client. Work-arounds can, however, be implemented. For example, a Java applet that polls the server in regular intervals can be used.

8.2.4 MIME Types and File Extensions

The MIME type for JDF is not yet registered with IANA: http://www.iana.org/. The registration process is ongoing and the MIME types will be registered as:
JDF: `application/vnd.cip4-jdf+xml`
JMF: `application/vnd.cip4-jmf+xml`

It is recommended that the controller use a file extension of .jdf when using file-based JDF in an environment that supports file name extensions. Although JMF is not foreseen to use a file based protocol, agents that serialize JMF to a file should use a file extension of .jmf.

8.3 MIS Requirements

MIS systems may:

- Ignore Audit elements when they receive complete information about a process execution via JMF.
- Decompose JDF into an internal format such as database tables.

Appendix A Encoding

This appendix lists a number of commonly used JDF data types and structures and their XML encoding. Data types are simple data entities such as strings, numbers and dates. They have a very straightforward string representation and are used as XML attribute values. Data structures, on the other hand, describe more complex structures that are built from the defined data types, such as colors

A.1 XML Schema Data Types

JDF is based on the XML Schema specification. The JDF data types used in this specification are summarized in the table below and comply with the lexical representation of (primitive) data types defined by [XML Schema Part 2: Datatypes]. For a complete definition of each of these data types, please refer to the specification of XML Schema Datatypes.

Table A.1 XML Schema Data Types

XML Data Type	Description	Example
boolean	Has the value space required to support the mathematical concept of binary-valued logic: {*true*, *false*}.	`<Example Enable="true"/>`
date	A calendar date, it represents a time period that starts at midnight on a specified day and lasts for 24 hours. Based on ISO 8601.	`<Example StartDate="1999-05-31"/>`
dateTime	Represents a specific instant of time. It must be a Coordinated Universal Time (UTC) or the time zone must be indicated by the offset to UTC. In other words, the time must be unique in all time zones around the world.	`<Example Start="1999-05-31T18:20:00Z"/>` `<Example Start="1999-05-31T13:20:00-05:00"/>`
double	Corresponds to IEEE double-precision 64-bit floating point type	`<Example Pi="3.14"/>`
duration	Represents a duration of time. Based on ISO 8601.	`<Example Duration="P1Y2M3DT10H30M"/>`
enumeration	Limited set of **NMTOKEN**.	`<Example H="P1Y2M3DT10H30M"/>`
enumerations	Whitespace-separated list of enumeration data types.	`<Example Orientation="Flip90"/>`
gYearMonth	Represents a specific Gregorian month in a specific Gregorian year. Based on ISO 8601.	`<Example Month="2002-11"/>`
hexBinary	Represents arbitrary hex encoded binary data.	`<Example Hex="0A1C"/>`
ID	Represents the ID attribute from [XML Specification Version 1.0]. It basically represents a name or string that contains no space characters.	`<Example ID="R-16"/>`
IDREF	Represents the IDREF attribute from [XML Specification Version 1.0]. For a valid XML-document, an element with the ID value specified in IDREF must be present in the scope of the document.	`<Example IDREF="R-16"/>`

XML Data Type	Description	Example
IDREFS	Represents the IDREFS attribute from [XML Specification Version 1.0]. More specifically, this is a whitespace-separated list of IDREFs.	`<Example IDREFS="R-12 R-16"/>`
integer	Represents numerical integer values.	`<Example Copies="36"/>`
language	Represents a natural language defined in IETF rfc 1766. http://www.ietf.org/rfc/rfc1766.txt	`<Example Language="de"/>`
NMTOKEN	Represents the NMTOKEN attribute type from [XML Specification Version 1.0]. It basically represents a name or string that contains no space characters.	`<Example Alias="ABC_6"/>`
NMTOKENS	Represents the NMTOKENS attribute type from [XML Specification Version 1.0]. More specifically, this is a whitespace-separated list of NMTOKENs.	`<Example AliasList="ABC_6 ABCD_3 DEGF"/>`
string	Represents character strings in XML.	`<Example Name="Test"/>`
URI	Short for URI-reference. Represents a Uniform Resource Identifier (URI) Reference as defined in Section 4 of [RFC 2396].	`<Example URI="http://www.w3.org/1999/XMLSchema"/>`
URL	Short for **URL-reference**. Represents a Uniform Resource Locator (URL) Reference as defined in Section 4 of [RFC 2396].	`<Example URL=" file://hubble/test.txt"/>`

A.2 JDF Data Types

The data types listed and described in this section are defined by JDF. They are also found in PJTF and CIP3.

A.2.1 CMYKColor

XML attributes of type *CMYKColor* are used to specify CMYK colors.

Encoding
CMYKColor attributes are primitive data types and are encoded as a string of four numbers in the range of [0…1.0] separated by whitespace. A value of 0 specifies no ink and a value of 1 specifies full ink.

Example:
```
<Color cmyk = "0.3 0.6 0.8 0.1"> (brick red)
```

A.2.2 DurationRange

XML attributes of type *DurationRange* are used to describe a range of time durations. More specifically, it describes a time span that has a relative start and end.

Encoding
A *DurationRange* is represented by two durations, separated by a "~" (tilde) character and optional additional whitespace.

Example:
```
<XXX range="P1Y2M3DT10H30M~P1Y2M3DT10H35M"/>
```

A.2.3 IntegerList

XML attributes of type *IntegerList* are used to describe a variable length list of integer values.

Encoding

An *IntegerList* is encoded as a string of integers separated by whitespace.

Example

```
<XXX list="0 1 2 3 4 1 3 0"/>
```

A.2.4 IntegerRange

XML attributes of type *IntegerRange* are used to describe a range of integers. In some cases, ranges are defined for an unknown number of objects. In these cases, a negative value denotes a number counted from the end. For example, -1 is the last object, -2 the second to last, and so on. *IntegerRanges* that follow this convention are marked in the respective attribute descriptions.

If the first element of an *IntegerRange* specifies an element that is behind the second element, the Range specifies a list of integers in reverse order, counting backwards. For example "6~4" = "6 5 4" and "-1~0" = "last… 2 1 0".

Encoding

An *IntegerRange* is represented by two integers, separated by a "~" (tilde) character and optional additional whitespace.

Example:

```
<XXX range="-3~-5"/>
```

A.2.5 IntegerRangeList

XML attributes of type *IntegerRangeList* are used to describe a list of *IntegerRanges* and/or enumerated integers.

Encoding

A *IntegerRangeList* is represented by a sequence of *IntegerRanges* and integers, separated by whitespace.

Example:

```
<XXX list="-1~-6 3~5 7 9~128 131"/>
```

A.2.6 LabColor

XML attributes of type *LabColor* are used to specify absolute Lab colors. The Lab values are normalized to a Light of D50 and an angle of 2 degrees as specified in CIE Publication 15.2 - 1986 "Colorimetry, Second Edition" and ISO 13655:1996 "Graphic technology - Spectral measurement and colorimetric computation for graphic arts images"

This corresponds to a white point of X = 0.9642, Y = 1.0000, and Z = 0.8249 in CIEXYZ color space. L is restricted to a range of [0..100]; a and b are unbounded.

Encoding

LabColors are primitive data types and are encoded as a string of three numbers separated by whitespace:
"L a b"

Example:

```
<Color … Lab="51.9 12.6 -18.9">
```

A.2.7 Matrix

Coordinate transformation matrices are widely used throughout the whole printing process, especially in layout resources. They represent 2D transformations as defined by the PostScript and PDF Reference manuals. For more information, refer to the respective Reference Manuals, and look for "Coordinate Systems and Transformations."

Encoding

Coordinate transformation matrices are primitive data types and are encoded as a string attribute of six numbers, separated by whitespace:

```
"a b c d Tx Ty"
```

Tx and Ty describe distances and are defined in points.

Example:
```
<ContentObject CTM="1 0 0 1 3.14 21631.3" … />
```

A.2.8 NamedColor

XML attributes of type *NamedColor* are not sufficient for process color definition, but rather serve to define the colors of preprocessed products such as Wire-O binders and cover leaflets.

The entries in the following table may be prefixed by either "Dark" or "Light". The result may additionally be prefixed by "Clear" to indicate translucent material. For example, "ClearDarkBlue" indicates a translucent dark blue, "ClearBlue" a translucent blue and "Blue" indicates an opaque blue.

Table A.Named colors

Color name
White
Black
Gray
Red
Yellow
Green
Blue
Turquoise
Violet
Orange
Brown
Gold
Silver
Pink
Buff
Ivory
Goldenrod
Mustard New in JDF 1.1
MultiColor New in JDF 1.1
NoColor

Encoding
NamedColor are based on NMTOKEN.

Example:
```
<SomePlasticStuff CoverColor="ClearDarkBrown" … />
```

A.2.9 NameRange

XML attributes of type *NameRange* are used to describe a range of NMTOKEN data that are acquired from a list of named elements, such as named pages in a PDL file. It depends on the ordering of the targeted list, which names are assumed to be included in the *NameRange*. The following two possibilities exist:

1. There is no explicit ordering. In this case, alphabetical ordering is implied.

2. There is explicit ordering, such as in a list of named pages in a **RunList**. In this case, the ordering of the Runlist defines the order and all pages between the end pages are included in the *NameRange*.

Encoding

A *NameRange* attribute is represented by two NMTOKEN-{~}, separated by a "~" (tilde) character and optional additional whitespace.

Example:
```
<XXX NameRange="Jack~Jill"/>
```

A.2.10 NameRangeList

XML attributes of type *NameRangeList* are used to describe a list of *NameRanges*.

Encoding

A *NameRangeList* is represented by a sequence of *NameRanges* and NMTOKEN, separated by whitespace.

Example:
```
<XXX list="A b~f x z"/>
```

A.2.11 NumberList

XML attributes of type *NumberList* are used to describe a variable length list of numbers (double or integer).

Encoding

A *NumberList* is encoded as a string of space-separated numbers.

Example:
```
<XXX list="3.14 1 .6"/>
```

A.2.12 NumberRange

XML attributes of type *NumberRange* are used to describe a range of numbers. Mathematical spoken, the two numbers define a closed interval.

Encoding

A *NumberRange* is represented by two numbers, separated by a "~" (tilde) character and optional additional whitespace.

Example:
```
<XXX range="-3.14~5.13"/>
```

A.2.13 NumberRangeList

XML attributes of type *NumberRangeList* are used to describe a list of *NumberRanges* and/or enumerated numbers.

Encoding

A *NumberRangeList* is a sequence of *NumberRanges* and numbers separated by whitespace.

Example:
```
<XXX list="-1~-6 3.14~5.13 7 9~128 131"/>
```

A.2.14 Path

XML attributes of type *Path* are used in JDF for describing parameters such as trap zones and clip paths. In PJTF, *Paths* are encoded as a series of moveto-lineto operations. JDF has a different encoding, which is able to describe more complex paths, such as Beziers.

Encoding

Paths are encoded as an XML string attribute formatted with PDF path operators. This allows for easy adoption in PS and PDF workflows. PDF operators are limited to those described in Section 8.6.1 "Path Construction Operators" in "Portable Document Format Reference Manual", Version 1.3.

Example:
```
<ElementWithPath path="0 0 m 10 10 l 20 20 l"/>
```

A.2.15 Rectangle

XML attributes of type *Rectangle* are used to describe rectangular locations on the page, sheet, or other printable surface. A *Rectangle* is represented as an array of four numbers—llx lly urx ury—specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the rectangle, in that order. This is equivalent to the ordering: Left Bottom Right Top. All numbers are defined in points.

Encoding

To maintain compatibility with PJTF, *Rectangles* are primitive data types and are encoded as a string of four numbers, separated by whitespace:

```
"llx lly urx ury" or "l b r t"
```

Example:
```
<ContentObject ClipBox="0 0 3.14 21631.3" … >
```

Implementation Remark

Since all numbers are real numbers, any comparison of boxes should take into account certain rounding errors. For example, different *XYPairs* may be considered equal when all numbers are the same within a range of 1 point.

A.2.16 shape

XML attributes of type s*hape* are used to describe a three dimensional box.

Encoding

A *shape* is represented as an array of three numbers—x y z—specifying the Width x, height y and depth z coordinates of the shape, in that order.

Example:
```
<XXX Dimensions="10 20 40"/>
```

A.2.17 ShapeRange

XML attributes of type *ShapeRange* are used to describe a range of *Shapes* (three dimensional boxes). The range "x1 y1 z1~x2 y2 z2" describes the area $x1<=x<=x2$ and $y1<=y<=y2$ and $z1<=z<=z2$. Thus the **Shape** "2 3 4" is within "1 2 1~ 3 4 4". Note that this implies that all three values of the second entry must be >= the corresponding values of the first entry. The following example is therefore invalid: "1 2 1 ~ 0 4 4".

Encoding

A ShapeRange is represented by two Shapes, separated by a "~" (tilde) character and optional additional whitespace.

Example:
```
<XXX Shaperange="1 2~3 4"/>
```

A.2.18 ShapeRangeList

XML attributes of type *ShapeRangeList* are used to describe a list of *ShapeRange* and/or *Shapes*.

Encoding

A *ShapeRangeList* is a sequence of *ShapeRange* and *Shapes* separated by whitespace.

Example:

The brackets below the example illustrate the grouping of *Shapes* and *ShapeRanges*.
```
<XXX Shapelist="100 200 300~110 220 330 150 300 150  2 3 0 ~ 3 4 5"/>
```

( ) ( ) ( )

A.2.19 sRGBColor

XML attributes of type *sRGBColors* are used to specify *sRGB* colors.

Encoding

sRGBColors are primitive data types and are encoded as a string of three numbers in the range of [0…1.0] separated by whitespace A value of 0 specifies no intensity (black) and a value of 1 specifies full intensity:
"r g b"

Example:
```
<Color sRGB="0.3 0.6 0.8" … >
```

A.2.20 TimeRange

XML attributes of type *TimeRange* are used to describe a range of time. More specifically, it describes a time span that has a specific start and end. Mathematically, two *dateTime* values define a closed time interval.

Encoding

A *TimeRange* is represented by two *dateTimes*, separated by a "~" (tilde) character and optional additional whitespace.

Example:
```
<XXX range="1999-05-31T13:20:00-05:00~2000-08-03T23:50:00-05:00"/>
```

A.2.21 TransferFunction

XML attributes of type *TransferFunction* are functions that have a one-dimensional input and output. In JDF, they are encoded as a simple kind of sampled functions and used to describe transfer curves of processes such as ***Film-to-Plate-copy***, ***LaserCalibration*** and ***Press Calibration***. They may also be used in Color specifications, e.g., when converting a spot tint value to a CMYK value.

A transfer curve consists of a series of XY pairs where each pair consist of the stimuli(X) and the resulting value(Y). To calculate the result of a certain stimuli, the following algorithms must be applied:

1. If x <= first stimuli, then the result is the y value of the first xy pair.
2. If x >= the last stimuli, then the result is the y value of the last xy pair.
3. Search the interval in which x is located.
4. Return the linear interpolated value of x within that interval.

Encoding

A *TransferCurve* is encoded as a string of space-separated numbers. The numbers are the XY pairs that build up the transfer curve.

Example:
```
<someElementWithTransferCurve someCurve="0 0 .1 .2 .5 .6 .8 .9 1 1"/>
```

A.2.22 XYPair

XML attributes of type *XYPair* are used to describe sizes like *Dimensions* and *PageSize*. They can also be used to describe positions on a page. All numbers that describe lengths are defined in points.

Encoding

XYPair attributes are primitive data types and are encoded as a string of two numbers, separated by whitespace:
"x y"

Example:
```
<CutBlock BlockSize="612 792">
```

Implementation Remark

Since all numbers are real numbers, comparison of *XYPairs* should take into account certain rounding errors. For example, different *XYPairs* may be considered equal when all numbers are the same within a range of 1 point.

A.2.23XYPairRange

XML attributes of type *XYPairRange* are used to describe a range of *XYPairs*. The range "x1 y1~x2 y2" describes the area x1<=x<=x2 and y1<=y<=y2. Thus the *XYPair* "2 3" is within "1 2 ~ 3 4". Note that this implies that both values of the second entry must be >= the corresponding values of the first entry. The following example is therefore invalid: "1 2 ~ 0 4".

Encoding

An *XYPairRange* is represented by two *XYPairs*, separated by a "~" (tilde) character and optional additional whitespace.

Example:

```
<XXX XYrange="1 2~3 4"/>
```

A.2.24 XYPairRangeList

XML attributes of type *XYPairRangeList* are used to describe a list of *XYPairRange* and/or *XYPairs*.

Encoding

A *XYPairRangeList* is a sequence of *XYPairRange* and *XYPairs* separated by whitespace.

Example:

The brackets below the example illustrate the grouping of *XYPairs* and *XYPairRanges*.
```
<XXX XYlist="100 200~110 220  150 300  150 350  200 300 ~ 400 500"/>
              (                    )(         )(         )(                          )
```

A.3 JDF Data Structures

The following data structures are unique to JDF, although they may be comprised of existing XML structures.

A.3.1 Links

Links are defined by a combination of XML attributes of type *ID* and XML attributes of type *IDREF*. The referenced element or target of the link contains the actual information and an *ID* attribute, whereas the reference or link itself contains an *IDREF* attribute. The value of an *ID* attribute must be unique within an XML file. In order to keep the implementation burden on JDF compliant processors low, linking between distributed JDF files is not supported. The *ID* attribute of the target is always named *ID*. This is not required by XML, but it makes implementation simpler. The *IDREF* attribute in a link, however, can have varying names depending on the link type. The names of the *IDREF* attributes are defined in this document. The following example specifies a trivial link and target pair[1]:

```
<Target ID="id1" (lots of attributes)><Subelement/></Target>
…
<Link rRef="id1"/>
```

A.4 JDF File Formats

This section describes the specific file formats used by JDF. JDF uses MIME files to package different files in a single file for transmission, and when representing preview images, JDF uses the PNG image file format. The following sections explain in what ways MIME and PNG are used in JDF.

A.4.1 MIME File Packaging

JDF files are XML files but may contain references (URLs) to external data files. The following external data file types are identified, although any valid MIME file type may be referenced:

• Preview images (They are encoded using the PNG format.)

• ICC Profiles

---

[1] Note that the element names were chosen for simplicity and do not imply any naming conventions for targets and links.

- Preflight Profiles
- PDL files (PageDescription files)

One of the requirements for JDF is to support the ability to make a single, self-contained job package that contains the JDF with all of its related files, maintaining the external data references. That package will be send to a remote location where it is used for further processing. This section describes how JDF uses MIME to achieve this requirement.

MIME (Multipurpose Internet Mail Extensions) is an Internet standard that defines mechanisms for specifying and describing the format of Internet message bodies. One of its applications is the MIME Multipart/related type and is used by JDF. The MIME Multipart/Related Content-type specification can be found at http://rfc.roxen.com/rfc/rfc2112.html "The MIME Multipart/Related Content-type"

A 4.1.1   MIME Basics
MIME is comprised of headers and bodies. In case of Multipart messages, the body consists of multiple messages, each identified by the individual MIME header and separated by an unique boundary string. Normally a MIME-user agent uses the boundary string to separate different message parts, and JDF MIME files are compliant with that mechanism. Furthermore, JDF defines a Content-Length mechanism that enables fast scanning of MIME files for their body parts.

A 4.1.2   MIME Fields

Content Type
This field is always required. Content Type identifies the MIME type of the message (part). The Multipart header uses this to identify itself as a multipart message and the subparts also have MIME types to identify their content.

Content ID
This field is required for every part that is referenced by other parts. Content ID identifies each different part within a multipart MIME message. Its value can be anything as long as it is defined using USASCII. It is good practice to limit yourself to using only alphanumeric characters or only the first 127 characters of the USASCII character set in order to avoid confusing less intelligent MIME agents.

Content Transfer Encoding
This field is optional, and its default = *none*. MIME prescribes three different encodings: None, Base64 and QuotedPrintable. When no encoding is used, the data are only encapsulated by MIME headers. Base64 and QuotedPrintable encodings are commonly used algorithms for converting 8-bit and binary data into 7-bit data and vice versa. Although these encodings are not imposed, JDF agents that support MIME must be able to handle them.

A 4.1.3   CID URL scheme
One of the benefits of the MIME multipart/related mediatype is the ability to refer from one bodypart to another bodypart. This is done by using the cid: URL addressing scheme, specified in http://rfc.roxen.com/rfc/rfc2111.html "Content-ID and Message-ID Uniform Resource Locators". Please look at the example to see how it is used.

Example
```
MIME-Version: 1.0
Content-Type: multipart/related; boundary=abcdefg0123456789

--abcdefg0123456789
Content-Type: text/xml

<JDF … >
<PreviewImage Separation = "Pantone 128" URL="cid:123456.png" />
</JDF>

--abcdefg0123456789
Content-Type: image/png
Content-Transfer-Encoding: base64
Content-ID: 123456.png
Content-Length: 12345

BASE64DATA
```

```
BASE64DATA
```

```
--abcdefg0123456789--
```

A 4.1.4   JDF Agent Requirements

All JDF agents must be prepared to receive JDF files that are MIME encoded. They may choose not to support it, but they should be able to handle these JDF files gracefully. Agents that do support MIME must support Base64 and QuotedPrintable encodings.

A.4.2 HTTP 1.0 Field

Content Length

Although this field is optional, it is recommended that it be included. Content Length is used to optimize the performance of scanning multipart messages. Each multipart bodypart may have an optional Content-Length header field. Its syntax is identical to the syntax defined by RFC1945 "HTTP1.0".

When present, the Content Length identifies the number of octets of the encoded bodypart. When no encoding as is the case with 7bit, 8bit, binary, it represents the size of the bodypart. Otherwise it depends on what encoding method is used encoding (Base64, QuotedPrintable) and what the relationship is between the encoded size and the bodypart size. If an agent composing a MIME message can not derive a Content Length for its encoded body parts, it must omit the Content-Length field.

An agent parsing such a message can use the Content-Length field to seek to the end of the body. This position is calculated by using the position of the first byte of the bodypart and adding the Content Length. At that position (one byte after the bodypart contents), the agent must check if the following characters are one of either "\r\n—boundary" or "—boundary." If not, the agent must ignore the Content-Length field and resume the normal MIME Multipart behavior and restart scanning for the boundary from the beginning of the bodypart.

A.4.3 PNG Image Format

JDF uses the PNG images for representing preview images. CIP3 defined two formats: composite CMYK and separated. With PNG, only the separated format is supported for color spaces other than RGB. The composite CMYK or spot color representations must be represented as separated CMYK or spot colors. Thus, preview images are stored as separate PNG images and JDF links them together. Viewable images and thumbnails can be represented as composite RGB PNG images.

References: http://www.w3.org/Graphics/png.

Appendix B Schema

XML Schema for JDF (and JMF) will be published on: http://www.CIP4.org .

The XML Schema in the current version is not sufficient to completely validate a JDF job. For example, partitioned resources or process node types as defined in JDF cannot be validated by XML Schema processors. In other words, the structure of some elements depends on the context of usage which cannot currently described by XML Schema. Thus, the XML Schema for JDF will be structured in a way that it enables a prevalidation of valid JDF-candidates but does not preclude all syntactically invalid files to be validated.

Using JDF Schema

Your MIS system should be capable of validating whether or not a JDF Job is complete and meets JDF requirements. The schema itself may be subsetted into multiple schemas that are used for validation purposes at different points in the workflow. For instance, a JMF schema subset may be used to test and operated JDF-compliant devices on your shop floor. A process intent subset may be used to check customer submitted job specifications.

Appendix C Converting PJTF to JDF

This appendix is provided as a non-normative guide to developers writing applications that will consume PJTF version 1.1 jobs and produce JDF.

C.1 PJTF Object Conversion

Many PJTF objects are directly translatable to JDF processes or resources. Others, especially those containing multiple keys, correspond to multiple processes and resources. For example, the **JobTicketContents** object corresponds to four JDF processes and three JDF resources. And still others, such as **AuditObject**, cannot be translated to JDF at all.

Listed below are the prominent PJTF objects and the JDF components to which they correspond. Each section heading contains the title of the object in question, and each section contains a descriptive table. The first column in the tables, entitled JDFKey or Object, contains a list of the keys or objects contained within the object being described. For example, the **Accounting** object contains an **Address** object, while the **Address** object contains an **Address** key. If no subobject or key is contained within the object, then the first column is left blank and the process or resource listed is assumed to correspond directly to that object.

C.1.1 Accounting

PJTF Key or Object	JDF Process	JDF Resource	Description
Address	**-**	**Address**	-

C.1.2 Address

PJTF Key or Object	JDF Process	JDF Resource	Description
Address	**-**	**Address**	Used whenever people or organizations need to be identified.

C.1.3 Analysis

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**Analysis**	-

C.1.4 AuditObject

Audit objects must not be translated. PJTF Audit objects describe the results of operations on files, while JDF Audit elements describe the results of processes, so there is a basic incompatibility between the two. In addition, PJTF Audit objects will not be needed to direct further processing of the job after it is converted to JDF.

C.1.5 ColorantAlias

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **ColorantControl** resource.

C.1.6 ColorantControl

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**ColorantControl**	-

C.1.7 ColorantDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Keys in the PJTF ColorantDetails dictionary are a set of colorant names. The values are DeviceColorant objects.

C.1.8 ColorantZoneDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**TrappingParams**	DeviceColorant map to the ColorantZoneDetails subelement of the **TrappingParams**. resource.

C.1.9 ColorSpaceSubstitute

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **ColorantControl** resource.

C.1.10 Delivery

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Delivery*	**Address**	Specifies a quantity of a product to be delivered to an address.

C.1.11 DeviceColorant

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a Color subelement of the **ColorPool** resource. The name is entered in the SeparationSpec of a **TrappingDetails** resource.

C.1.12 Document

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
bleed media trim	**-**	**RunList**	Maps to attributes of the **RunList** resource or to processes in which they are used.
ColorantControl	**-**	**ColorantControl**	-
Files	**-**	**RunList** **FileSpec**	Maps to **FileSpec** resources contained within RunList elements.
Finishing	*AdhesiveBinding* *EndSheetGluing* *SaddleStitching* *SideSewing* *Stitching* *ThreadSewing*	**AdhesiveBindingParams** **EndSheetGluingParams** **SaddleStitchingParams** **SideSewingParams** **StitchingParams** **ThreadSewingParams**	-
FontPolicy	**-**	**FontPolicy**	The resource is attached to the applicable processes.
IgnoreHalftone	**-**	**-**	Maps to the *IgnoreHalftone* attribute of the **PDFToPSConversionParams** resource.

PJTF Key or Object	JDF Process	JDF Resource	Description
InsertPage	*Imposition*	**RunList** **Sheet**	Occurs as an attribute either of **RunList** resources or of **Sheet** resources referenced by *Imposition* processes.
NewSheet	*Imposition*	**InsertSheet**	**NewSheets** become instances of **InsertSheet** resources on **RunLists** with a *SheetUsage* attribute of "*Header*".
Media	**-**	**Media**	Maps to a subelement of the **ExposedMedia** resource.
MediaSource	**-**	**-**	Maps to a **Media** resource refelement of a **DigitalPrintingParams** resource.
MediaUsage	*Dividing*	**DividingParams**	Specifies controls for roll-fed media.
Rendering	*Rendering*	**-**	-
Trailer	*Imposition*	**InsertSheet**	Trailers become instances of **InsertSheet** resources on **RunLists** with a *Usage* attribute of "trailer"
Trapping	*Trapping*	**-**	-

C.1.13 Finishing

Finishing operations are derived from CIP3 PPF. Conversion of PJTF **Finishing** objects is vendor-dependent, since the PJTF specification does not describe any detail for **Finishing** objects.

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*AdhesiveBinding* *EndSheetGluing* *SaddleStitching* *SideSewing* *Stitching* *ThreadSewing*	**AdhesiveBinding-Params** **EndSheetGluing-Params** **SaddleStitching-Params** **SideSewing-Params** **StitchingParams** **ThreadSewing-Params**	-

C.1.14 FontPolicy

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Interpreting*	**FontPolicy**	

C.1.15 InsertPage

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**RunList**	**InsertPage** objects may generate a **InsertSheet** resource within a **RunList**.

C.1.16 InsertSheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**InsertSheet**	-

C.1.17 Inventory

PJTF Key or Object	JDF Process	JDF Resource	Description

C.1.18 JobTicket

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys except Audit, Scheduling, PreflightResults	Any process	Any resource	Keys may be represented at various levels of the JDF tree. Contents are represented as processes, resources, and versions.

C.1.19 JobTicketContents

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
Accounting	**-**	**-**	Maps to the CustomerInfo element.
Administrator	**-**	**-**	Maps to the CustomerInfo element.
ColorantControl	**-**	**ColorantControl**	-
Delivery	***Delivery***	**DeliveryParams**	-
Documents	**-**	**RunList**	May require more than one **RunList** resource.
EndMessage	**-**	**-**	Maps to the *End* attribute of the NodeInfo element.
Finishing	***AdhesiveBinding*** ***EndSheetGluing*** ***SaddleStitching*** ***SideSewing*** ***Stitching*** ***ThreadSewing***	**AdhesiveBinding-Params** **EndSheetGluing-Params** **SaddleStitching-Params** **SideSewing-Params** **StitchingParams** **ThreadSewing-Params**	-
FontPolicy	***Interpreting*** ***PDFToPS-Conversion***	**FontPolicy**	The **FontPolicy** resource is attached to any process that uses it.
IgnoreHalftone	**-**	**-**	Maps to the *IgnoreHalftone* attribute of the **PDFToPS-ConversionParams** resource.
InsertPage	***Imposition***	**RunList** **Sheet**	Occurs as an attribute either of **RunList** resources or of **Sheet** resources referenced by ***Imposition*** processes.
JobName			CustomerJobName in the CustomerInfo element of the **JobInfo** node.
Layout	***Imposition***	**Layout**	-
MarkDocuments	***Imposition***	**RunList**	Requires one of two **RunList** resources, each of which is a resource of the ***Imposition*** process.

PJTF Key or Object	JDF Process	JDF Resource	Description
MediaSource	*-*	*-*	Maps to a **MediaSource** resource refelement of a **DigitalPrintingParams** resource.
MediaUsage	*Dividing*	**DividingParams**	Specifies controls for roll-fed media.
NewSheet	*Imposition*	**InsertSheet**	**NewSheets** become instances of **InsertSheet** resources on **RunLists** with a *Usage* attribute of "header"
PrintLayout	*Imposition*	-	Maps to a subelement of the **Layout** resource.
Rendering	*Rendering*	-	Maps to the attribute of the *Rendering* process.
Scheduling	*-*	*-*	The **Scheduling** object is not translated.
StartMessage	*-*	*-*	Maps to the *Start* attribute of the NodeInfo element.
Submitter	*-*	*-*	Maps to the CustomerInfo element.
Trailer	*Imposition*	**InsertSheet**	Trailers become instances of **InsertSheet** resources on **RunLists** with a *Usage* attribute of "trailer"
Trapping	*Trapping*	-	-

C.1.20 JTFile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*-*	*-*	In most cases, JTFile objects will become **FileSpec** resources.
			If a **FilesDictionary** is present, the resource may need to be partitioned by Separation.
			If a **PlaneOrder** is present, **RunLists** which reference the file will need to be partitioned by Separation and structured to reference the page in the file appropriately.

C.1.21 Layout

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Imposition*	**Layout**	-

C.1.22 Media

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*-*	**Media**	Maps to a subelement of the **ExposedMedia** resource.

C.1.23 MediaSource

PJTF Key or Object	JDF Process	JDF Resource	Description
ManualFeed	*-*	*-*	Maps to the *ManualFeed* attribute of a **MediaSource** resource pointed to by a refelement of a **DigitalPrintingParams** or **IDPrintingParams** resource.

PJTF Key or Object	JDF Process	JDF Resource	Description
LeadingEdge			Maps to the *LeadingEdge* attribute of a **MediaSource** resource refelement of a **DigitalPrintingParams** or **IDPrintingParams** resource.
Media	-	-	Maps to a Media refelement of a **MediaSource** resource.
MediaClass	-	-	Maps to the *MediaTypeDetails* attribute of a **Media** resource of a **DigitalPrintingParams** or **IDPrintingParams** resource.
Position	-	-	Maps to the *MediaLocation* attribute of a **MediaSource** resource.

C.1.24 MediaUsage

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Dividing*	**DividingParams**	Specifies controls for roll-fed media.

C.1.25 PageRange

JobTicketContents, **Document** and **PageRange** objects are decomposed into a number of different JDF objects. Most of the key/value pairs translate into various resources.

PJTF Key or Object	JDF Process	JDF Resource	Description
bleed media trim	-	**RunList**	Maps to attributes of the **RunList** resource or to processes in which they are used.
ColorantControl	-	**ColorantControl**	-
Delivery	*Delivery*	**DeliveryParams**	-
Files	-	**RunList** **FileSpec**	Maps to **FileSpec** resources contained within RunList elements.
Finishing	*AdhesiveBinding* *EndSheetGluing* *SaddleStitching* *SideSewing* *Stitching* *ThreadSewing*	**AdhesiveBinding-Params** **EndSheetGluing-Params** **SaddleStitching-Params** **SideSewing-Params** **StitchingParams** **ThreadSewing-Params**	-
FontPolicy	*Interpreting* *PDFToPS-Conversion*	**FontPolicy**	The **FontPolicy** resource is attached to any process that uses it.
IgnoreHalftone	-	-	Maps to the *IgnoreHalftone* attribute of the **PDFToPS-ConversionParams** resource.
InsertPage	*Imposition*	**RunList** **Sheet**	Occurs as an attribute either of **RunList** resources or of **Sheet** resources referenced by *Imposition* processes.
Media	-	**Media**	Maps to a subelement of the **ExposedMedia** resource.

PJTF Key or Object	JDF Process	JDF Resource	Description
MediaSource	**-**	**--**	Maps to a **Media** resource refelement of a **DigitalPrintingParams** resource.
MediaUsage	***Dividing***	**DividingParams**	Specifies controls for roll-fed media.
NewSheet	***Imposition***	**InsertSheet**	**NewSheets** become instances of **Insert-Sheet** resources on **RunLists** with a *Usage* attribute of "header"
Rendering	***Rendering***	**-**	-
Trailer	***Imposition***	**InsertSheet**	Trailers become instances of **InsertSheet** resources on **RunLists** with a *Usage* attribute of "trailer"
Trapping	***Trapping***	**-**	-
Which	**-**	**RunList**	The *Pages* attribute or combination of *FirstPage* and *SkipPage* in **RunLists** reflect the values of *Which*. Note: More than one *PageRange* may generate *Pages* entries for a single Run.

C.1.26 PlacedObject

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **Surface** resource.

C.1.27 PlaneOrder

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**RunList**	See Section 0, Translating the Contents Hierarchy

C.1.28 Preflight

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	***Preflight***	**-**	-

C.1.29 PreflightConstraint

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **PreflightProfile** resource.

C.1.30 PreflightDetail

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **PreflightAnalysis** resource.

C.1.31 PreflightInstance

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Subelement of the **PreflightAnalysis** resource

C.1.32 PreflightInstanceDetail

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys			Subelement of the **PreflightAnalysis** resource

C.1.33 PreflightResults

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	This object is not translated.

C.1.34 PrintLayout

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Imposition*	**-**	Maps to a subelement of the **Layout** resource.

C.1.35 Profile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Preflighting*	**PreflightProfile**	

C.1.36 Rendering

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Rendering*	**RenderingParams**	-

C.1.37 ResourceAlias

PJTF Key or Object	JDF Process	JDF Resource	Description
Location		**PDLResourceAlias**	**Location** is Device
File		**PDLResourceAlias**	**File** is supported via the **SourceFile** fileref.
This		**PDLResourceAlias**	**This** is supported via the **SourceFile** fileref.
ResourceName		**PDLResourceAlias**	This key is not used. References to the aliased resource run via the ResourceLink element.
SourceFile		**PDLResourceAlias**	Source file maps to an attribute of this resource.

PJTF **ResourceAlias** objects provide a unified namespace that allows each PJTF object to refer to the resources it needs to execute the job of which it is a part. More specifically, PJTF version 1.1 supports the use of **ResourceAlias** objects to allow references to halftones and colorspaces.

For the **ResourceAlias**::**Location** key, the **File** and **This** keys are supported by a **SourceFile** attribute whose value is a fileref. The translator must provide a reference to the original PJTF file (for this) or a copy that contains the referenced resources.

C.1.38 Scheduling

Scheduling objects are not translated. It is presumed that translation of PJTF jobs into JDF is performed to allow the reuse of PJTF jobs that have been archived. Thus, the original scheduling information embedded in the PJTF is irrelevant.

C.1.39 Signature

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**-**	Maps to a subelement of the **Layout** resource.

C.2 Sheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**Sheet**	-

C.2.1 SlipSheet

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**InsertSheet**	**SlipSheets** become an **InsertSheet** resource which may define new media, and which has a *Usage* attribute of "trailer".

C.2.2 Surface

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**Surface**	-

C.2.3 Tile

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Tiling*	**Tile**	-

C.2.4 Trapping

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	*Trapping*	**TrappingParams**	-

C.2.5 TrappingDetails

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**TrappingDetails**	See the PJTF DeviceColorant object entry for details on how it is translated.

C.2.6 TrappingParameters

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**TrappingParams**	-

C.2.7 TrapRegion

PJTF Key or Object	JDF Process	JDF Resource	Description
All keys	**-**	**TrapRegion**	-

C.3 Translating Values

The PJTF version 1.1 specification lists twelve data types that may occur for the values of keys in PJTF objects. The following table describes how each of these datatypes must be represented in JDF.

PJTF Data Type	JDF Representation	Comment
boolean	boolean	-
Number	number	-
Name	name	-

PJTF Data Type	JDF Representation	Comment
Dictionary	element	All PJTF objects are dictionaries. These dictionaries generally become resources or processes as specified above.
		In addition, some PJTF objects contain embedded dictionaries whose keys are not specified (examples include TrappingParameters and ColorantDetails). These dictionaries are converted to arrays of elements, with the key name from the PJFT dictionary becoming an attribute of the subelement.
Stream	URL	PJTF supports PDF streams by reference to an object in a PDF file. The same mechanism is supported in JDF, with the JDF URL data type being used to identify the PDF file.
Rectangle	rectangle	-
Filespec	URL	-
Text	string	-
String	string	-
Date	date	-
Phone number	Phone number	The standard for the representation of phone numbers in PJTF is used here as well.

C.4 Translating the Contents Hierarchy

The contents of a PJTF job are represented in the "contents hierarchy". The hierarchy is headed by the JobTicket-Contents object, with Document, PageRange and JTFile objects occurring below. The hierarchy implicitly specifies the sequence of source pages for the job.

The contents sequence comprises all the pages specified by the first, then second, then last *PageRange* for the first Document, followed by the pages specified by the first, then last *PageRange* for the second Document, followed by the pages for the first, then last *PageRange* for the last Document. This sequence of source pages is consumed when the job is printed via PrintLayout (discussed below).

The contents hierarchy must be translated into a JDF **RunList** resource. Each LayoutElement entry in a RunList can reference a file via the FileSpec:*URL* attribute and a set of pages in the file via the Pages element. There are several additional issues related to this translation which are discussed below.

C.5 Representing Pages

In PJTF, source pages are represented as a hierarchy of Document and *PageRange* objects. Pages are referenced by page number out of files; files are represented in JTFile objects. *PageRange* objects can reference a single page, or a set of contiguous pages.

In JDF, source pages are represented as a set of partitions of the **RunList**, which reference files via URL, and pages from the files via an *IntegerRangeList* (such as '1 3~5 7~ -1').

As a consequence of this difference, pages from more than one PJTF *PageRange* object can be represented in a single **RunList** resource, assuming that all the other keys for the multiple *PageRanges* have the same values.

C.6 Representing Preseparated Documents

In preseparated workflows, all planes of each page may occur in the same file, or there may be a separate file for each plane. When all the planes occur in a single file, PJTF JTFile objects use a *PlaneOrder* object to specify which pages in the file represent each colorant plane for each source page. When each plane occurs in a separate file, the JTFile objects use a FilesDictionary to associate files with each colorant.

In JDF, both of these cases are handled through the **RunList** resource. In the case where the planes occur in separate files, the RunList is partitioned; and each partition contains the name of the colorant and the URL for the file for that colorant. In the case where the colorant planes are intermingled via *PlaneOrder* objects, the RunLists

are partitioned, but only a single URL is used for each RunList partition. Each *PlaneOrder* object will become one RunList partition.

C.7 Representing Inherited Characteristics

In PJTF, many of the characteristics of source pages—including *MediaBox*, *ColorantControl*, and *InsertPage*—may occur at all levels of the contents hierarchy. This inheritance scheme is not provided in JDF. Therefore, the correct values for each of the attributes must be translated to the appropriate element for each **RunList** element.

C.8 Translating Layout

PJTF provides two mechanisms to image a set of source pages onto a larger surface for printing: Layout and Print-Layout. Layout is a mechanism for explicitly associating specific source pages with specific locations on the surface. PrintLayout is a method for automatically positioning a sequence of source pages onto a series of surfaces.

Layout is represented as a hierarchy of PJTF objects: Signatures, Sheets, Surfaces and PlaceObjects. The Layout hierarchy may have one or more Signature objects. Each Signature must have one or more Sheets. Each Sheet must have 1 or 2 Surfaces. Each Surface may have 0 or more PlacedObjects.

PlacedObjects directly reference source pages by referring to a Document object via its Doc key, and a specific page within the sequence of pages specified by all the PageRanges in Pages arrays for that Document.

JDF defines resources which are direct translations of Signature, Sheet and Surface. PlacedObjects and MarkObjects are subelements of the Surface resource. Note: PlacedObjects identify specific source pages via a combination of Ord and either Doc or MarkDoc. Ord identifies one page out of the sequence of pages specified by all the PageRange objects for the document identified by either Doc or MarkDoc.
In the JDF PlacedObject subelement, the Ord attribute is an index into the entire sequence of pages specified by all the partitions with *IsPage* = *true* in the **RunList**. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

Similarly, in the JDF MarkObject subelement, the Ord attribute is an index into the entire sequence of pages specified by all the partitions in the **RunList** for marks. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

C.9 Translating PrintLayout

PrintLayout uses the same hierarchy of objects as Layout, but with the restriction that there can be only a single Signature. The Signature is used as a template that is repeated to consume all the source pages specified by the contents hierarchy for the job.

In addition, the PlacedObjects that occur in a PrintLayout hierarchy are not references to specific source pages. Instead, they represent the intent that a page from the sequence of source pages specified by the contents hierarchy be consumed and placed onto the Surface each time the Signature is executed.

In JDF, PrintLayout is represented via the same set of resources as Layout, except that the top of the hierarchy is an AutomatedLayout resource instead of Layout. This resource is constrained to have only one Signature resource. Note that when translating PJTF PlacedObjects to PlacedObject subelements of a Surface resource in the AutomatedLayout hierarchy, the Ord values from the PJTF PlacedObjects need not be modified. However, as in the creation of Layout, the Ord attribute for JDF MarkObject subelements are indices into the entire sequence of pages specified by all the partitions in the **RunList** for marks. So there is a translation required between the PJTF Ord value and the JDF Ord attribute.

C.10 Translating Trapping

Trapping controls are represented in PJTF as several objects: Trapping, TrappingDetails, ColorantDetails and DeviceColorants; TrappingParameters and ColorantZoneDetails; and TrapRegions. These objects can occur in multiple places in the PJTF job, and they work together to determine, for each page in the job, whether it will be trapped and how. There is also a key in the JobTicketContents object, TrappingSourceSelector, which determines which set of trapping controls will be honored.

The trapping controls in PJTF are the same, whether the trapping will be done pre-RIP or in-RIP. In translating PJTF trapping controls to JDF, there are several tasks to perform:

•   Create the required Trapping node

- Add the resources to represent the TrappingParameters which will be used
- Create the resources which represent the TrapRegions which will be used
- Determine the pages to be trapped
- Determine which controls to use for each page
- Add references to the pages in the **RunList** in the TrapRegion resource

Note: The contents hierarchy for the PJFT job must be translated into **RunList**s before trapping objects can be translated. Paths in JDF are specified as a set of path operators. PJTF TrapZone paths are a sequence of coordinates with an implied moveto at the beginning, and an implied closepath the end.

Appendix D Converting PPF to JDF

This appendix gives non-normative advice on how to convert CIP3 PPF 3.0 files to JDF encoded files. Since JDF was designed with the intention of providing the highest possible level of compatibility with PPF, many of these conversions are relatively straightforward. From the point of view of JDF, CIP3's PPF is mainly resource-based. Most of the PPF structures were, therefore, translated to JDF resources of a corresponding process. Meanwhile, the PPF product definition operations are easily translated to JDF processes of the same name, as quoted in **CIP3ProductOperation**. This kind of conversion is possible because the component structure of PPF is adopted by JDF, with some enhancements. Parameters of PPF product definition operations (**CIP3ProductParams**) are given the abbreviated name "Params," and this name is appended to the **CIP3ProductOperation** name. Thus SideSewing becomes SideSewing-Params.

In many cases, PPF key names became JDF attribute or element names with the "CIP3" prefix removed. An example of this kind of translation is provided below, and the CIP3 product structure shown in the example is expressed as a JDF process in Figure D.1, following the example.

Example: A CIP3 PPF product definition operation

```
/CIP3Products [
<<
      /CIP3ProductName (sewed book block)
      /CIP3ProductOperation /ThreadSewing
      /CIP3ProductParams <<
            /NumberOfNeedles 4
            /GlueLineRefSheets [ 0 ]
            /GlueLine <<
                  ...
            >>
            /BlindStitch false
            /Sealing false
      >>
      /CIP3ProductComponents
      [
            <<
                  /SourceType /PartialProduct
                  /SourceProduct (book block)
                  ...
            >>
      ]
>>

<<
      /CIP3ProductName (book block)
      % ...  the definition of the book block operation would go here ...
>>
] def
```
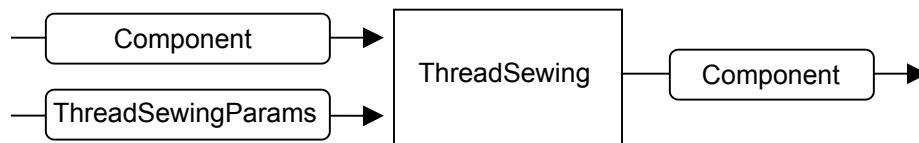

Figure D.8.1 JDF node of a CIP3 product structure

In Figure D.1, the input **Component** represents the "book block," the output **Component** represents the "sewed book block," and **ThreadSewingParams** covers all information of the **CIP3ProductParams** structure. Whenever possible, the formal conversion and translation conventions described above were followed, but because extensions and operations new to PPF are included in JDF, some exceptions were made. These exceptions are explained in de-

tail for each PPF structure in the sections that follow. Before they are explained, however, a translation of PPF data types is provided.

D.1 Converting PPF Data Types

The following table shows all PPF data types, and how they are transformed. All measuring units of CIP3 must be converted to the JDF native unit point (1/72 inch). Comments are only provided when there is something unusual or noteworthy about the translation; thus, not all translations require comment.

Table D.1 Conversion of PPF Data Types

PPF Data Type	JDF Data Type	Comments
boolean	boolean	-
Integer	integer	-
Real	double	The exponent symbol must be a capital "E" in XML.
Number	double	The exponent symbol must be a capital "E" in XML.
Name	enumeration *or* NMTOKEN	When PPF Names are used as a closed set of predefined values, they are converted to an enumeration. Otherwise, they are converted to an NMTOKEN.
String	string	Some PostScript string characters cannot be used in XML.
Array	Sequence of elements *or* IntegerList *or* NumberList	If the array consists of homogeneous integers or doubles, it is converted to an IntegerList or NumberList, otherwise to a sequence of corresponding elements.
Dictionary	element	In most cases, the structure of a Dictionary is directly converted to a XML element. Exceptions to this rule are described in the following sections.

D.2 PPF Product Definitions

The information stored in **CIP3Products** and **CIP3FinalProducts** is implicitly expressed by the structure of the JDF tree. Each product definition step is converted to a JDF node, and a product node is created for every final product of a PPF file. This is also the case for each partial product that is used in two or more final products. The following table provides information that explains how to accomplish these transformations and make these conversions. The content of the entities **CIP3ProductJobName**, **CIP3ProductJobCode**, **CIP3ProductCopyright** and **CIP3ProductCustomer** must also be copied to the parent product node. The sections that follow contain information about the conversion requirements of prominent postpress processes.

Table D.2 JDF Representation of a product definition step

PPF Key	JDF Representation	Comments
CIP3ProductName	This is expressed by an output resource link.	-
CIP3ProductOperation	JDF node	See Section 3.1 JDF Nodes.
CIP3ProductParams	Resource identified by the name of the JDF node + "Params"	For example, during a **CIP3ProductOperation** of the type "**SaddleStitching**", the JDF representation of the **CIP3ProductParams** is **SaddleStitchingParams**
CIP3ProductComponent	**Component**	See Section D.2.1, below
CIP3ProductJobName	Comment element of the JDF node	-

PPF Key	JDF Representation	Comments
CIP3ProductJobCode	*JobID* or *JobPartID* attribute of the JDF node	If the output of this step is a final product and it is only final product, it should be converted into *JobID* of the root node. Otherwise, it is converted into a *JobPartID* of the corresponding process node.
CIP3ProductCopyright	Comment element of the JDF node	-
CIP3ProductCustomer	CustomerInfo element of the JDF node	Note that the CustomerInfo element is structured, while the **CIP3ProductCustomer** is not.
CIP3ProductVolume	*Amount* attribute of the output **Component** resource link	-

D.2.1 Comparison of the PPF Component to the JDF Component

The structure of the PPF **Component** is very similar to the structure of the JDF **Component**, so it is easy to convert one to the other. The following table gives advice on how to do this. Some information stored in the PPF **Component** must be used for linking the correct resources to a process. Other implicit information, such as the bounding box of the component or an overfold, must be calculated and explicitly specified in the subelements of the **Component**. Furthermore, the appropriate algorithms can be very complex for some operations, such as folding. For further information about the **Component** resource, see Section 7.2.27 Component.

Table D.3 Converting a PPF Component

PPF Key	JDF Representation	Comments
SourceType	*ComponentType* attribute of **Component**	-
SourceSheet	*SourceSheet* attribute of **Component**	-
-	*SheetPart* attribute of **Component**	Calculable out of the cut block structure.
SourceBlock	Expressed by an input resource link to an output **Component** of a previous *Cutting* process.	see Section D.3.6 Cutting Data
SourceProduct	Expressed by an input resource link to a **Component**.	-
Params	*Transformation* attribute of **Component**	In most CIP3 operations, there is only one component parameter called **Orientation**. This matrix is renamed to *Transformation*. The only exception is the *EndSheetGluing* process. See Section **EndSheetGluing** for more information.

D.2.2 Collecting

To convert a **Collection** operation, follow the previous descriptions. This process contains no special considerations to take into account.

D.2.3 Gathering

To convert a **Gathering** operation, follow the previous descriptions. This process contains no special considerations to take into account.

D.2.4 ThreadSewing

Convert the entries of **CIP3ProductParams** structure directly to the **ThreadSewingParams** resource. Add this resource as an input resource link to the originated *ThreadSewing* process.   See Section 7.2.135 ThreadSewingParams for more information.

D.2.5 SaddleStitching

Convert the entries of **CIP3ProductParams** structure directly to the **StitchingParams** resource.   Set *StitchType*="*Saddle*". Add this resource as an input resource link to the originated *Stitching* process.   See *Stitching* for more information.

D.2.6 Stitching

Convert the entries of **CIP3ProductParams** structure directly to the **StitchingParams** resource. Set *StitchType*="*Side*". Add this resource as an input resource link to the originated *Stitching* process. See Section *Stitching* for more information.

D.2.7 SideSewing

Convert the entries of **CIP3ProductParams** structure directly to the **ThreadSewingParams** resource. Add this resource as an input resource link to the originated *ThreadSewing* process. See *ThreadSewing* for more information.

D.2.8 EndSheetGluing

The **EndSheetGluing** CIP3 operation is the only operation that requires more information than **Orientation** in the PPF Component **Params**. This additional information of the front and the back end sheet components is transferred to the **EndSheetGluingParams** resource, as described in the following table. See Section 7.2.50 for more information.

Table D.4 Converting the PPF EndSheetGluing operation to JDF

PPF Key	JDF Representation	Comments
Offset	*Offset* attribute of the EndSheet element of **EndSheetGluingParams**	-
GlueLine	GlueLine element of the EndSheet element of **EndSheetGluingParams**	See Section 7.2.50 for information on how to convert the **GlueLine** structure.

D.2.9 AdhesiveBinding

The PPF main adhesive binding operation dictionary is translated to the **AdhesiveBindingParams** resource. All single suboperations that were resident in the PPF **Processes** array are converted to special elements inside the **AdhesiveBindingParams** (see Section 7.2.3 AdhesiveBindingParams). For each type of adhesive binding suboperation there exists one extra element. The suboperations **SpinePreparation** and **GlueApplication** can simply be translated by removing the **ProcessType** entry and converting all other entries directly to the appropriate element.

The following tables show how to convert the main operation and its other suboperations. Because new features were added, the CIP3 **Lining** operation was renamed to SpineTaping.

Table D.5 Converting the PPF AdhesiveBinding operation to JDF

PPF Key	JDF Representation	Comments
Processes	Several single process:	See description above.
- BackPreparation	*SpinePreparation*	
- GlueApplication	*Gluing*	
- Lining	*SpineTaping*	
- CoverApplication	*CoverApplication*	
PullOutValue	*PullOutValue* attribute of all **SpinePreparationParams** resources, which are part of the *AdhesiveBinding* process chain.	-
PullOutMake	-	Not needed.
FlexValue	*FlexValue* attribute of **AdhesiveBindingParams**	-
FlexMake	-	Not needed.

The following tables show how to convert the main operation and its other sub-operations. Because new features were added, the CIP3 **Lining** operation was renamed to SpineTaping. Convert the PPF AdhesiveBinding sub-operation Lining to a ***SpineTaping*** process. Copy the parameters of the sub-operation to the equivalent attributes of the **SpineTapingParams** resource and link them with the process.

Table D.6 Converting the PPF AdhesiveBinding suboperation Lining

PPF Key	JDF Representation	Comments
ProcessType	Name of the JDF process.	
TopLiningExcess	*TopExcess* attribute of SpineTapingParams	-
LiningExcess	*HorizontalExcess* attribute of SpineTapingParams	-
LiningLength	*StripLength* attribute of SpineTapingParams	-
LiningMaterial	*StripMaterial* attribute of SpineTapingParams	-
LiningBrand	*StripBrand* attribute of SpineTapingParams	-

Table D.7 Converting the PPF AdhesiveBinding suboperation CoverApplication

PPF Key	JDF Representation	Comments
ProcessType	-	There is an extra element for each type of **Adhe-siveBinding** suboperation.
CoverOffset	*CoverOffset* attribute of CoverApplication	-
ScoringOffsets and **ScoringSide**	Several Score elements inside of CoverApplication	The Score element is much more structured than these two single entries.

D.2.10 Trimming

Convert the entries of **CIP3ProductParams** structure directly to the **TrimmingParams** resource. Add this resource as an input resource link to the originated ***Trimming*** process. See Section 6.5.45.9 Trimming for more information.

D.2.11 GluingIn

Because extended features have been added, the PPF **GluingIn** operation was renamed to the ***Inserting*** process. Consequently, the parameters of this CIP3 operation are transformed into the **InsertingParams** resource. For more information see Section 7.2.75 InsertingParams.

Table D.8 Converting the PPF GluingIn operation to JDF

PPF Key	JDF Representation	Comments
SheetOffset	*SheetOffset* attribute of **InsertingParams**	-
-	*Location* attribute of **InsertingParams**	Must be *Front*
GlueLines	Several GlueLine elements in **InsertingParams**	See Section 7.2.75 InsertingParams for information on how to convert the GlueLine structure.
Sample	Comment of the corresponding **Component**	Converted to an input **Component** of *Type PartialProduct*

Most of the entries of the PPF **GlueLine** structure can be directly mapped to the GlueLine element. Note that the *GluingPattern* attribute cannot have an empty array to describe a solid glue line. For this purpose, use an array of "1 0".

D.2.12 Folding

Like all formats, JDF follows a structured approach in the description of the folding process. That is why every suboperation has its own element type and has no need of the function entry. Normally, the names of the CIP3 fold functions was taken for the name of the respective corresponding process names. One of the specialized processes:

- ***Folding,***
- ***Creasing***
- ***Cutting,***
- ***Perforating***
- ***Gluing.***

is created for each folding sub-operation.

Because of inherent naming obscurities, the CIP3 functions **Groove** and **Lime** were renamed to Crease and Gluing in JDF. The following tables give advice on how to convert the PPF structures to JDF elements.

Table D.9 Converting the PPF Folding operation to JDF

PPF Key	JDF Representation	Comments
CIP3FoldDescription	-	If required, it can be expressed by the *FoldCatalog* attribute or by the fold operations.
CIP3FoldSheetIn	-	In CIP3 the parameters of the folding procedure will be scaled, if the value of the CIP3FoldSheetIn array is different from the dimension of the input component. In JDF a scaling mechanism is not supported.
CIP3FoldProc **- Fold** **- Lime** **- Cut** **- Groove** **- Perforate**	Several processes Folding Gluing Cutting Creasing Perforating	See previous description

The PPF Folding suboperation is translated to a ***Folding*** process. The parameters of the PPF command are copied into a Fold element inside the **FoldingParams** resource. The table below shows how to assign the parameters of the PPF Fold command to the equivalent attributes inside the Fold element.

Table D.10 Converting the PPF Folding suboperation of type Fold

PPF Key	JDF Representation	Comments
travel	*Travel* attribute of Fold	-
from	*From* attribute of Fold	-
to	*To* attribute of Fold	-
function	-	-

For every lime operation, a ***Gluing*** process is generated. Create a **GluingParams** resource and add a Glue element. Insert the value of the working-direction attribute into the *WorkingDirection* attribute. Attach a GlueApplication element. To this element add a GlueLine element. The attributes start-position and working-path can put into the equivalent attributes *StartPosition* and *WorkingPath* inside the GlueLine.

Table D.11 Converting the PPF Folding suboperation of type Lime

PPF Key	JDF Representation	Comments
start-position	*StartPosition* attribute of the GlueLine element of the Gluing element	JDF uses the GlueLine element because of the advantage of more optional attributes of this type of element.

PPF Key	JDF Representation	Comments
working-path	*WorkingPath* attribute of the GlueLine element of the Gluing element	JDF uses the GlueLine element because of the advantage of more optional attributes of this type of element.
working-direction	*WorkingDirection* attribute of the Gluing element	-
function	-	-

The remaining operation types can converted to one of the following processes:
- *Cutting*. Create a **CuttingParams** resource and link it to the process. Transfer the parameters of the PPF Cut command into equivalent attributes of a Cut element and insert this into the **CuttingParams** resource.
- *Creasing*. The same as above except that there is a **CreasingParams** resource with a Crease element inside which will fill with the converted parameters of the PPF Groove command.
- *Perforate*. The same as above except that there is a **PerforatingParams** resource with a Perforate element inside which will fill with the converted parameters of the PPF Perforate command.

Table D.12 Converting the PPF Folding suboperation of all other types

PPF Key	JDF Representation	Comments
start-position	*StartPosition* attribute of the respective Cut / Crease / Perforate element	-
working-path	*WorkingPath* attribute of the respective Cut / Crease / Perforate element	-
working-direction	*WorkingDirection* attribute of the respective Cut / Crease / Perforate element	-
function	-	There is an extra element for each type of a **Folding** suboperation. The extra elements are: Cut, Crease, and Perforate

D.3 PPF Sheet Structure

The conversion of the PPF sheet structures is much more complex than the conversion of the product operations. A JDF layout structure, which is not directly specified in PPF, must be built up in order to place the mark objects such as register mark or density measuring field. All other sheet information is stored in specialized resources. These resources are often partitionable to specify the sheet, surface and separation to which they belong (see Section 3.9.2 Description of Partitionable Resources). The result is an inheritance of attributes comparable to the inheritance process in CIP3.

To build the layout structure, create a **Layout** resource that includes one Signature element with a unique *Name*. For each PPF **Sheet**, add one **Sheet** resource to the Signature. Set the *Name* of the corresponding **Sheet** to the value of **CIP3AdmSheetName**. For each surface (front or back) initiate a **Surface** resource with one PlacedObjects element. In order to define a mark object, i.e., CutMark, CIELABMeasuringField, DensityMeasuringField, ColorControlStrip, or RegisterMark, build a MarkObject element inside PlacedObjects. In that element, define *CTM* and an appropriate *LayoutElement*. The CIP3 information is added to the MarkObject by including the mark-specific element, e.g., RegisterMark for a register mark. Note: The coordinate system of the JDF **Sheet** is specified by the *SurfaceContentsBox*, which defaults to the page coordinates and the coordinate system of the CIP3 **Sheet** is the PSExtent coordinates.
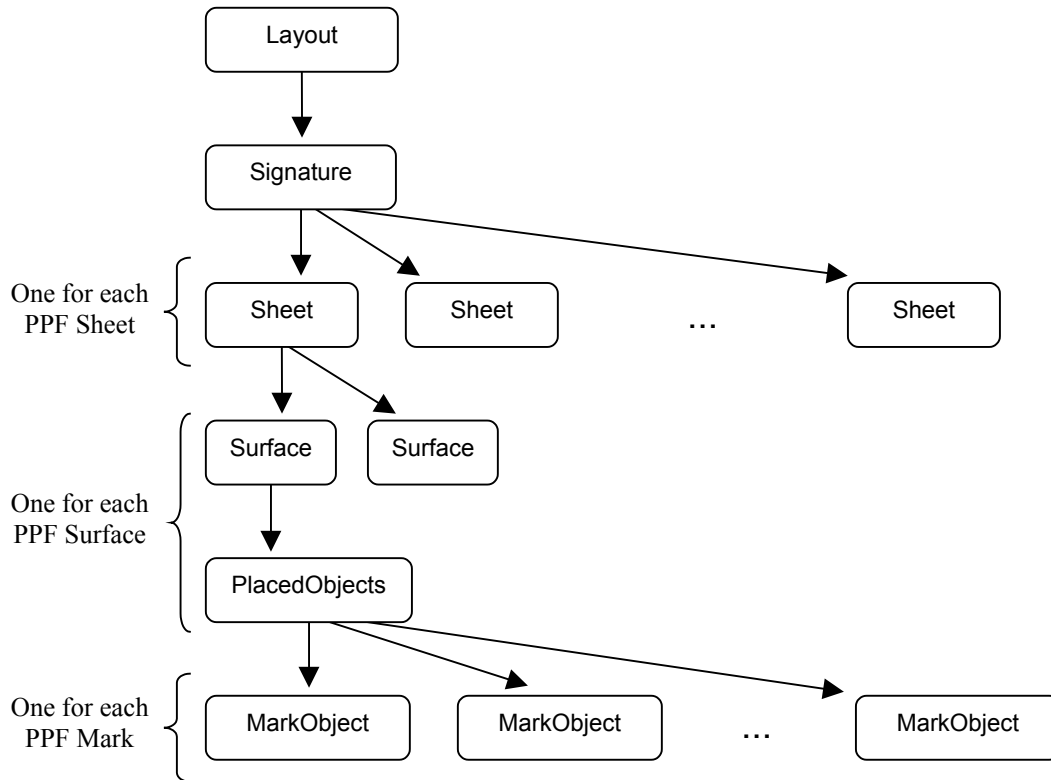
Figure D.8.2 JDF representation of sheets

If there are no product definitions in the PPF file, create JDF product nodes which are the results of all cutting and folding information in the sheet structure.

D.3.1 Administration Data

The following table defines how to convert the administration data of CIP3. In some situations, it may not be clear whether or not conversion is necessary. Processes such as **CIP3AdmFilmType**, for example, contain limited information, making it difficult to tell.

Table D.13 Converting administration data

PPF Key	JDF Representation	Comments
CIP3AdmSheetName	*Name* attribute of the corresponding **Sheet**	If there is no **CIP3AdmSheetName**, define a unique new one.
CIP3AdmJobName	**Comment** of the corresponding product node	-
CIP3AdmJobCode	*JobPart* of the corresponding product node	May conflict with **CIP3ProductJobCode**.
CIP3AdmMake	-	Not supported.
CIP3AdmModel	-	Not supported.
CIP3AdmSoftware	-	Not supported.
CIP3AdmCreationTime	-	Not supported.
CIP3AdmArtist	**Comment** of the corresponding product node	-
CIP3AdmCopyright	**Comment** of the corresponding product node	-

PPF Key	JDF Representation	Comments
CIP3AdmCustomer	CustomerInfo element of the corresponding product node	May conflict with **CIP3ProductCustomer**. Note: The CustomerInfo element is structured while the **CIP3AdmCustomer** is not.
CIP3AdmPSExtent	indirect	-
CIP3AdmTypeOfScreen	see description	Not possible to convert appropriately.
CIP3AdmFilmType	*Brand* attribute of the corresponding **Media** resource	*MediaType* of the **Media** is *Film*.
CIP3AdmFilmExtent	*Dimension* attribute of the corresponding **Media** resource	-
CIP3AdmFilmTrf	TransferCurveSet:*CTM*	TransferCurveSet:*Name* = "*Film*"
CIP3AdmPlateType	*Brand* attribute of the corresponding **Media** resource	*MediaType* of the **Media** is *Plate*.
CIP3AdmPlateExtent	*Dimension* attribute of the corresponding **Media** resource	-
CIP3AdmPlateTrf	TransferCurveSet:*CTM*	TransferCurveSet:*Name* = "*Plate*"
CIP3AdmPaperGrade	*Grade* attribute of the corresponding **Media** resource	*MediaType* of the **Media** is *Paper*
CIP3AdmPaperGrammage	*Weight* attribute of the corresponding **Media** resource	See **CIP3AdmPaperGrade.**
CIP3AdmPaperThickness	*Thickness* attribute of the corresponding **Media** resource	See **CIP3AdmPaperGrade.**
CIP3AdmPaperColor	*Lab* attribute of the Color element of the corresponding **Media** resource	See **CIP3AdmPaperGrade.**
CIP3AdmPaperExtent	*Dimension* attribute of the corresponding **Media** resource	-
CIP3AdmPaperTrf	TransferCurveSet:*CTM*	TransferCurveSet:*Name* = "*Paper*"
CIP3AdmSeparationNames	see description	Create a **ConventionalPrinting** process (see Section 6.4.1) and a corresponding **ColorantControl** resource. Fill the *ColorantOrder* parameter.
CIP3AdmSheetLay	*SheetLay* attribute of the corresponding **ConventionalPrintingParams** or **FoldingParams** resource	-
CIP3AdmPrintVolume	*Amount* attribute of the output **Component** resource link of the printing process	-
CIP3AdmPressTrf	TransferCurveSet:*CTM*	TransferCurveSet:*Name* = "*Press*"
CIP3AdmPressExtent	indirect	-
CIP3AdmInkInfo	*Name* attribute of the Color element of the corresponding **Ink** resource	Create a partitioned **Ink** matching the side and separation. Add the **Ink** to the **ConventionalPrinting** process of **CIP3AdmSeparationNames**
CIP3AdmInkColors	*LabColor* attribute of the Color element defined by the *ColorName* of the Ink resource.	see **CIP3AdmInkInfo**

D.3.2 Preview Images

In PPF, preview images are coded as an in-line image. This is not possible in version 1.0 of XML, so JDF uses the *URL* attribute within the **Preview** resource (see Section 7.2.106 Preview), which points to an external PNG file. The following table shows how to translate the PPF preview structure to the PNG header. Use the partition feature to assign a preview image to a specific separation and surface.

Table D.14 PPF preview representation as PNG

PPF Key	JDF Representation	Comments
CIP3PreviewImageWidth	"Width" of the "IHDR" chunk of the PNG file	-
CIP3PreviewImageHeight	"Height" of the "IHDR" chunk of the PNG file	-
CIP3PreviewImageBitsPerComp	"Bit depth" of the "IHDR" chunk of the PNG file	-
CIP3PreviewImageComponents	-	Because of a lack of CMYK composite support by PNG, PPF previews of this type must be separated.
CIP3PreviewImageImageMatrix	-	Not needed. Convert image data to the PNG native sequence.
CIP3PreviewImageResolution	"pHYs" chunk of the PNG file	Use the meter unit and convert DPI to DPM.
CIP3PreviewImageEncoding	-	Not needed.
CIP3PreviewImageCompression	-	Not needed. Use PNG's own compression.
CIP3PreviewImageFilterDict	-	Not needed.
CIP3PreviewImageByteAlign	-	Not needed.
CIP3PreviewImageDataSize	-	Not needed.

To calculate ink zones, JDF uses a process chain of *PreviewGeneration* and *InkZoneCalculation* processes. Add the converted CIP3 previews as an input resource to *InkZoneCalculation*. The *ProfileOffset* attribute of **InkZoneCalculationParams** can be calculated out of the different CIP3 coordinate systems.

D.3.3 Transfer Curves

Simply convert all CIP3 transfer curves to elements of a partitioned **TransferCurvePool** (see Section 7.2.136 Tile). Add this **TransferCurvePool** as an input resource to a corresponding *InkZoneCalculation* process.

D.3.4 Register Marks

The table provides information about how to create a JDF **RegisterMark** and place this element inside the respective MarkObject.

Table D.15 Converting the parameter of the CIP3PlaceRegisterMark command

PPF Key	JDF Representation	Comments
translate-x and **translate-y**	*Center* attribute of **RegisterMark**	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
rotation	*Rotation* attribute of **RegisterMark**	-
type	*MarkType* attribute of **RegisterMark**	-

PPF Key	JDF Representation	Comments
Current **CIP3SetRegisterMark-Separations** context	Several SeparationSpec elements inside the **RegisterMark**	-

D.3.5 Color and Ink Control

In CIP3, the two types of measuring fields are specified by an entry of the data dictionary in the **CIP3PlaceMeasuringField** command. In JDF, this approach is replaced by two different types of JDF elements: CIELABMeasuringField and DensityMeasuringField. All parameters of the **CIP3PlaceMeasuringField** command are merged into these elements. See the following tables as well as Section 7.2.15 CIELABMeasuringField and Section 7.2.42 DensityMeasuringField for further information. All PPF entries that are not explicitly listed in the following tables can be directly converted. Place the originated element inside the appropriate MarkObject.

Table D.16 Converting PPF color-measuring data

PPF Key	JDF Representation	Comments
position-x and **position-y** of the respective **CIP3-PlaceMeasuringField** command	*Center* attribute of **CIE-LABMeasuringField**	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
Type	-	There is an extra resource for each type of CIP3 measuring field.
CIE-L*, **CIE-a*** and **CIE-b***	*CIELab* attribute of **CIE-LABMeasuringField**	-

Table D.17 Converting PPF density-measuring data

PPF Key	JDF Representation	Comments
position-x and **position-y** of the respective **CIP3-PlaceMeasuringField** command	*Center* attribute of **DensityMeasuringField**	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
Type	-	There is an extra resource for each type of CIP3 measuring field.
DensityCyan, **DensityMagenta**, **DensityYellow** and **Density-Black**	*Density* attribute of **DensityMeasuringField**	-

Like the measuring fields, the **CIP3PlaceColorControlStrip** command is translated to a structured element. All parameters of this command can be converted to the ColorControlStrip element (see Section 7.2.20) by following the instructions in table D.18, below.

Table D.18 Converting the parameter of the CIP3PlaceColorControlStrip command

PPF Key	JDF Representation	Comments
position-x and **position-y**	*Center* attribute of **Color-ControlStrip**	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
rotation	*Rotation* attribute of **Color-ControlStrip**	-
width and **height**	*Size* attribute of **Color-ControlStrip**	-

PPF Key	JDF Representation	Comments
data	Sequence of **Density-MeasuringField** elements within the **ColorControl-Strip**	The entries of the **data** parameter have to be converted to **DensityMeasuringField** elements.
name	*StripType* attribute of **ColorControlStrip**	-

D.3.6 Cutting Data

CIP3's cut block structure is translated to JDF by defining *Cutting* processes. Since CIP3 has the ability to create nested cut blocks, one separate *Cutting* process is needed for each nested block set. Simply follow the instructions in the following table, and add all originated **CutBlock** resources as input the corresponding *Cutting* process. The **CIP3CutModel** entry is not used in JDF.

Table D.19 Converting the Cutting Data structure

PPF Key	JDF Representation	Comments
CIP3BlockTrf	*BlockTrf* attribute of **CutBlock**	If the **CutBlock** is at the uppermost level, apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
CIP3BlockSize	*BlockSize* attribute of **CutBlock**	-
CIP3BlockElementSize	*BlockElementSize* attribute of **CutBlock**	-
CIP3BlockSubdivision	*BlockSubdivision* attribute of **CutBlock**	Determines how many **Components** are produced.
CIP3BlockType	*BlockType* attribute of **CutBlock**	-
CIP3BlockElementType	*BlockElementType* attribute of **CutBlock**	-
CIP3BlockName	This is expressed by resource links	Not needed in JDF.
CIP3BlockFoldingProcedure	A *Folding* process	See Folding

For cut marks, follow the instructions in the table below. Place the originated element inside the appropriate MarkObject.

Table D.20 Converting the parameter of the CIP3PlaceCutMark command

PPF Key	JDF Representation	Comments
position-x and **position-y**	*Center* attribute of **Cut-Mark**	Apply all transformations of the CIP3 coordinate systems to get from the PS system to the **Layout** system.
mark-type	*MarkType* attribute of **Cut-Mark**	-

D.3.7 Folding Data

When a CIP3 cut block has a folding operation defined (**CIP3BlockFoldingProcedure**), append a JDF *Folding* process which uses the respective output **Component** of the respective *Cutting* process as an input **Component**. See *Folding* for more information on how to translate the CIP3 folding procedure, which is used to fold the cut block.

D.3.8 Comments and Annotations

PPF comments can either be converted to an XML comment or to a human-readable form by transforming them into a Comment telem of the next element. In most cases, PPF comments can simply be ignored. Annotations are not supported by JDF.

D.3.9 Private Data and Content

For your private data, you should first examine if one of the new JDF elements or attributes fits your requirements. If not, please use the extension capabilities of JDF to express your needs. They are described in Section 3.11.

Appendix E Modeling IfraTrack in JDF

Introduction

Job tracking and production control are integral parts of a workflow system. IFRA, described in this section, has defined a job tracking system called IfraTrack that fulfills a large number of the job tracking requirements of a production scenario and is especially effective in newspaper production. The JDF messaging system generalizes the IfraTrack approach, expanding its focus from a newspaper workflow to one that encompasses the entire graphic arts industry. This appendix provides further detail about the way in which JDF expands upon the existing IfraTrack technology.

E.1 IFRA Objects and JDF Nodes

IfraTrack traces the status of objects, and these objects are modified by processes that are only generic. JDF, on the other hand, precisely defines process nodes that create output resources. These JDF output resources are equivalent to IfraTrack objects, so tracking the state of a JDF node conveys a superset of the information communicated by tracking the state of an IfraTrack. The sections that follow define the mapping of IFRA concepts to JDF concepts in greater detail.

E.1.1 Object Identification

IfraTrack defines objects with an object path. The object path, in turn, may be a unique identifier, or UID. JDF also supports UIDs for internal linking of objects, although these UIDs should not be exported beyond the scope of a JDF document. External references to JDF nodes should be made the JobID/JobPartID pair. These values may be defined by an external system, such as MIS, and can be used to uniquely track JDF nodes.

E.1.2 IFRA Object Hierarchy

IfraTrack defines an explicit hierarchy to define a newspaper, from Issue through Edition, EditionVersion, and so on. JDF, on the other hand, defines a generic hierarchy of products containing a description attribute that allows the products to be named. An IfraTrack-conforming JDF job consequently includes a product hierarchy with product nodes that contain the appropriate description fields. Furthermore, the abstract IFRA Element type is mapped to the JDF **LayoutElement** type.

E.1.3 Object States

IFRA defines object states that define the status of a resource, although they also define the status of the process that defines a resource. JDF defines explicit states for both processes and resources. In addition, JDF defines a descriptive string to denote the details of each status. The mapping is defined in the following table.

Table E.1 IFRA object states

IFRA Object Status	JDF Node Status	JDF Resource Status	Description
Not Started	*Waiting*	*Unavailable*	Status prior to InProgress.
	Ready	*Unavailable*	JDF defines a test-run mode that allows generalized preflighting. *Ready* is the status after *TestRun*.
In Progress	*Setup*	*Unavailable*	A process is InProgress but not yet producing any output.
	InProgress	*Unavailable*	A process is InProgress.
	Cleanup	*Available*	A process is running after all output has been produced.
On Hold	*Stopped*	*Unavailable*	A process is active but not currently producing, as when maintenance is run during a job.

IFRA Object Status	JDF Node Status	JDF Resource Status	Description
Completed	*Completed*	*Available*	Completed
Aborted	*Aborted*	*Unavailable*[1]	Fatal Error

E.1.4 Deadlines and Scheduling

In IfraTrack, activities may be linked to deadlines. JDF defines deadlines in the NodeInfo element of every node. The definition of deadline values is identical.

IFRA defines an integer value for deadline level. JDF defines four explicit enumerations for *DueLevel* in order to assure that devices in a heterogeneous system have the same concept of deadline level.

E.2 JMF Messages that Translate IfraTrack Messages

The messages explained in Section 5.5.2 Device/Operator Status and Job Progress Messages can be used to emulate IfraTrack functionality. Specifically the messages:

- 5.5.2.3 Status
- 5.5.2.4 Track

---

[1] Unless aborted during cleanup

Appendix F   Mapping between JDF and IPP

The mapping between JDF and IPP is specified in Appendix F in JDF/1.0 using the IDPrinting process. However, for JDF/1.1, the IDPrinting process is deprecated. Thus for JDF/1.1, mapping between JDF/1.1 and IPP should be done with the DigitalPrinting process and many other JDF/1.1 processes as a combined process node.

F.1 IPP References

The documents below give detailed information about IPP attributes.

- IPP Model and Semantics, RFC 2911, September 2000

- Collection attribute syntax, <draft-ietf-ipp-collection-05.txt>, July 17, 2001

- Production Printing Attributes - Set1, IEEE-ISTO 5100.3-2001,
  ftp://ftp.pwg.org/pub/pwg/standards/pwg5100.3.pdf, .doc, .rtf, February 17, 2001

- Override Attributes for Documents and Pages, IEEE-ISTO 5100.4-2001,
  ftp://ftp.pwg.org/pub/pwg/standards/pwg5100.4.pdf, .doc, .rtf, February 7, 2001

- IPP/1.0 & 1.1: "Output-bin" attribute extension, IEEE-ISTO 5100.2-2001,
  ftp://ftp.pwg.org/pub/pwg/standards/pwg5100.2.pdf, .doc, .rtf, February 7, 2001

- IPP/1.1: finishings attribute values extension, IEEE-ISTO 5100.1-2001,
  ftp://ftp.pwg.org/pub/pwg/standards/pwg5100.1.pdf, .doc, .rtf, February 5, 2001

- Job Progress Attributes, <draft-ietf-ipp-job-prog-03.txt>, July 17, 2001.

Appendix G StatusDetails Supported Strings

The *StatusDetails* attribute refines the concept of a job status to be job specific or a device status to be device specific. The following tables define individual *StatusDetail* values and map them to the appropriate job specific state *Status* or device specific state *DeviceStatus*.

Table G.1 StatusDetails and Status mapping for generic devices

StatusDetails	Status	DeviceStatus	Description
ControllDeferred	-	*Stopped*	The device is controlled by a master device and cannot be accessed.

Table G.2 StatusDetails and Status mapping for conventional printing devices

StatusDetails	Status	DeviceStatus	Description
Good	*InProgress*	*Running*	Production of sheets in progress, good copy counter is on.
Waste	*InProgress*	*Running*	Production of sheets in progress, good copy counter is off.
FormChange	*Setup*	*Setup*	In conventional printing. changing of plates or in digital printing changing of images.
SizeChange	*Setup*	*Setup*	Changing setup for media size.
WashUp	*Cleanup*	*Cleanup*	Machine is washed before, during or after production. *WashUp* is a super-term for *BlanketWash*, *CylinderWash*, *CleaningInkingUnit*, or *CleaningInkFountain*. *WashUp* is the default which is assumed if **StatusDetails** is not specified.
InkingRollerWash	*Cleanup*	*Cleanup*	Washing of the inking roller, subterm of *WashUp*.
PlateWash	*Cleanup*	*Cleanup*	Washing of the plate, subterm of *WashUp*.
DampeningRoller-Wash	*Cleanup*	*Cleanup*	Washing of the dampening roller, subterm of *WashUp*.
BlanketWash	*Cleanup*	*Cleanup*	Washing of the blanket, subterm of *WashUp*.
CylinderWash	*Cleanup*	*Cleanup*	Washing of impression cylinders, subterm of *WashUp*.
CleaningInkFountain	*Cleanup*	*Cleanup*	Cleaning of the ink fountain, subterm of *WashUp*.
Pause	*Stopped*	*Stopped*	Machine paused, restart is possible.
MissResources	*Stopped*	*Stopped*	Production has been stopped because resources are missed. For example, if the machine has consumed paper, ink, plates, etc., and waits for new resources, subterm of Pause.
WaitForApproval	*Stopped*	*Stopped*	Production has been stopped because a required approval is still missing, subterm of Pause.
ShutDown	*Stopped*	*Down*	Machine stopped (may be switched off), restart requires a run up.
BreakDown	*Stopped*	*Down*	Breakdown of the device, repair required.
Repair	*Stopped*	*Down*	After a breakdown the device is being repaired.
Failure	*Stopped*	*Stopped*	Failure of the device. Requires some maintenance in order to restart the device.
PaperJam	*Stopped*	*Stopped*	Paper jam in the device, subterm of *Failure*.
Maintenance	*Stopped*	*Stopped*	Maintenance of the device.

StatusDetails	Status	DeviceStatus	Description
BlanketChange	*Stopped*	*Stopped*	Changing of blankets, subterm for *Maintenance*.
SleeveChange	*Stopped*	*Stopped*	Changing of sleeves, subterm for *Maintenance*.

Appendix H ModuleType Supported Strings

Both the ModuleStatus element (see Table 5-46 Contents of the ModuleStatus element) and the ModulePhase element (see Table 3-33 Contents of the ModulePhase element) contain a *ModuleType* attribute that defines individual modules within a machine. The following table defines individual *ModuleType* values.

Table H.1 ModuleType definition for conventional printing devices

ModuleType	Description
Feeder	Feeder module, feeds the device with paper.
PrintModule	Unit for printing a color.
CoatingModule	Unit for coatings, for example, full coating of varnish.
Drier	Module for drying the previously printed color or varnish.
PerfectingModule	Unit for perfecting, reversing device.
ExtensionModule	Unit for extending the distance between modules, for example to increase the distance between the last printing module and the delivery module.
Delivery	Delivery module, unit for gathering the printed sheets.
Imaging	Imaging Module in a direct to plate machine.
Numbering	Numbering unit.

Appendix I   Supported Error Codes in JMF

The following list defines the standard *ReturnCode* for messaging. The ID numbers are decimal. Error messages below 100 are reserved for protocol errors. Error messages above 100 are used for device and controller errors and error messages above 200 for job and pipe specific errors.

Table J.1 Return codes for JMF

ReturnCode	Description
0	Success
1 – 99	Protocol errors
1	General error
2	Internal error
3	XML parser error, e.g., if a MIME file is sent to an XML controller.
4	XML validation error
5	Query/command not implemented
6	Invalid parameters
7	Insufficient parameters
8	Device not available (controller exists but not the device or queue)
9	Message incomplete. Message Service is busy
100 – 199	Device and controller errors
100	Device not running
101	Device incapable of fulfilling request, e.g., a RIP that has been asked to cut a sheet.
102	No executable node exists in the JDF
103	Job ID not known by controller
104	JobPartID not known by controller
105	Queue entry not in queue
106	Queue request failed because queue entry is already executing
107	Queue entry is already executing. Late change is not accepted
108	Selection or applied filter results in an empty list
109	Selection or applied filter results in an incomplete list. A buffer cannot provide the complete list queried for.
110	Queue request of a job submission failed because the requested completion time of the job cannot be fulfilled.
111	Subscription request denied.
112	Queue request failed because the Queue is closed and does not accept new entries. New in JDF 1.1
200 – …	Job and pipe specific errors
200	Invalid resource parameters
201	Insufficient resource parameters
202	PipeID unknown
203	Unlinked resource link

Appendix J   NotificationDetails

The Notification element is used for messaging and logging of events. It is defined in Section 3.10.1.2 Notification. Notifications are grouped into five classes: *event*, *information*, *warning*, *error*, and *fatal*. For notification classes see Section 4.6.1 Classification of Notifications. In addition to the classes, the *Type* attribute and abstract NotificationDetails element provide a container for detailed information about the notification.

Elements derived from the abstract NotificationDetails element represent a structured and extensible data type. It is defined in section 3.10.1.2.1 NotificationDetails. The structure of various predefined NotificationDetails-types and their descriptions are listed in the following sections.

J.1 Predefined NotificationDetails

This section defines elements that are derived from the abstract element.

J.1.1 Barcode

A bar code has been scanned.

Table J. 1 Contents of the Barcode element

Name	Data Type	Description
Code	string	Contains the scanned bar code.

J.1.2 FCNKey

A function key has been activated at a console.

Table J. 2 Contents of the FCNKey element

Name	Data Type	Description
Key	integer	Contains the number of that function key.

J.1.3 SystemTimeSet

The system time of a device/controller/agent has been set, e.g., readjusted, changed to daylight saving time, etc..

Table J. 3   Contents of the SystemTimeSet element

Name	Data Type	Description
NewTime	dateTime	Contains the new time.
OldTime ?	dateTime	Contains the old time.

J.1.4 CounterReset

The production counter of a device has been reset.

Table J. 4 Contents of the CounterReset element

Name	Data Type	Description
CounterID ?	string	Identification of the counter that has been set.
LastCount ?	integer	Last counter value before reset.

J.1.5 Error

This element provides additional information for common errors.

Table J. 5 Contents of the Error element, derived from NotificationDetails

Name	Data Type	Description
ErrorID	string	Internal Error ID of the application that declares the error.

Appendix K   Examples

Note that these examples were generated using prototype tools and should be used for general overview only.   The emphasis is *not* on the individual bytes, e.g., capitalization or exact keywords.   Normative examples will be provided at http://www.CIP4.org when available.

K.1   Brief Example

K.1.1   Before Processing

This is a simple example of a JDF that describes color conversion for one file.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="ColorTest" Type="ColorSpaceConversion" JobID="ColorJob" Status="Waiting" Version="1.1"
xmlns="http://www.CIP4.org/JDFSchema_1_1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <NodeInfo/>
  <ResourcePool>
    <RunList ID="Link0003" Class="Parameter" Status="Available" Pages="0~-1">
      <LayoutElement>
        <FileSpec URL="File://in/colortest.pdf"/>
      </LayoutElement>
    </RunList>
    <ColorSpaceConversionParams ID="Link0004" Class="Parameter" Status="Available">
      <FileSpec ResourceUsage="FinalTargetDevice" URL="File://SMProcessCMYK.icc"/>
      <ColorSpaceConversionOp SourceCS="RGB" Operation="Convert" SourceObjects="ImagePhotographic
ImageScreenShot SmoothShades" SourceProfile="File:://image.icc" RenderingIntent="Perceptual"/>
      <ColorSpaceConversionOp SourceCS="RGB" Operation="Convert" SourceObjects="Text LineArt"
SourceProfile="File://text.icc" RenderingIntent="Perceptual"/>
    </ColorSpaceConversionParams>
    <ColorPool ID="Link0005" Class="Parameter" Status="Available">
      <Color CMYK="1 0 0 0" Name="Cyan"/>
      <Color CMYK="0 1 0 0" Name="Magenta"/>
      <Color CMYK="0 0 1 0" Name="Yellow"/>
      <Color CMYK="0 0 0 1" Name="Black"/>
      <Color CMYK="0.8 0.8 0 0" Name="Blue"/>
    </ColorPool>
    <ColorantControl ID="Link0006" Class="Parameter" rRefs="Link0005" Status="Available" Process-
ColorModel="DeviceCMYK">
      <ColorPoolRef rRef="Link0005"/>
    </ColorantControl>
    <RunList ID="Link0007" Class="Parameter" Status="Unavailable" Pages="0~-1">
      <LayoutElement>
        <FileSpec URL="File://out/colortest.pdf"/>
      </LayoutElement>
    </RunList>
  </ResourcePool>
  <ResourceLinkPool>
    <RunListLink rRef="Link0003" Usage="Input"/>
    <ColorSpaceConversionParamsLink rRef="Link0004" Usage="Input"/>
    <ColorPoolLink rRef="Link0005" Usage="Input"/>
    <ColorantControlLink rRef="Link0006" Usage="Input"/>
    <RunListLink rRef="Link0007" Usage="Output"/>
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="Rainer's JDFWriter 0.2000" TimeStamp="2000-11-01T10:26:11+01:00"/>
  </AuditPool>
</JDF>
```

K.1.2   After Processing

This is a simple example of a JDF that describes color conversion for one file after the color conversion process has been executed.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="ColorTest " Type="ColorSpaceConversion" JobID="ColorJob" Status="Completed" Ver-
sion="1.1" xmlns="http://www.CIP4.org/JDFSchema_1_1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <ResourcePool>
    <RunList ID="Link0003" Class="Parameter" Status="Available" Pages="0~-1">
      <LayoutElement
```

```
            <FileSpec URL="File://in/colortest.pdf"/>
        </LayoutElement>
      </RunList>
      <ColorSpaceConversionParams ID="Link0004" Class="Parameter" Status="Available">
        <FileSpec ResourceUsage="FinalTargetDevice" URL="File://SMProcessCMYK.icc"/>
        <ColorSpaceConversionOp SourceCS="RGB" Operation="Convert" SourceObjects="ImagePhotographic
ImageScreenShot SmoothShades" SourceProfile="File://image.icc" RenderingIntent="Perceptual"/>
        <ColorSpaceConversionOp SourceCS="RGB" Operation="Convert" SourceObjects="Text LineArt"
SourceProfile="File://text.icc" RenderingIntent="Perceptual"/>
      </ColorSpaceConversionParams>
      <ColorPool ID="Link0005" Class="Parameter" Status="Available">
        <Color CMYK="1 0 0 0" Name="Cyan"/>
        <Color CMYK="0 1 0 0" Name="Magenta"/>
        <Color CMYK="0 0 1 0" Name="Yellow"/>
        <Color CMYK="0 0 0 1" Name="Black"/>
        <Color CMYK="0.8 0.8 0 0" Name="Blue"/>
      </ColorPool>
      <ColorantControl ID="Link0006" Class="Parameter" rRefs="Link0005" Status="Available" Process-
ColorModel="DeviceCMYK">
        <ColorPoolRef rRef="Link0005"/>
      </ColorantControl>
      <RunList ID="Link0007" Class="Parameter" Status="Available" Pages="0~-1">
        <LayoutElement>
          <FileSpec URL="File://out/colortest.pdf"/>
        </LayoutElement>
      </RunList>
    </ResourcePool>
    <ResourceLinkPool>
      <RunListLink rRef="Link0003" Usage="Input"/>
      <ColorSpaceConversionParamsLink rRef="Link0004" Usage="Input"/>
      <ColorPoolLink rRef="Link0005" Usage="Input"/>
      <ColorantControlLink rRef="Link0006" Usage="Input"/>
      <RunListLink rRef="Link0007" Usage="Output"/>
    </ResourceLinkPool>
    <AuditPool>
      <Created Author="Rainer's JDFWriter 0.2000" TimeStamp="2000-11-01T10:26:11+01:00"/>
      <Modified Author="EatJDF Complete: task=*" TimeStamp="2000-11-01T10:26:57+01:00"/>
      <PhaseTime End="2000-11-01T10:26:57+01:00" Start="2000-11-01T10:26:57+01:00" Status="Setup"
TimeStamp="2000-11-01T10:26:57+01:00"/>
      <PhaseTime End="2000-11-01T10:26:57+01:00" Start="2000-11-01T10:26:57+01:00"
Status="InProgress" TimeStamp="2000-11-01T10:26:57+01:00"/>
      <PhaseTime End="2000-11-01T10:26:57+01:00" Start="2000-11-01T10:26:57+01:00" Status="Cleanup"
TimeStamp="2000-11-01T10:26:57+01:00"/>
      <ProcessRun End="2000-11-01T10:26:57+01:00" Start="2000-11-01T10:26:57+01:00" End-
Status="Completed" TimeStamp="2000-11-01T10:26:57+01:00"/>
    </AuditPool>
</JDF>
```

K.2 Product JDF

The following example describe a pair of college textbooks, one teachers edition and one students edition as product intent. Most intent resources are intentionally left empty.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="bookTest" Type="Product" JobID="bookJob" Status="Waiting" Version="1.1"
xmlns="http://www.CIP4.org/JDFSchema_1_1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <ResourcePool>
    <Component ID="Link0003" Class="Quantity" Amount="100" Status="Unavailable" Descriptive-
Name="Teacher's Book"/>
    <Component ID="Link0005" Class="Quantity" Amount="2000" Status="Unavailable" Descriptive-
Name="Cover">
      <!--This cover is reused by both-->
    </Component>
    <LayoutIntent ID="Link0006" Class="Intent" Status="Available">
      <Dimensions Range="576 756~648 828" DataType="NumberSpan" Preferred="612 792"/>
    </LayoutIntent>
    <LayoutIntent ID="Link0008" Class="Intent" Status="Available">
      <Dimensions Range="576 756~648 828" DataType="NumberSpan" Preferred="612 792"/>
      <Pages DataType="IntegerSpan" Preferred="240"/>
```

```
        </LayoutIntent>
        <Component ID="Link0011" Class="Quantity" Amount="1000" Status="Unavailable" Descriptive-
Name="Student's Book">
          <!--Students Book Intent-->
        </Component>
        <LayoutIntent ID="Link0014" Class="Intent" Status="Available">
          <Dimensions Range="576 756~648 828" DataType="NumberSpan" Preferred="612 792"/>
          <Pages DataType="IntegerSpan" Preferred="198"/>
        </LayoutIntent>
      </ResourcePool>
      <AuditPool>
        <Created Author="Rainer's JDFWriter 0.2000" TimeStamp="2000-11-01T12:46:56+01:00"/>
      </AuditPool>
      <JDF ID="Link0002" Type="Product" Status="waiting" JobPartID="0" DescriptiveName="Teacher's
Edition">
        <ResourcePool>
          <Component ID="Link0009" Class="Quantity" Amount="100" Status="Unavailable" Descriptive-
Name="Insert"/>
        </ResourcePool>
        <ResourceLinkPool>
          <ComponentLink rRef="Link0003" Usage="Output" Amount="100"/>
          <ComponentLink rRef="Link0009" Usage="Input" Amount="100"/>
          <ComponentLink rRef="Link0005" Usage="Input" Amount="100"/>
        </ResourceLinkPool>
        <JDF ID="Link0007" Type="Product" Status="waiting" JobPartID="2" DescriptiveName="Teacher's
Insert">
          <ResourceLinkPool>
            <LayoutIntentLink rRef="Link0008" Usage="Input"/>
            <ComponentLink rRef="Link0009" Usage="Output" Amount="100"/>
          </ResourceLinkPool>
        </JDF>
      </JDF>
      <JDF ID="Link0004" Type="Product" Status="waiting" JobPartID="1" DescriptiveName="Cover">
        <ResourceLinkPool>
          <ComponentLink rRef="Link0005" Usage="Output" Amount="2000"/>
          <LayoutIntentLink rRef="Link0006" Usage="Input"/>
        </ResourceLinkPool>
      </JDF>
      <JDF ID="Link0010" Type="Product" Status="waiting" JobPartID="3" DescriptiveName="Student's
Edition">
        <ResourcePool>
          <Component ID="Link0013" Class="Quantity" Amount="1000" Status="Unavailable" Descriptive-
Name="Insert"/>
        </ResourcePool>
        <ResourceLinkPool>
          <ComponentLink rRef="Link0011" Usage="Output" Amount="1000"/>
          <ComponentLink rRef="Link0013" Usage="Input" Amount="1000"/>
          <ComponentLink rRef="Link0005" Usage="Input" Amount="1000"/>
        </ResourceLinkPool>
        <JDF ID="Link0012" Type="Product" Status="waiting" JobPartID="4" DescriptiveName="Student's
Insert">
          <ResourceLinkPool>
            <ComponentLink rRef="Link0013" Usage="Output" Amount="1000"/>
            <LayoutIntentLink rRef="Link0014" Usage="Input"/>
          </ResourceLinkPool>
        </JDF>
      </JDF>
  </JDF>
</JDF>
```

K.3 Spawning and Merging

The following set of examples show a JDF job in the relevant stages of spawning and merging. One example defines a simple brochure with a cover and an insert. The red node, which defines the cover, is spawned, modified, and subsequently merged. Blue elements represent metadata that apply to spawning and merging.

K.3.1 Example 2 Component JDF before Spawning

The following JDF file describes a two-component brochure. The resources are not fleshed out.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="SpawnTest" Type="Product" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting"
Version="1.1" JobPartID="Part1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <AuditPool>
    <Created Author="CIP4 JDFWriter 1.0.01 beta" TimeStamp="2002-04-05T15:27:58+02:00"/>
  </AuditPool>
  <ResourcePool>
    <Component ID="r0043" Class="Quantity" Amount="10000" Status="Unavailable"/>
    <BindingIntent ID="r0044" Class="Intent" Status="Available"/>
    <ProductionIntent ID="r0045" Class="Intent" Status="Available">
      <PrintProcess Range="Gravure" DataType="EnumerationSpan"/>
    </ProductionIntent>
    <Component ID="r0047" Class="Quantity" Status="Unavailable"/>
    <Component ID="r0051" Class="Quantity" Status="Unavailable"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="r0043" Usage="Output"/>
    <BindingIntentLink rRef="r0044" Usage="Input"/>
    <ProductionIntentLink rRef="r0045" Usage="Input"/>
    <ComponentLink rRef="r0047" Usage="Input"/>
    <ComponentLink rRef="r0051" Usage="Input"/>
  </ResourceLinkPool>
  <JDF ID="n0046" Type="Product" Status="Waiting" JobPartID="Part2" DescriptiveName="Cover">
    <ResourceLinkPool>
      <ComponentLink rRef="r0047" Usage="Output"/>
      <LayoutIntentLink rRef="r0048" Usage="Input"/>
      <ColorIntentLink rRef="r0049" Usage="Input"/>
    </ResourceLinkPool>
    <ResourcePool>
      <LayoutIntent ID="r0048" Class="Intent" Status="Available"/>
      <ColorIntent ID="r0049" Class="Intent" Status="Available"/>
    </ResourcePool>
  </JDF>
  <JDF ID="n0050" Type="Product" Status="Waiting" JobPartID="Part3" DescriptiveName="Insert">
    <ResourceLinkPool>
      <ComponentLink rRef="r0051" Usage="Output"/>
      <LayoutIntentLink rRef="r0052" Usage="Input"/>
      <ColorIntentLink rRef="r0053" Usage="Input"/>
    </ResourceLinkPool>
    <ResourcePool>
      <LayoutIntent ID="r0052" Class="Intent" Status="Available"/>
      <ColorIntent ID="r0053" Class="Intent" Status="Available"/>
    </ResourcePool>
  </JDF>
</JDF>
```

K.3.2 Example 2 Component JDF Parent after spawning the cover node

The following JDF is the parent JDF after spawning. The **Component** that describes the cover is marked as *SpawnedRW*, since it was copied into the spawned node and may be modified. A Spawned audit was inserted into the Cover nodes parent's AuditPool, and the Spawned node itself has a *Status* of *Spawned*.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="SpawnTest" Type="Product" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting"
Version="1.1" JobPartID="Part1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <AuditPool>
    <Created Author="CIP4 JDFWriter 1.0.01 beta" TimeStamp="2002-04-05T15:27:58+02:00"/>
    <Spawned URL="File://spawn.jdf" jRef="n0046" TimeStamp="2002-04-05T15:34:43+02:00" NewS-
pawnID="Sp0057" rRefsRWCopied="r0047"/>
  </AuditPool>
  <ResourcePool>
    <Component ID="r0043" Class="Quantity" Amount="10000" Status="Unavailable"/>
    <BindingIntent ID="r0044" Class="Intent" Status="Available"/>
    <ProductionIntent ID="r0045" Class="Intent" Status="Available">
      <PrintProcess Range="Gravure" DataType="EnumerationSpan"/>
    </ProductionIntent>
    <Component ID="r0047" Class="Quantity" Status="Unavailable" SpawnIDs="Sp0057" Spawn-
Status="SpawnedRW"/>
```

```
      <Component ID="r0051" Class="Quantity" Status="Unavailable"/>
    </ResourcePool>
    <ResourceLinkPool>
      <ComponentLink rRef="r0043" Usage="Output"/>
      <BindingIntentLink rRef="r0044" Usage="Input"/>
      <ProductionIntentLink rRef="r0045" Usage="Input"/>
      <ComponentLink rRef="r0047" Usage="Input"/>
      <ComponentLink rRef="r0051" Usage="Input"/>
    </ResourceLinkPool>
    <JDF ID="n0046" Type="Product" Status="Spawned" JobPartID="Part2" DescriptiveName="Cover">
      <ResourceLinkPool>
        <ComponentLink rRef="r0047" Usage="Output"/>
        <LayoutIntentLink rRef="r0048" Usage="Input"/>
        <ColorIntentLink rRef="r0049" Usage="Input"/>
      </ResourceLinkPool>
      <ResourcePool>
        <LayoutIntent ID="r0048" Class="Intent"  Status="Available" SpawnIDs="Sp0057" Spawn-
Status="SpawnedRO"/>
        <ColorIntent ID="r0049" Class="Intent"  Status="Available" SpawnIDs="Sp0057" Spawn-
Status="SpawnedRO"/>
      </ResourcePool>
    </JDF>
    <JDF ID="n0050" Type="Product" Status="Waiting" JobPartID="Part3" DescriptiveName="Insert">
      <ResourceLinkPool>
        <ComponentLink rRef="r0051" Usage="Output"/>
        <LayoutIntentLink rRef="r0052" Usage="Input"/>
        <ColorIntentLink rRef="r0053" Usage="Input"/>
      </ResourceLinkPool>
      <ResourcePool>
        <LayoutIntent ID="r0052" Class="Intent"  Status="Available"/>
        <ColorIntent ID="r0053" Class="Intent"  Status="Available"/>
      </ResourcePool>
    </JDF>
    <AncestorPool/>
</JDF>
```

K.3.3 Example 2 Component JDF spawned node

The Component that represents the cover was copied into the spawned node, since it is the output resource. It is not locked, since it was spawned in RW mode. The existence of an AncestorPool denotes the node as spawned and defines the parent node.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="n0046" Type="Product" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting"
SpawnID="Sp0057" Version="1.1" JobPartID="Part2" DescriptiveName="Cover">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <AuditPool>
    <Created Author="CIP4 JDFWriter 1.0.01 beta" TimeStamp="2002-04-05T15:34:43+02:00"/>
  </AuditPool>
  <ResourceLinkPool>
    <ComponentLink rRef="r0047" Usage="Output"/>
    <LayoutIntentLink rRef="r0048" Usage="Input"/>
    <ColorIntentLink rRef="r0049" Usage="Input"/>
  </ResourceLinkPool>
  <ResourcePool>
    <LayoutIntent ID="r0048" Class="Intent" Status="Available"/>
    <ColorIntent ID="r0049" Class="Intent" Status="Available"/>
    <Component ID="r0047" Class="Quantity"  Status="Available" SpawnIDs="Sp0057"/>
  </ResourcePool>
  <AncestorPool>
    <Ancestor NodeID="SpawnTest" FileName="testjdf4.jdf"/>
  </AncestorPool>
</JDF>
```

K.3.4 Example 2 Component JDF after merging

In this example, it is assumed that the cover output component was created by some processor that processed the spawned node. This resulted in the Component becoming available. The Component was also removed from the copy of the spawned node, since it would otherwise exist twice.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="SpawnTest" Type="Product" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting"
Version="1.1" JobPartID="Part1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <AuditPool>
    <Created Author="CIP4 JDFWriter 1.0.01 beta" TimeStamp="2002-04-05T15:27:58+02:00"/>
    <Spawned URL="File://spawn.jdf" jRef="n0046" TimeStamp="2002-04-05T15:34:43+02:00" NewS-
pawnID="Sp0057" rRefsRWCopied="r0047"/>
    <Merged URL="File://spawn.jdf" jRef="n0046" MergeID="Sp0057" TimeStamp="2002-04-
05T15:40:20+02:00" rRefsOverwritten="r0047"/>
  </AuditPool>
  <ResourcePool>
    <Component ID="r0043" Class="Quantity" Amount="10000"  Status="Unavailable"/>
    <BindingIntent ID="r0044" Class="Intent"  Status="Available"/>
    <ProductionIntent ID="r0045" Class="Intent"  Status="Available">
      <PrintProcess Range="Gravure" DataType="EnumerationSpan"/>
    </ProductionIntent>
    <Component ID="r0047" Class="Quantity" Status="Available"/>
    <Component ID="r0051" Class="Quantity"  Status="Unavailable"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ComponentLink rRef="r0043" Usage="Output"/>
    <BindingIntentLink rRef="r0044" Usage="Input"/>
    <ProductionIntentLink rRef="r0045" Usage="Input"/>
    <ComponentLink rRef="r0047" Usage="Input"/>
    <ComponentLink rRef="r0051" Usage="Input"/>
  </ResourceLinkPool>
  <JDF ID="n0046" Type="Product" xmlns="http://www.CIP4.org/JDFSchema_1_1" Status="Waiting" Ver-
sion="1.1" JobPartID="Part2" DescriptiveName="Cover">
    <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
    <AuditPool>
      <Created Author="CIP4 JDFWriter 1.0.01 beta" TimeStamp="2002-04-05T15:34:43+02:00"/>
    </AuditPool>
    <ResourceLinkPool>
      <ComponentLink rRef="r0047" Usage="Output"/>
      <LayoutIntentLink rRef="r0048" Usage="Input"/>
      <ColorIntentLink rRef="r0049" Usage="Input"/>
    </ResourceLinkPool>
    <ResourcePool>
      <LayoutIntent ID="r0048" Class="Intent" Status="Available"/>
      <ColorIntent ID="r0049" Class="Intent" Status="Available"/>
    </ResourcePool>
  </JDF>
  <JDF ID="n0050" Type="Product" Status="Waiting" JobPartID="Part3" DescriptiveName="Insert">
    <ResourceLinkPool>
      <ComponentLink rRef="r0051" Usage="Output"/>
      <LayoutIntentLink rRef="r0052" Usage="Input"/>
      <ColorIntentLink rRef="r0053" Usage="Input"/>
    </ResourceLinkPool>
    <ResourcePool>
      <LayoutIntent ID="r0052" Class="Intent" Status="Available"/>
      <ColorIntent ID="r0053" Class="Intent" Status="Available"/>
    </ResourcePool>
  </JDF>
  <AncestorPool/>
</JDF>
```

K.3.5 Example of a Partitioned ImageSetting Node before Spawning

The following example shows a simple ImageSetting node that is partitioned by Separation. The resources are not filled with data. The input resources are Available.

```xml
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="n20020701190951" Type="ImageSetting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Waiting" Version="1.1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.1.01 beta-->
  <ResourcePool>
    <ImageSetterParams ID="r0052" Class="Parameter" Locked="false" Status="Available"/>
```

```
    <Media ID="r0053" Class="Consumable" Locked="false" Status="Available" PartID-
Keys="Separation">
      <Media Separation="Cyan"/>
      <Media Separation="Magenta"/>
      <Media Separation="Yellow"/>
      <Media Separation="Black"/>
    </Media>
    <ExposedMedia ID="r0054" Class="Handling" Locked="false" Status="Unavailable" PartID-
Keys="Separation">
      <ExposedMedia Separation="Cyan"/>
      <ExposedMedia Separation="Magenta"/>
      <ExposedMedia Separation="Yellow"/>
      <ExposedMedia Separation="Black"/>
    </ExposedMedia>
    <RunList ID="r0055" Class="Parameter" Locked="false" Status="Available"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ImageSetterParamsLink rRef="r0052" Usage="Input"/>
    <MediaLink rRef="r0053" Usage="Input"/>
    <ExposedMediaLink rRef="r0054" Usage="Output"/>
    <RunListLink rRef="r0055" Usage="Input"/>
  </ResourceLinkPool>
</JDF>
```

K.3.6 The Spawned Cyan Partition of the ImageSetting Node

The following example shows the spawned Cyan partition of the ImageSetting node from the previous example.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="n20020701190951" Type="ImageSetting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Waiting" SpawnID="Sp0059" Version="1.1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.1.01 beta-->
  <AuditPool/>
  <ResourcePool>
    <ImageSetterParams ID="r0052" Class="Parameter" Locked="true" Status="Available"/>
    <Media ID="r0053" Class="Consumable" Locked="true" Status="Available" PartID-
Keys="Separation">
      <Media Separation="Cyan"/>
    </Media>
    <ExposedMedia ID="r0054" Class="Handling" Locked="true" Status="Unavailable" PartID-
Keys="Separation">
      <ExposedMedia Separation="Cyan"/>
    </ExposedMedia>
    <RunList ID="r0055" Class="Parameter" Locked="true" Status="Available"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ImageSetterParamsLink rRef="r0052" Usage="Input"/>
    <MediaLink rRef="r0053" Usage="Input">
      <Part Separation="Cyan"/>
    </MediaLink>
    <ExposedMediaLink rRef="r0054" Usage="Output">
      <Part Separation="Cyan"/>
    </ExposedMediaLink>
    <RunListLink rRef="r0055" Usage="Input"/>
  </ResourceLinkPool>
  <AncestorPool>
    <Part Separation="Cyan"/>
    <Ancestor Type="ImageSetting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
NodeID="n20020701190951" Status="Waiting" Version="1.1" FileName="testjdf5.jdf"/>
  </AncestorPool>
</JDF>
```

K.3.7 The Root Partitioned ImageSetting Node after Spawning

Note …

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="n20020701190951" Type="ImageSetting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Pool" Version="1.1">
```

```
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.1.01 beta-->
  <AuditPool>
    <Spawned URL="File://spawnIS.jdf" jRef="n20020701190951" Status="Waiting" TimeStamp="2002-07-
01T19:18:03+02:00" NewSpawnID="Sp0059">
      <Part Separation="Cyan"/>
    </Spawned>
  </AuditPool>
  <ResourcePool>
    <ImageSetterParams ID="r0052" Class="Parameter" Locked="false" Status="Available"
SpawnIDs="Sp0059" SpawnStatus="SpawnedRO"/>
    <Media ID="r0053" Class="Consumable" Locked="false" Status="Available" SpawnIDs="Sp0059" Par-
tIDKeys="Separation">
      <Media Locked="true" Separation="Cyan" SpawnStatus="SpawnedRW"/>
      <Media Separation="Magenta"/>
      <Media Separation="Yellow"/>
      <Media Separation="Black"/>
    </Media>
    <ExposedMedia ID="r0054" Class="Handling" Locked="false" Status="Unavailable"
SpawnIDs="Sp0059" PartIDKeys="Separation">
      <ExposedMedia Locked="true" Separation="Cyan" SpawnStatus="SpawnedRW"/>
      <ExposedMedia Separation="Magenta"/>
      <ExposedMedia Separation="Yellow"/>
      <ExposedMedia Separation="Black"/>
    </ExposedMedia>
    <RunList ID="r0055" Class="Parameter" Locked="false" Status="Available" SpawnIDs="Sp0059"
SpawnStatus="SpawnedRO"/>
  </ResourcePool>
  <ResourceLinkPool>
    <ImageSetterParamsLink rRef="r0052" Usage="Input"/>
    <MediaLink rRef="r0053" Usage="Input"/>
    <ExposedMediaLink rRef="r0054" Usage="Output"/>
    <RunListLink rRef="r0055" Usage="Input"/>
  </ResourceLinkPool>
  <StatusPool Status="Waiting">
    <PartStatus Status="Spawned">
      <Part Separation="Cyan"/>
    </PartStatus>
  </StatusPool>
</JDF>
```

K.3.8 The Merged ImageSetting Node

The Node has now been executed and merged.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="n20020701190951" Type="ImageSetting" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Pool" Version="1.1">
  <AuditPool>
    <Spawned URL="File://spawnIS.jdf" jRef="n20020701190951" Status="Waiting" TimeStamp="2002-07-
01T20:25:03+02:00" NewSpawnID="Sp0059">
      <Part Separation="Cyan"/>
    </Spawned>
    <Merged URL="File://spawnIS2.jdf" jRef="n20020701190951" MergeID="Sp0059" TimeStamp="2002-07-
01T20:27:51+02:00">
      <Part Separation="Cyan"/>
    </Merged>
  </AuditPool>
  <ResourcePool>
    <ImageSetterParams ID="r0052" Class="Parameter" Status="Available"/>
    <Media ID="r0053" Class="Consumable" Status="Available" PartIDKeys="Separation">
      <Media Separation="Cyan" Status="Unavailable"/>
      <Media Separation="Magenta"/>
      <Media Separation="Yellow"/>
      <Media Separation="Black"/>
    </Media>
    <ExposedMedia ID="r0054" Class="Handling" Status="Unavailable" PartIDKeys="Separation">
      <ExposedMedia Status="Available" Separation="Cyan"/>
      <ExposedMedia Separation="Magenta"/>
      <ExposedMedia Separation="Yellow"/>
      <ExposedMedia Separation="Black"/>
    </ExposedMedia>
```

```
      <RunList ID="r0055" Class="Parameter" Status="Available"/>
   </ResourcePool>
   <ResourceLinkPool>
      <ImageSetterParamsLink rRef="r0052" Usage="Input"/>
      <MediaLink rRef="r0053" Usage="Input"/>
      <ExposedMediaLink rRef="r0054" Usage="Output"/>
      <RunListLink rRef="r0055" Usage="Input"/>
   </ResourceLinkPool>
   <StatusPool Status="Waiting">
      <PartStatus Status="Completed">
        <Part Separation="Cyan"/>
      </PartStatus>
   </StatusPool>
</JDF>
```

K.4 Conversion of PJTF to JDF

K.4.1 PJTF input

The following code defines 4-up duplex impositioning of a 17 page pdf document in Adobe PJTF format:

```
%JTF-1.2
1 0 obj
<<
/A [ 3 0 R ]
/V 1.1
/Cn [ 2 0 R ]
>>
endobj
2 0 obj
<<
/Type /JobTicketContents
/D [ 6 0 R ]
/PL 8 0 R
>>
endobj
3 0 obj
<<
/D (D:19991111173640)
/JTM (Default JT Creator)
/C (JT created)
>>
endobj
4 0 obj
<<
/Type /Catalog
/JT 1 0 R
>>
endobj
5 0 obj
<<
/Producer (HD PDFWrite vs. 0.1)
>>
endobj
6 0 obj
<<
/Fi [ 7 0 R ]
>>
endobj
7 0 obj
<<
/Fi (panrt17a.pdf)
>>
endobj
8 0 obj
<<
/Si 9 0 R
```

```
>>
endobj
9 0 obj
<<
/S 10 0 R
>>
endobj
10 0 obj
[ 11 0 R ]
endobj
11 0 obj
<<
/MS
<<
/Cl (sheet of paper)
/Me 12 0 R
>>
/Fr 13 0 R
/B 18 0 R
>>
endobj
12 0 obj
<<
/Dm [ 842 1191 842 1191 ]
>>
endobj
13 0 obj
<<
/PO [ 14 0 R 15 0 R 16 0 R 17 0 R ]
>>
endobj
14 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 624 ]
/O 0
/Cl [ 21 624 399 1159 ]
>>
endobj
15 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 624 ]
/O 1
/Cl [ 442 624 820 1159 ]
>>
endobj
16 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 29 ]
/O 2
/Cl [ 21 29 399 564 ]
>>
endobj
17 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 29 ]
/O 3
/Cl [ 442 29 820 564 ]
>>
endobj
18 0 obj
<<
/PO [ 19 0 R 20 0 R 21 0 R 22 0 R ]
>>
endobj
19 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 624 ]
/O 4
/Cl [ 21 624 399 1159 ]
>>
endobj
```

```
20 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 624 ]
/O 5
/Cl [ 442 624 820 1159 ]
>>
endobj
21 0 obj
<<
/CTM [ 0.45 0 0 0.45 21 29 ]
/O 6
/Cl [ 21 29 399 564 ]
>>
endobj
22 0 obj
<<
/CTM [ 0.45 0 0 0.45 442 29 ]
/O 7
/Cl [ 442 29 820 564 ]
>>
endobj
xref
0 23
0000000000 65535 f
0000000009 00000 n
0000000071 00000 n
0000000146 00000 n
0000000233 00000 n
0000000283 00000 n
0000000338 00000 n
0000000377 00000 n
0000000419 00000 n
0000000453 00000 n
0000000487 00000 n
0000000516 00000 n
0000000608 00000 n
0000000660 00000 n
0000000722 00000 n
0000000810 00000 n
0000000900 00000 n
0000000985 00000 n
0000001072 00000 n
0000001134 00000 n
0000001222 00000 n
0000001312 00000 n
0000001397 00000 n
trailer
<<
/Root 4 0 R
/Info 5 0 R
/Size 23
>>
startxref
1484
%%EOF
```

K.4.2 JDF output

This JDF file describes the Imposition process defined by the PJTF file.

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="PJTFJob" Type="Impositioning" JobID="Job" Status="Waiting"
xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <NodeInfo/>
  <ResourcePool>
    <Layout ID="Link0002" Class="Parameter" Status="Available">
      <Signature ID="Cos9">
        <SheetRef rRef="Cos11"/>
      </Signature>
    </Layout>
```

```
    <Surface ID="Cos13" Side="Front">
      <ContentObject ID="Cos14" CTM="0.45 0 0 0.45 21 624" Ord="0" ClipBox="21 624 399 1159"/>
      <ContentObject ID="Cos15" CTM="0.45 0 0 0.45 442 624" Ord="1" ClipBox="442 624 820 1159"/>
      <ContentObject ID="Cos16" CTM="0.45 0 0 0.45 21 29" Ord="2" ClipBox="21 29 399 564"/>
      <ContentObject ID="Cos17" CTM="0.45 0 0 0.45 442 29" Ord="3" ClipBox="442 29 820 564"/>
    </Surface>
    <Surface ID="Cos18" Side="Back">
      <ContentObject ID="Cos19" CTM="0.45 0 0 0.45 21 624" Ord="4" ClipBox="21 624 399 1159"/>
      <ContentObject ID="Cos20" CTM="0.45 0 0 0.45 442 624" Ord="5" ClipBox="442 624 820 1159"/>
      <ContentObject ID="Cos21" CTM="0.45 0 0 0.45 21 29" Ord="6" ClipBox="21 29 399 564"/>
      <ContentObject ID="Cos22" CTM="0.45 0 0 0.45 442 29" Ord="7" ClipBox="442 29 820 564"/>
    </Surface>
    <Sheet ID="Cos11" rRefs="Cos18 Cos13">
      <SurfaceRef rRef="Cos18"/>
      <SurfaceRef rRef="Cos13"/>
    </Sheet>
    <Media ID="Cos12" Dimensions="842 1191 842 1191"/>
    <RunList ID="Link0003" Class="Parameter" NPage="17" Status="Available" Pages="0~16">
      <LayoutElement>
        <FileSpec URL="File://panrt17a.pdf"/>
      </LayoutElement>
    </RunList>
    <RunList ID="Link0004" Class="Parameter" Status="Unavailable"/>
  </ResourcePool>
  <ResourceLinkPool>
    <RunListLink rRef="Link0003" Usage="Input"/>
    <LayoutLink rRef="Link0002" Usage="Input"/>
    <RunListLink rRef="Link0004" Usage="Output"/>
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="PJTF2JDF" TimeStamp="2000-11-07T17:42:15+01:00"/>
  </AuditPool>
</JDF>
```

K.5 Conversion of PPF to JDF

Simple example of a PPF.

```
%!PS-Adobe-3.0
%%CIP3-File Version 2.0

CIP3BeginSheet
(This example was manually created by Stefan Daun) CIP3Comment
/CIP3AdmJobName (8 pages with workturn and 5 color separations) def
/CIP3AdmSoftware (Text editor) def
/CIP3AdmCreationTime (Wed Feb 19 12:00:00 1997) def
/CIP3AdmArtist (Joerg Zedler) def
/CIP3AdmCopyright (Copyright by Fraunhofer-IGD, 1997) def
/CIP3AdmSheetName (E08P5C) def
/CIP3AdmSheetLay /Left def
/CIP3AdmPSExtent [ 40 inch 27 inch ] def
/CIP3TransferFilmCurveData [0.0 0.0 1.0 1.0 ] def
/CIP3TransferPlateCurveData [0.0 0.0 1.0 1.0 ] def
/CIP3AdmFilmTrf [0 1 -1 0 1944 0] def
/CIP3AdmPlateTrf [0 -1 1 0 0 2880] def
CIP3BeginFront
/CIP3AdmSeparationNames [ (Cyan) (Magenta) (Yellow) (Black) (Pantone Green CV)] def

CIP3BeginPreviewImage

CIP3BeginSeparation
(First separation of Front) CIP3Comment
/CIP3PreviewImageWidth 2030 def
/CIP3PreviewImageHeight 1370 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [0 1370 -2030 0 1370 0] def
/CIP3PreviewImageResolution [50.75 50.75] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageByteAlign 4 def
```

```
CIP3PreviewImage
... <image data>
CIP3EndSeparation

CIP3BeginSeparation
(Second separation of Front) CIP3Comment
/CIP3PreviewImageWidth 2030 def
/CIP3PreviewImageHeight 1370 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [0 1370 -2030 0 1370 0] def
/CIP3PreviewImageResolution [50.75 50.75] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageByteAlign 4 def
CIP3PreviewImage
... <image data>
CIP3EndSeparation

CIP3BeginSeparation
(Fourth separation of Front) CIP3Comment
/CIP3PreviewImageWidth 2030 def
/CIP3PreviewImageHeight 1370 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [0 1370 -2030 0 1370 0] def
/CIP3PreviewImageResolution [50.75 50.75] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageByteAlign 4 def
CIP3PreviewImage
... <image data>
CIP3EndSeparation

CIP3BeginSeparation
(Fifth separation of Front) CIP3Comment
/CIP3PreviewImageWidth 2030 def
/CIP3PreviewImageHeight 1370 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [0 1370 -2030 0 1370 0] def
/CIP3PreviewImageResolution [50.75 50.75] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageByteAlign 4 def
CIP3PreviewImage

CIP3BeginSeparation
(Second separation of Front) CIP3Comment
/CIP3PreviewImageWidth 2030 def
/CIP3PreviewImageHeight 1370 def
/CIP3PreviewImageBitsPerComp 8 def
/CIP3PreviewImageComponents 1 def
/CIP3PreviewImageMatrix [0 1370 -2030 0 1370 0] def
/CIP3PreviewImageResolution [50.75 50.75] def
/CIP3PreviewImageEncoding /Binary def
/CIP3PreviewImageCompression /RunLengthDecode def
/CIP3PreviewImageByteAlign 4 def
CIP3PreviewImage
... <image data>
CIP3EndSeparation
CIP3EndSeparation
CIP3EndPreviewImage

CIP3BeginRegisterMarks
20 inch 0 0 /cross&circle CIP3PlaceRegisterMark
CIP3EndRegisterMarks

CIP3BeginColorControl
/C100        << /CIE-L*  62 /CIE-a* -31 /CIE-b* -48 /Diameter 4.7 mm /Light /D65 /Observer 2
/Tolerance 5 /Type /CIELAB >> def
```

```
/M100        << /CIE-L*  48 /CIE-a*  83 /CIE-b*  -3 /Diameter 4.7 mm /Light /D65 /Observer 2
/Tolerance 5 /Type /CIELAB >> def
/Y100        << /CIE-L*  94 /CIE-a* -14 /CIE-b* 100 /Diameter 4.7 mm /Light /D65 /Observer 2
/Tolerance 5 /Type /CIELAB >> def
/K100        << /CIE-L*   0 /CIE-a*   0 /CIE-b*   0 /Diameter 4.7 mm /Light /D65 /Observer 2
/Tolerance 5 /Type /CIELAB >> def
0 0 0 360 18
[
  [  14.77 0 C100 ]
  [  41.85 0 Y100 ]
  [  68.92 0 M100 ]
  [ 177.23 0 K100 ]

] /PrepsColorBar CIP3PlaceColorControlStrip
CIP3EndColorControl

CIP3BeginCutData
CIP3BeginCutBlock
/CIP3BlockTrf [1 0 0 1 44 mm 45.9 mm] def
/CIP3BlockSize [ 420 mm 594 mm] def
/CIP3BlockType /CutBlock def
/CIP3BlockName (Front Sides) def
/CIP3BlockFoldingProcedure /F08-07_li_2x2_1 def
CIP3EndCutBlock

CIP3BeginCutBlock
/CIP3BlockTrf [1 0 0 1 552 mm 45.9 mm] def
/CIP3BlockSize [ 420 mm 594 mm] def
/CIP3BlockType /CutBlock def
/CIP3BlockName (Back Sides) def
/CIP3BlockFoldingProcedure /F08-07_li_2x2_1 def
400 400 /RightHorizontalCutMark CIP3PlaceCutMark
CIP3EndCutBlock
100 200 /TopVerticalCutMark CIP3PlaceCutMark
CIP3EndCutData
CIP3BeginFoldProcedures
/F08-07_li_2x2_1 <<
        /CIP3FoldDescription (F8-7)
        /CIP3FoldSheetIn [210 mm 297 mm]
        /CIP3FoldProc
        [
                297.638 /Front /Up Fold
                420.945 /Left /Up Fold
        ]
        >> def
CIP3EndFoldProcedures
CIP3EndFront
CIP3EndSheet
%%CIP3EndOfFile
```

The translated JDF:

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="PPFJDF" Type="Product" JobID="MyJob" xmlns="http://www.CIP4.org/JDFSchema_1_1"
Status="Waiting" Version="1.1">
  <!--Generated by the CIP4 C++ open source JDF Library version CIP4 JDFWriter 1.0.01 beta-->
  <JDF ID="n1152" Type="InkZoneCalculation" Status="Waiting">
    <ResourceLinkPool>
      <LayoutLink rRef="r1106" Usage="Input"/>
      <PreviewLink rRef="r1116" Usage="Input"/>
      <TransferCurvePoolLink rRef="r1111" Usage="Input"/>
      <InkZoneCalculationParamsLink rRef="r1118" Usage="Input"/>
      <InkZoneProfileLink rRef="r1119" Usage="Output"/>
    </ResourceLinkPool>
    <ResourcePool>
      <Layout ID="r1106" Class="Parameter" rRefs="r1107"  Status="Available">
        <Signature Name="HDM">
          <SheetRef rRef="r1107"/>
        </Signature>
      </Layout>
```

```
      <Sheet ID="r1107" Name="E08P5C" Class="Parameter" rRefs="r1112"  Status="Unavailable" Sur-
faceContentsBox="0 0 2880 1944">
         <SurfaceRef rRef="r1112"/>
      </Sheet>
      <Surface ID="r1112" Side="Front" Class="Parameter" rRefs="r1114 r1115 r1130 r1134"
Status="Unavailable">
         <MarkObject CTM="1 0 0 1 0 0" Type="Mark">
           <ColorControlStripRef rRef="r1114"/>
         </MarkObject>
         <MarkObject CTM="1 0 0 1 0 0">
           <RegisterMarkRef rRef="r1115"/>
         </MarkObject>
         <MarkObject CTM="1 0 0 1 0 0" Type="Mark">
           <CutMarkRef rRef="r1130"/>
         </MarkObject>
         <MarkObject CTM="1 0 0 1 0 0" Type="Mark">
           <CutMarkRef rRef="r1134"/>
         </MarkObject>
      </Surface>
      <ColorControlStrip ID="r1114" Size="360 18" Class="Parameter" Center="0 0"
Status="Unavailable" Rotation="0">
         <CIELABMeasuringField rRefs="RCMC" Center="14.77 0" CIE_Lab="62 -31 -48" Diame-
ter="13.3228346457" Observer="2" Tolerance="5">
            <ColorMeasurementConditionsRef rRef="RCMC"/>
         <CIELABMeasuringField(>
         <CIELABMeasuringField rRefs="RCMC" Center="41.85 0" CIE_Lab="94 -14 100" Diame-
ter="13.3228346457" Tolerance="5">
            <ColorMeasurementConditionsRef rRef="RCMC"/>
         <CIELABMeasuringField(>
         <CIELABMeasuringField rRefs="RCMC" Center="68.92 0" CIE_Lab="48 83 -3" Diame-
ter="13.3228346457" Tolerance="5">
            <ColorMeasurementConditionsRef rRef="RCMC"/>
         <CIELABMeasuringField(>
         <CIELABMeasuringField rRefs="RCMC" Center="177.23 0" CIE_Lab="0 0 0" Diame-
ter="13.3228346457" Tolerance="5">
            <ColorMeasurementConditionsRef rRef="RCMC"/>
         <CIELABMeasuringField(>
      </ColorControlStrip>
      <ColorMeasurementConditions ID="RCMC" Illumination="D65" Observer="2"/>
      <RegisterMark ID="r1115" Class="Parameter" Center="1440 0"  Status="Unavailable" Mark-
Type="cross&amp;circle" Rotation="0"/>
      <CutMark ID="r1130" Class="Parameter"  Status="Available" MarkType="TopVerticalCutMark" Po-
sition="100 200"/>
      <CutMark ID="r1134" Class="Parameter" Blocks="Back_Sides"  Status="Available" Mark-
Type="RightHorizontalCutMark" Position="400 400"/>
      <Preview ID="r1116" Class="Parameter"  Status="Available" PartIDKeys="SheetName Side Sepa-
ration" PreviewType="Separation">
         <Preview SheetName="E08P5C">
           <Preview Side="Front">
             <Preview URL="file://Bild0000.png" Separation="Cyan"/>
             <Preview URL="file://Bild0001.png" Separation="Magenta"/>
             <Preview URL="file://Bild0002.png" Separation="Yellow"/>
             <Preview URL="file://Bild0003.png" Separation="Black"/>
             <Preview URL="file://Bild0004.png" Separation="Pantone Green CV"/>
           </Preview>
         </Preview>
      </Preview>
      <TransferCurvePool ID="r1111" Class="Parameter"  Status="Available">
         <TransferCurveSet CTM="0 1 -1 0 1944 0" Name="Film">
           <TransferCurve Curve="0 0 1 1"/>
         </TransferCurveSet>
         <TransferCurveSet CTM="1 0 0 1 0 0" Name="Press">
           <TransferCurve Curve="0 0 1 1"/>
         </TransferCurveSet>
         <TransferCurveSet CTM="0 -1 1 0 0 2880" Name="Plate"/>
         <TransferCurveSet CTM="1 0 0 1 0 0" Name="Paper"/>
      </TransferCurvePool>
      <InkZoneCalculationParams ID="r1118" Class="Parameter"  Status="Available"/>
      <InkZoneProfile ID="r1119" Class="Parameter"  Status="Unavailable"/>
   </ResourcePool>
  </JDF>
```

```
  <JDF ID="n1153" Type="ConventionalPrinting" Status="Waiting">
    <ResourceLinkPool>
      <LayoutLink rRef="r1106" Usage="Input"/>
      <ColorantControlLink rRef="r1113" Usage="Input"/>
      <InkZoneProfileLink rRef="r1119" Usage="Input"/>
      <ComponentLink rRef="r1125" Usage="Output" ProcessUsage="Good"/>
      <MediaLink rRef="r1108" Usage="Input"/>
      <ConventionalPrintingParamsLink rRef="r1126" Usage="Input"/>
      <InkLink rRef="r1127" Usage="Input"/>
      <ExposedMediaLink rRef="r1123" Usage="Input"/>
    </ResourceLinkPool>
    <ResourcePool>
      <ColorantControl ID="r1113" Class="Parameter"  Status="Available" PartIDKeys="SheetName
Side" ProcessColorModel="DeviceCMYK">
        <ColorantControl SheetName="E08P5C">
          <ColorantControl Side="Front">
            <ColorantParams>
              <SeparationSpec Name="Pantone Green CV"/>
            </ColorantParams>
            <ColorantOrder>
              <SeparationSpec Name="Cyan"/>
              <SeparationSpec Name="Magenta"/>
              <SeparationSpec Name="Yellow"/>
              <SeparationSpec Name="Black"/>
              <SeparationSpec Name="Pantone Green CV"/>
            </ColorantOrder>
          </ColorantControl>
        </ColorantControl>
      </ColorantControl>
      <Component ID="r1125" Class="Quantity" rRefs="r1107"  Status="Unavailable" PartID-
Keys="SheetName">
        <Component SheetName="E08P5C" ComponentType="Sheet">
          <SheetRef rRef="r1107"/>
        </Component>
      </Component>
      <Media ID="r1108" Class="Consumable"  Status="Available" MediaType="Paper" PartID-
Keys="SheetName Side">
        <Media Dimension="2880 1944" SheetName="E08P5C">
          <Media Side="Front" Dimension="2880 1944"/>
        </Media>
      </Media>
      <ConventionalPrintingParams ID="r1126" Class="Parameter"  Status="Available" PartID-
Keys="SheetName Side">
        <ConventionalPrintingParams SheetLay="Left" SheetName="E08P5C">
          <ConventionalPrintingParams Side="Front"/>
        </ConventionalPrintingParams>
      </ConventionalPrintingParams>
      <Ink ID="r1127" Class="Consumable"  Status="Draft"/>
      <ExposedMedia ID="r1123" Class="Handling" rRefs="r1110"  Status="Unavailable">
        <MediaRef rRef="r1110"/>
      </ExposedMedia>
      <Media ID="r1110" Class="Consumable"  Status="Available" MediaType="Plate" PartID-
Keys="SheetName Side">
        <Media Dimension="2880 1944" SheetName="E08P5C">
          <Media Side="Front" Dimension="2880 1944"/>
        </Media>
      </Media>
    </ResourcePool>
  </JDF>
  <JDF ID="n1154" Type="Cutting" Status="Waiting">
    <ResourceLinkPool>
      <ComponentLink rRef="r1125" Usage="Input">
        <Part SheetName="E08P5C"/>
      </ComponentLink>
      <CuttingParamsLink rRef="r1129" Usage="Input"/>
      <ComponentLink rRef="r1131" Usage="Output"/>
    </ResourceLinkPool>
    <ResourcePool>
      <CuttingParams ID="r1129" Class="Parameter" rRefs="r1130 r1132 r1133 r1134"
Status="Available">
        <CutMarkRef rRef="r1130"/>
```

```
        <CutBlockRef rRef="r1132"/>
        <CutBlockRef rRef="r1133"/>
        <CutMarkRef rRef="r1134"/>
     </CuttingParams>
     <CutBlock ID="r1132" Class="Parameter"  Status="Available" BlockTrf="1 0 0 1 124.724409449
130.110236221" BlockName="Front_Sides" BlockSize="1190.55118111 1683.77952756" Block-
Type="CutBlock"/>
     <CutBlock ID="r1133" Class="Parameter"  Status="Available" BlockTrf="1 0 0 1 1564.72440945
130.110236221" BlockName="Back_Sides" BlockSize="1190.55118111 1683.77952756" Block-
Type="CutBlock"/>
     <Component ID="r1131" Class="Quantity" rRefs="r1107"  Status="Unavailable" PartID-
Keys="BlockName">
       <Component BlockName="Front_Sides" SourceSheet="E08P5C" ComponentType="Block">
         <SheetRef rRef="r1107"/>
       </Component>
       <Component BlockName="Back_Sides" SourceSheet="E08P5C" ComponentType="Block">
         <SheetRef rRef="r1107"/>
       </Component>
     </Component>
   </ResourcePool>
  </JDF>
  <JDF ID="n1155" Type="ImageSetting" Status="Waiting">
    <ResourceLinkPool>
      <ImageSetterParamsLink rRef="r1121" Usage="Input"/>
      <MediaLink rRef="r1110" Usage="Input"/>
      <RunListLink rRef="r1122" Usage="Input"/>
      <ExposedMediaLink rRef="r1123" Usage="Output"/>
    </ResourceLinkPool>
    <ResourcePool>
      <ImageSetterParams ID="r1121" Class="Parameter"  Status="Available"/>
      <RunList ID="r1122" Class="Parameter"  Status="Available"/>
    </ResourcePool>
  </JDF>
  <JDF ID="n1158" Type="Folding" Status="Waiting">
    <ResourceLinkPool>
      <FoldingParamsLink rRef="r1136" Usage="Input"/>
      <ComponentLink rRef="r1131" Usage="Input">
        <Part BlockName="Front_Sides"/>
      </ComponentLink>
      <ComponentLink rRef="r1138" Usage="Output"/>
    </ResourceLinkPool>
    <ResourcePool>
      <FoldingParams ID="r1136" Class="Parameter"  Status="Available" DescriptionType="FoldProc">
        <Fold To="Up" From="Front" Travel="297.638"/>
        <Fold To="Up" From="Left" Travel="420.945"/>
      </FoldingParams>
      <Component ID="r1138" Class="Quantity"  Status="Unavailable" ComponentType="Block" Descrip-
tiveName="Front_Sides"/>
    </ResourcePool>
  </JDF>
  <JDF ID="n1159" Type="Folding" Status="Waiting">
    <ResourceLinkPool>
      <FoldingParamsLink rRef="r1140" Usage="Input"/>
      <ComponentLink rRef="r1131" Usage="Input">
        <Part BlockName="Back_Sides"/>
      </ComponentLink>
      <ComponentLink rRef="r1142" Usage="Output"/>
    </ResourceLinkPool>
    <ResourcePool>
      <FoldingParams ID="r1140" Class="Parameter"  Status="Available" DescriptionType="FoldProc">
        <Fold To="Up" From="Front" Travel="297.638"/>
        <Fold To="Up" From="Left" Travel="420.945"/>
      </FoldingParams>
      <Component ID="r1142" Class="Quantity"  Status="Unavailable" ComponentType="Block" Descrip-
tiveName="Back_Sides"/>
    </ResourcePool>
  </JDF>
</JDF>
```

K.6 Runlist

The following example shows the various separation types, all mixed into one big RunList. Both in-line and Re-sourceRef versions of **LayoutElement** are used.

```
<ResourcePool>
  <Runlist ID="Link0003" Class="Parameter" NPage="10" rRefs="Link0004 Link0005"
Status="Available" PartIDKeys="Run Separation">
      <Comment>Preseparated Runs in multiple files
               All LayoutElements are inline resources
      </Comment>
    <RunList Run="1" NPage="1" FirstPage="0">
      <RunList Separation="Cyan">
        <LayoutElement Status="Unavailable">
          <FileSpec URL="File://Cyan.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Magenta">
        <LayoutElement Status="Unavailable">
          <FileSpec URL="File://Magenta.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Yellow">
        <LayoutElement Status="Unavailable">
          <FileSpec URL="File://Yellow.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="Black">
        <LayoutElement Status="Unavailable">
          <FileSpec URL="File://Black.pdf"/>
        </LayoutElement>
      </RunList>
      <RunList Separation="SpotGreen">
        <LayoutElement Status="Unavailable">
          <FileSpec URL="File://Green.pdf"/>
        </LayoutElement>
      </RunList>
    </RunList>
    <RunList Run="2" NPage="2" SkipPage="4">
      <Comment>
          Preseparated Runs in one file CMYKGCMYKG
          LayoutElements are inter-resource links
      </Comment>
      <RunList FirstPage="0" Separation="Cyan">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="1" Separation="Magenta">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="2" Separation="Yellow">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="3" Separation="Black">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="4" Separation="SpotGreen">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
    </RunList>
    <RunList Run="3" NPage="1" SkipPage="3">
      <Comment>
          No Magenta, the missing sep does not exist as a page
      </Comment>
      <RunList FirstPage="10" Separation="Cyan">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="11" Separation="Yellow">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
      <RunList FirstPage="12" Separation="Black">
        <LayoutElementRef rRef="Link0004"/>
      </RunList>
```

```
        <RunList FirstPage="13" Separation="Green">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
      </RunList>
      <RunList Run="4" NPage="2" SkipPage="4">
        <Comment>
            Continuation of Preseparated Runs in one file CMYKGCMYKG -
            the missing sep of the previous page does not exist as a page
        </Comment>
        <RunList FirstPage="14" Separation="Cyan">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
        <RunList FirstPage="15" Separation="Magenta">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
        <RunList FirstPage="16" Separation="Yellow">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
        <RunList FirstPage="17" Separation="Black">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
        <RunList FirstPage="18" Separation="SpotGreen">
          <LayoutElementRef rRef="Link0004"/>
        </RunList>
      </RunList>
      <RunList Run="5" NPage="2">
        <Comment>
            Preseparated Runs in one file CCMMYYKKGG
        </Comment>
        <RunList FirstPage="0" Separation="Cyan">
          <LayoutElementRef rRef="Link0005"/>
        </RunList>
        <RunList FirstPage="2" Separation="Magenta">
          <LayoutElementRef rRef="Link0005"/>
        </RunList>
        <RunList FirstPage="4" Separation="Yellow">
          <LayoutElementRef rRef="Link0005"/>
        </RunList>
        <RunList FirstPage="6" Separation="Black">
          <LayoutElementRef rRef="Link0005"/>
        </RunList>
        <RunList FirstPage="8" Separation="SpotGreen">
          <LayoutElementRef rRef="Link0005"/>
        </RunList>
      </RunList>
      <RunList Run="6" NPage="2">
        <Comment>
            Combined Runs in one file
        </Comment>
        <LayoutElement ElementType="document">
          <FileSpec URL="File://Combined.pdf"/>
        </LayoutElement>
      </RunList>
    </Runlist>
    <LayoutElement ID="Link0004" Class="Parameter" Status="Available">
      <FileSpec URL="File://PreSepCMYKG.pdf"/>
    </LayoutElement>
    <LayoutElement ID="Link0005" Class="Parameter" Status="Available">
      <FileSpec URL="File://PreSepCCMMYYKKGG.pdf"/>
    </LayoutElement>
</ResourcePool>
```

K.7 Messages

K.7.1 Simple KnownMessages

The following simple example shows a KnownMessages Query and the Response sent by a fairly dumb controller:

Query:
```
<?xml version='1.0' encoding='utf-8' ?>
```

```
<JMF SenderID="JMFClient" TimeStamp="2000-11-07T13:15:56+01:00"
xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1">
  <Query ID="Q0001" Type="KnownMessages">
    <KnownMsgQuParams ListQueries="true" ListSignals="false" ListCommands="true"/>
  </Query>
</JMF>
```

Response:
```
<?xml version='1.0' encoding='utf-8' ?>
<JMF SenderID="JMFClient #2" TimeStamp="2000-11-07T13:15:56+01:00"
xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1">
  <Response ID="R0001" Type="KnownMessages" refID="Q0001">
    <KnownMessages>
      <MessageService Type="KnownMessages" Query="true"/>
      <MessageService Type="Status" Query="true" Persistent="true"/>
      <MessageService Type="StopPersistentChannel" Command="true"/>
    </KnownMessages>
  </Response>
</JMF>
```

K.7.2 Simple persistent channel

The following query requests a persistent channel for Status messages. An update is requested whenever an attribute changes.

```
<?xml version='1.0' encoding='utf-8' ?>
<JMF SenderID="JMFClient" TimeStamp="2000-11-07T16:02:09+01:00"
xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1">
  <Query ID="Q0011" Type="Status">
    <Subscription URL="http:://123.123.123.123/message/recipient">
      <ObservationTarget Attributes="*"/>
    <StatusQuParams JobDetails="brief"/>
  </Query>
</JMF>
```

The following four examples are a set of typical, simple responses that are emitted whenever *DeviceStatus* changes.

This is the Response that is sent immediately within the same HTTP connection as the Query.
```
<?xml version='1.0' encoding='utf-8' ?>
<JMF SenderID="JMFClient #2" TimeStamp="2000-11-07T16:02:19+01:00"
xmlns="http://www.CIP4.org/JDFSchema_1_1" Version="1.1">
  <Response ID="R0013" Type="Status" refID="Q0011">
    <DeviceInfo DeviceStatus="Idle"/>
  </Response>
</JMF>
```

Appendix L   JDF/CIP4 Hole Pattern Catalog

The following table defines the specifics of the predefined holes in **HoleMakingParams** and **HoleMakingIntent.**
Notes:
1. All patterns are centered on the sheet along the process edge.
2. Process Edge is always defined relative to a portrait orientation of the medium, regardless of the orientation of the printed image or processing path.
3. Thumbcuts are available in various standard shapes (labeled "No. N" where N is minimally ranging from 2..7). "No. 3" seems to be the most widely used.
4. Single thumbcuts appear always in the center of the process edge.
5. Oval shape holes actually look sometimes more like rectangular holes with rounded corners.

Sources:
1. Printer Finishing MIB, IETF Draft, 2001-10-01 (http://www.ietf.org/internet-drafts/draft-ietf-printmib-finishing-12.txt)

Naming Scheme:

General	<m|i>: m = metric (millimeter is used), i = imperial (inch, where 1 inch = 25.4 mm)
Ring Binding	R<#holes><m|i>-<variant> Example: R2m-DIN = RingBind, 2 hole, metric, DIN
Plastic Comb	P<pitch><m|i>-<shape>-<#thumbcuts>t Example: P16:9m-round-0t = Plastic Comb, 9/16" pitch (16:9), round, no thumbcut
Wire Comb	W<pitch><m|i>-<shape>-<#thumbcuts>t Example: W2:1i-square-1t = Wire Comb, 1/2" pitch (2:1), square, one thumbcut
Coil/Spiral	C<pitch><m|i>-<shape>-<#thumbcuts>t Example: C9.5m-round-0t = Coil, 9.5 mm, round, no thumbcut

JDF Hole Pattern Catalog ID	Description	#Holes	Hole Shape	Hole Extent	Pattern Geometry	Pattern Axis Offset from Process Edge	JDF Default Pattern Axis Offset from Process Edge in pt (!)	Default Process Edge	Usage Notes	Source Standard
RING BINDING (R...)										
2 Holes (R2...)										
R2-generic	Generic request of a 2-hole pattern	2	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	34.02 (≅ 12 mm)	Left	See note (7).	N/A
R2m-DIN	DIN 2-hole MIB: 6 = twoHoleDIN and 10 = twoHoleMetric	2	●	5.5 ± 0.1 mm	80 ± 0.1 mm	7 or 11 ± 0.3 mm 7 mm for blocks of <= 15 mm thick	31.18 (≅ 11 mm)	Left	A4 and A5	DIN 5005:1991 DIN 821:1973
R2m-ISO	ISO 2-hole MIB: 6 = twoHoleDIN and 10 = twoHoleMetric	2	●	6 ± 0.5 mm	80 ± 0.5 mm	12 ± 1 mm Australian Standard AS P5-1969: 10 ± 1 mm	34.02 (≅ 12 mm)	Left	Also used in Japan	ISO 838:1974 (E)
R2m-MIB	Printer Finishing MIB twoHoleDIN and two-HoleMetric	2	●	5-8 mm	80 ± 0.5 mm	4.5 – 13 mm	31.18 (≅ 11 mm)	Left		Printer Finishing MIB
R2i-US-a	US 2-hole, Variant A MIB: 4 = twoHoldUS-Top and 12 = twoHoleUSSide	2	●	0.2 - 0.32"	2.75"	0.18 - 0.51"	29.25 (≅ 13/32")	Left for letter Top for ledger		Printer Finishing MIB
R2i-US-b	US 2-hole, Variant B	2	●	0.2-0.5" default: 5/16" typical: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	6"	0.25" + ½ diameter range: 6/16" - 1/2"	29.25 (≅ 13/32")	Left		

JDF Hole Pattern Catalog ID	Description	#Holes	Hole Shape	Hole Extent	Pattern Geometry	Pattern Axis Offset from Process Edge	JDF Default Pattern Axis Offset from Process Edge in pt (!)	Default Process Edge	Usage Notes	Source Standard
3 Holes (R3...)										
R3-generic	Generic request of a 3-hole pattern.	3	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	29.25 (≅ 13/32")	Left	See note (7).	N/A
R3i-US	US 3-hole MIB: 5 = threeHoleUS	3	●	std: 5/16" rng: 0.2-0.5" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	4.25"	0.25" + ½ diameter range: 6/16" - 1/2"	29.25 (≅ 13/32")	Left		Printer Finishing MIB
4 Holes (R4...)										
R4-generic	Generic request of a 4-hole pattern.	4	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	31.18 (≅ 11 mm)	Left	See note (7).	N/A
R4m-DIN-A4	DIN 4-hole for A4	4	●	5.5 ± 0.1 mm	80 ± 0.1 mm	7 or 11 ± 0.3 mm 7 mm for blocks of 15 mm or less	31.18 (≅ 11 mm)	Left	A4	DIN 5005:1991 DIN 821:1973
R4m-DIN-A5	DIN 4-Hole for A5	4	●	5.5 ± 0.1 mm	45-65-45 mm	7 or 11 ± 0.3 mm 7 mm for blocks of 15 mm or less	31.18 (≅ 11 mm)	Left	A5	DIN 5005:1991
R4m-swedish	Swedish 4-hole MIB: 11 = swed-ish4Hole	4	●	5 - 8 mm	21-70-21 mm	4.5 - 13 mm	31.18 (≅ 11 mm)	Left for A4 Top for A3	A4, A3	Printer Finishing MIB
R4i-US	US 4-Hole	4	●	0.2 - 0.5" std: 5/16" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	1.375-4.25-1.375"	0.25" + ½ diameter range: 6/16" - 1/2"	29.25 (≅ 0.25" + ½ x 5/16" = 13/32")	Left		

JDF Hole Pattern Catalog ID	Description	#Holes	Hole Shape	Hole Extent	Pattern Geometry	Pattern Axis Offset from Process Edge	JDF Default Pattern Axis Offset from Process Edge in pt (!)	Default Process Edge	Usage Notes	Source Standard
5 Holes (R5...)										
R5-generic	Generic request of a 5-hole pattern.	5	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	29.25 (≅ 13/32")	Left	See note (7).	N/A
R5i-US-a	US 5-hole, Variant A MIB: 13 = fiveHoleUS	5	●	0.2 - 0.32"	2-2.25-2.25-2"	0.18 - 0.51"	29.25 (≅ 13/32")	Left for letter Top for ledger		Printer Finishing MIB
R5i-US-b	US 5-hole, Variant B	5	●	0.2 - 0.5" std: 5/16" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	0.75-3.5-3.5-0.75"	0.25" + ½ diameter 0.375 - 0.5"	29.25 (≅ 0.25" + ½ x 5/16" = 13/32")	Left		
R5i-US-c	Combination of R2i-US-a and R3i-US	5	●	0.2 - 0.5" std: 5/16" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	1.25-3-3-1.25"	0.25" + ½ diameter 0.375 - 0.5"	29.25 (≅ 0.25" + ½ x 5/16" = 13/32")	Left		
6 Holes (R6...)										
R6-generic	Generic request of a 6-hole pattern.	6	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	31.18 (≅ 11 mm)	Left for A4/A5 Top for A3	See note (7).	N/A
R6m-4h2s	Norwegian 4-hole (round) mixed with 2 slots (rectangular) MIB: 16 = norweg6Hole	6	H: ● S: ▬	Holes: 5 - 8 mm Slots: 10 x 5.5 mm	4 holes/2 slots Pattern: H-H-S-S-H-H 64-18.5-75-18.5-64 mm	4.5 - 13 mm	31.18 (≅ 11 mm)	Left for A4 Top for A3		Printer Finishing MIB
R6m-DIN-A5	DIN 6-Hole for A5	6	●	5.5 ± 0.1 mm	37.5-7.5-65-7.5-37.5 mm	7 or 11 ± 0.3 mm 7 mm for blocks of <= 15 mm thick	31.18 (≅ 11 mm)	Left	Only used with A5	DIN 5005:1991

JDF Hole Pattern Catalog ID	Description	#Holes	Hole Shape	Hole Extent	Pattern Geometry	Pattern Axis Offset from Process Edge	JDF Default Pattern Axis Offset from Process Edge in pt (!)	Default Process Edge	Usage Notes	Source Standard
7 Holes (R7...)										
R7-generic	Generic request of a 7-hole pattern.	7	●	5 - 13 mm 0.2-0.51"	N/A	4.5 – 13 mm 0.18 - 0.51"	29.25 (≅ 13/32")	Left for letter Top for ledger	See note (7).	N/A
R7i-US-a	US 7-hole, Variant A MIB: 14 = seven-HoleUS	7	●	0.2 - 0.32"	1-1-2.25-2.25-1-1"	0.18 - 0.51"	29.25 (≅ 13/32")	Left for letter Top for ledger		Printer Finishing MIB
R7i-US-b	US 7-hole, Bell/AT&T Systems. Combination of R3i-US, R4i-US, R5i-US-b	7	●	0.2 - 0.5" std: 5/16" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	0.75-1.375-2.125-2.125-1.375-0.75"	0.25" + ½ diameter 0.375 - 0.5"	29.25 (≅ 0.25" + ½ x 5/16" = 13/32")	Left for letter Top for ledger		
R7i-US-c	US 7-hole, Variant C	7	●	0.2 - 0.5" std: 5/16" typ: 1/4", 9/32", 11/32", 3/8", 13/32", 1/2"	1.25-0.875-2.125-2.125-0.875-1.25"	0.25" + ½ diameter 0.375 - 0.5"	29.25 (≅ 13/32")	Left for letter Top for ledger		
11 Holes (R11...)										
R11m-7h4s	7-hole (round) mixed with 4 slots (rectangular) MIB: 15 = mixed7H4S	11	H: ● S: ▬	Holes: 5 - 8 mm Slots: 12 x 6 mm	7 holes/2slots Pattern: H-S-H-H-S-H-S-H-H-S-H 15-25-23-20-37-37-20-23-25-15 mm	4.5 - 13 mm	31.18 (≅ 11 mm)	Left for A4 Top for A3		Printer Finishing MIB

JDF Hole Pattern Catalog ID	Description	#Holes	Hole Shape	Hole Extent	Pattern Geometry	Pattern Axis Offset from Process Edge	JDF Default Pattern Axis Offset from Process Edge in pt (!)	Default Process Edge	Usage Notes	Source Standard
PLASTIC COMB BINDING (P...)										
P16_9i-rect-0t	US spacing, no thumbcut MIB: 9 = nineteen-HoleUS	A4: 21 Letter: 19	■	5/16" x 1/8" (8 x 3.2 mm)	9/16"	3/16"	13.54 ($\cong$ 0.188")	Left		Printer Finishing MIB
P12m-rect-0t	European spacing, no thumbcut		■	7 x 3 mm	12 mm	4.5 mm	12.76 ($\cong$ 4.5 mm)	Left		
WIRE COMB BINDING (W...)										
W2_1i-round-0t	2:1, round, no thumbcut MIB: 8 = twentyTwo-HoleUS	A4: 23 Letter: 21	●	0.2 - 0.32" std: 1/4" Europe typ: 6 or 6.4 mm	1/2"	3 mm + ½ diameter 0.318 - 0.438" Europe: 6 - 6.2 mm	17.50 ($\cong$ 0.243")	Left		Printer Finishing MIB
W2_1i-square-0t	2:1, square, no thumbcut	A4: 23 Letter: 21	■	0.2 - 0.32" std: 1/4" Europe typ: 6 or 6.4 mm	1/2"	3 mm + ½ diameter 0.318 - 0.438" Europe: 6 - 6.2 mm	17.50 ($\cong$ 0.243")	Left		
W3_1i-square-0t	3:1, square, no thumbcuts	A4: 34 A5: 24 Letter: 32	■	5/32 x 5/32" (4x4 mm)	1/3"	0.2"	14.40 ($\cong$ 0.2")	Left		
COIL/SPIRAL BINDING (C...)										
C9.5m-round-0t	9.5 mm, round, no thumbcut MIB: 17 - metric26Hole and 18 - metric30Hole	A4/A3: 30 JIS B5/B4: 26	●	5 - 8 mm	9.5 mm	4.5 - 13 mm	31.18 ($\cong$ 11 mm)	Left for A4/JIS B5 Top for A3/JIS B4		Printer Finishing MIB
SPECIAL (S...)										
										Reserved for future extensions

Appendix M New, Deprecated, Modified, Illegal, and Removed Items

M.1 New Items

Location	Section Title	Comments
Preface	User Overview	Provides information and guides for understanding the objectives, value, and purpose of JDF.
Section 1.1	Background on JDF	History and benefits of JDF.
Section 1.4.1	Conformance Terminology	Clarification of language used in this specification.
Section 1.4.2	Conformance Requirements for JDF Entities	Definition of general conformance requirement for JDF entities.
Section 2.5	Coordinate Systems in JDF	How coordinate systems are defined and used in JDF>
Section 4.9	Dynamic State Machines Using ResourceUpdate	
Section 6.5.45	Postpress Processes Structure	Revision of Packaging Processes. Merges all the process for making a book block.
Section 7.1.1.2	Structure of the Duration Span Subelement	Describes a selection of instances in time.
Section 7.1.1.8	Structure of the ShapeSpan Subelement	Describes ranges of numerical value pairs.
Section 7.1.6	Embossing Intent	Specifies the embossing and/or foil stamping intent.
Section 7.1.13	NumberingIntent	Describes the parameters of stamping or applying variable marks to produce unique components.
Section 7.2.29	ContactCopyParams	Describes the parameters of **ContactCopying**.
Section 7.2.53	FitPolicy	Specifies how to fit content into a receiving container.
Section 7.2.54	Fold	Describes an individual folding operation of the **Component**.
Section 7.2.60	GlueApplication	Specifies glue application in hard and soft cover book production
Section 7.2.63	HeadBandApplicationParams	Specifies how to apply headbands in hard cover book production.
Section 7.2.64	Hole	Describes an individual hole.
Section 7.2.65	HoleLine	Specifies parameters for holes series for transporting paper through continuous-feed printers and finishing devices.
Section 7.2.126	SpinePreparationParams	Describes the preparation of the spine of book blocks for hard and soft cover book production.
Section 7.2.143	StripBindingParams	Describes details of the **StripBinding** process.
Section 7.3	Device Capability Definitions	Specifies capabilities of devices.
Appendix A.2.2	DurationRange	Describes XML attributes of *DurationRange*.
Appendix A.2.16	ShapeRange	Describes XML attributes of *ShapeRange*.
Appendix A.2.17	ShapeRangeList	Describes XML attributes of *ShapeRangeList*.

M.2 Deprecated Items

Location	Table Info	Comments
Section 3.4 **Customer Information** Table 3.6	Company ? refelement	Company affiliation of Contacts is specified in Contact.
Section 3.5 **Node Information** Table 3.7	*MergeTarget* ? boolean	Avoiding concurrent access to the ancestor node is ill defined and cannot be implemented in an open system without proprietary locking mechanisms.
Section 3.7.1.6	**Selector Resources**	Resources of class *Selector* have been removed. Note that they are not only deprecated but actually removed from the format including the schema and must not be supported by a JDF 1.1 conforming agent

Location	Table Info	Comments
Section 3.8 **Resource Links** Table 3.17	*CombinedProcessType*	Replaced by *CombinedProcessIndex*.
Section 6.2.7 **Packing**		Replaced by the individual processes defined in Section *6.5.45.5 Packaging Processes*
Section .3.7 **FilmToPlate Copying**		Replaced by the more generic *ContactCopying*.
Section 6.3.21 **Rendering**	**Input Resources** Media	
Section 6.4.3 **IDPrinting**		Controls for *IDPrinting* are provided in the **IDPrintingParams** resource. These controls are intended to be somewhat limited in their scope. If greater control over various aspects of the printing process is required, *IDPrinting* should not be used.
Section 6.5.1 **Adhesive Binding**		The *AdhesiveBinding* has been split into: • *CoverApplication,* • *Gluing* • *SpinePreparation,* • *SpineTaping.* The parameters of the *GlueApplication* ABOperations have been moved into **CoverApplicationParams** and **SpineTapingParams** as GlueApplication refelements. The generic *GlueApplication* ABOperation is now described by the *Gluing* process.
Section 6.5.12 **Dividing**		*Dividing* has been replaced by *Cutting.*
Section 6.5.24 **Longitudinal Ribbon Operations**		In-line finishing is described using the "standard" finishing processes, e.g., *Creasing*, *Cutting,* or *Folding* in a combined node with *ConventionalPrinting*.
Section 6.5.30 **Saddle Stitching**		Replaced by *Stitching.*
Section 6.5.33 **SideSewing**		Replaced by *ThreadSewing.*
Section 7.1.2 **ArtDelivery Intent**	**Resource Structure** Company ? refelement	
	Structure of ArtDelivery Elements Company ? refelement	
	Structure of ArtDelivery Elements Component ? refelement	
Section 7.1.3 **BindingIntent**	**Resource Structure** *BindingType* EnumerationSpan	Replaced with *SoftCover* or *HardCover*.
	Resource Structure AdhesiveBinding ? element	
	Resource Structure BookCase ? element	
	Structure of the AdhesiveBinding Subelement	
	Structure of the BookCase Subele-	

Location	Table Info	Comments
	ment	
	Structure of the RingBinding Subelement *RingSystem* NameSpan	*2HoleEuro, 3HoleUS, 4HoleEuro* have been replaced by *HoleType*.
Section 7.1.5 **DeliveryIntent**	**Resource Structure** Pickup ? boolean	
	Resource Structure Company ? refelement	
	Structure of Delivery-Intent Elements: DropIntent Pickup? boolean	
	Structure of Delivery-Intent Elements: DropIntent Company? refelement	
Section 7.1.7 **FoldingIntent**	**Resource Structure** *Folds?* XYPair	
Section 7.1.8 **HoleMakingIntent**	**Resource Structure** *HoleType* StringSpan	*2HoleEuro* – Replace by either R2m-DIN or R2m-ISO. *3HoleUS* – Replace by R3I-US *4HoleEuro* – Replace by R4m-DIN-A4 or R4m-DIN-A5.
Section 7.1.9 **InsertingIntent**	**Structure of Insert Subelement** *SheetOffset?* XYPair	
Section .1.10 **LaminatingIntent**	**Resource Structure** *Laminated* OptionSpan	
Section 7.1.11 **LayoutIntent**	**Resource Structure** *FinishedPage Orientation* enumeration	In JDF 1.1, the page orientation is implied by the value of *Dimensions* and *FinishedDimensions.* If height (X) > width (Y), the product is portrait.
Section .1.12 **MediaIntent**	**Resource Structure** *HoleType ?* StringSpan	*2HoleEuro* – Replace by either R2m-DIN or R2m-ISO. *3HoleUS* – Replace by R3I-US *4HoleEuro* – Replace by R4m-DIN-A4 or R4m-DIN-A5.
	Resource Structure *HoleType ?* IntegerSpan	
Section 7.1.18 **SizeIntent**		All contents have been moved to **LayoutIntent**.
Section 7.2.3 **AdhesiveBinding Params**		
Section 7.2.15 **CIELABMeasuring Field**	**Resource Structure** *Light* NMTOKEN	
	Resource Structure *Observer* integer	
	Resource Structure *ScreenRuling ?*	

Location	Table Info	Comments
	NumberList	
	Resource Structure *ScreenShape ?* string	
	Resource Structure *Setup ?* string	
Section 7.2.21 **ColorCorrection Params**	**Resource Structure** FileSpec ? refelement	
Section 7.2.24 **ColorSpace Conver-sionParams**	**Resource Structure** FileSpec ? refelement	
Section .2.26 **Company**	**Resource Structure** Contact * refelement	
Section 7.2.27 **Component**	**Resource Structure** *Transformation ?* matrix	Use ResourceLink::*Transformation*.
Section 7.2.41 **DeliveryParams**	**Resource Structure Company** *?* refelement	
	Structure of the Drop Subelement Company *?* refelement	
Section 7.2.44 **Device**	**Resource Structure** *DeviceFamily ?* string	*DeviceFamily* is replaced by the appropriate *ModelXXX* attributes in this list.
Section 7.2.46 **Disjointing**	**Resource Structure** *Overfold ?* double	Moved to **Component**.
Section 7.2.47 **DividingParams**		
Section 7.2.55 **FoldingParams**	**Resource Structure** *FoldSheetIn ?* XYPair	
Section .2.66 **HoleMakingParams**	**Resource Structure** *HoleType* enumerations	*2HoleEuro* – Replace by either R2m-DIN or R2m-ISO. *3HoleUS* – Replace by R3I-US *4HoleEuro* – Replace by R4m-DIN-A4 or R4m-DIN-A5
Section 7.2.68 **IDPrintingParams**	**Structure of the Cover Subelement**	
	Properties of the IDPFinishing Subele-ment	
	Structure of IDPFold-ing Subelement	
	Structure of IDPHole-Making Subelement	
	Structure of the ID-PLayout Subelement	
	Structure of IDPStitching Subele-ment	
	Structure of IDPTrimming Subelement	
	Structure of the Im-	

Location	Table Info	Comments
	ageShift Subelement	
	Structure of the Job-Sheet Subelement	
Section 7.2.69 **ImageCompression Params**	**Structure of Image-Compression Subelement** *EncodeColorImages ?* boolean	Replaced with *EncodeImages*
Section 7.2.70 **ImageReplacement Params**	**Resource Structure** *MaxResolution ?* double	Replaced with a link to ImageCompressionParams in the process.
	Resource Structure *ResolutionReduction Strategy ?* enumeration	Replaced with a link to ImageCompressionParams in the process.
	Resource Structure SearchPath + telem	
Section 7.2.75 **InsertingParams**	**Resource Structure** *SheetOffset* XYPair	SheetOffset is implied by the Transformation matrix in ResourceLink:*Transformation* of the child's ComponentLink.
Section 7.2.78 **InterpretingParams**	**Structure of the InterpretingParams Resource** *FitToPage ?* boolean	Replaced by **FitPolicy ?** refelement.
Section 7.2.86 **LongitudinalRibbon OperationParams**	**LROperation**	
	LongFold	
	LongGlue	
	LongPerforate	
	LongSlit	
Section 7.2.88 **Media**	**Resource Structure** *HoleCount ?* integer	
Section 7.2.89 **MediaSource**		
Section 7.2.93 **PackingParams**		Replaced by the individual resources used by the processes defined in Section *6.5.45.5 Packaging Processes*
Section 7.2.96 **PDFToPSConversionParams**	**Resource Structure** *IgnoreDeviceExtGState* ? boolean	
Section 7.2.102 **PlateCopyParams**		
Section 7.2.113 **ResourceDefinition Params**	**Resource Structure** *DefaultID ?* NMTOKEN	
Section 7.2.114 **RingBindingParams**	**Resource Structure** *RingSystem ?* enumeration	
Section 7.2.116 **SaddleStitching Params**		
Section 7.2.125		

Location	Table Info	Comments
SideSewingParams		
Section 7.2.132 **Surface**	**Structure of the Abstract Placed Object Subelement** *Type* enumeration	
	Structure of Dynamic Field Subelement *InputField ?* string	

M.3 Modified Items

Location	Table Info	Comments
Section 1.4		Glossary of Terminology has been expanded to accommodate document additions.
Table 3.9	Part element	Specifies the selected part that the PartStatus is valid for. If a Part refers to less PartIDKeys than are available in the resource, the unspecified PartIDKeys are implied to be accepted.

M.4 Illegal Items

Location	Table Info	Comments
Section 3.7.1.4 **Physical Resources** Table 3.13	Weight ? double	This parameter collides with Media::Weight.

M.5 Removed Items

Location	Table Info	Comments
Section 3.7.1.6 **Selector Resources**		Resources are not only deprecated but actually removed from the format including the schema and must not be supported by a JDF 1.1 conforming agent
Section 4.1.2.1 Request for Quote		

M.6 New/Modified Attributes and Elements

M.6.1 Structure of JDF Nodes and Jobs

Location	Name	Data Type	Comment
Table 3-1 Generic Contents of elements	*BestEffortExceptions ?*	NMTOKENS	New
	MustHonorExceptions ?	NMTOKENS	New
	OperatorIntervention-Exceptions ?	NMTOKENS	New
Table 3-2 Contents of the Comment element	*Attribute ?*	NMTOKEN	New
Table 3-3 Contents of a JDF node	*ProjectID ?*	string	New
	SpawnID ?	NMTOKEN	New
	SettingsPolicy ?	enumeration	New
	Template ?	boolean	New
	Version ?	string	New
	xmlns ?	URI	New
Table 3-4 Contents of the AncestorPool element	Part *	Element	New
Table 3-5 Attributes of the An-	*SpawnID ?*	NMTOKEN	New

Location	Name	Data Type	Comment
cestor element	CustomerInfo ?	element	New
	NodeInfo ?	element	New
Table 3-6 Contents of the Cus-tomerInfo element	Company ?	refelement	Deprecated. Company affiliation of Contacts is specified in Con-tact.
	Contact *	refelement	New
Table 3-7 Contents of the NodeInfo element	*CleanupDuration ?*	duration	Data Type modified.
	End ?	dateTime	Data Type modified.
	FirstEnd ?	dateTime	Data Type modified.
	FirstStart ?	dateTime	Data Type modified.
	IPPVersion ?	dateTime	New
	JobPriority ?	integer	New
	LastEnd ?	dateTime	Data Type modified.
	LastStart ?	dateTime	Data Type modified.
	NaturalLang ?	language	New
	MergeTarget ?	boolean	Deprecated. Avoiding concurrent access to the ancestor node is ill defined and cannot be imple-mented in an open system without proprietary locking mechanisms.
	SetupDuration ?	duration	Data Type modified.
	Start ?	dateTime	Data Type modified.
	TotalDuration ?	duration	Data Type modified.
Table 3-9 Contents of the Part-Status element	Part	element	Modified. The cardinality of Part in PartStatus has been changed from * to none.
Table 3-11 Contents of the ab-stract Resource element	*SettingsPolicy ?*	enumeration	New
	SpawnIDs ?	NMTOKENS	New
	Status	enumeration	modified value list. Added *Com-plete*
	UpdateID ?	NMTOKEN	New
Table 3-12 Additional contents of the abstract parameter Re-source elements	*NoOp ?*	boolean	New
Table 3-13 Additional contents of the abstract physical Re-source elements	*ResourceWeight ?*	double	New
	Weight ?	double	Illegal. Collides with Me-dia::Weight.
	IdentificationField *	refelement	New
Table 3-14 Contents of the Lo-cation element	*LocationName ?*	string	New
Table 3-15 Contents of the ab-stract ResourceUpdate	*UpdateID*	NMTOKEN	New
Table 3-17 Contents of the ab-stract ResourceLink element	*CombinedProcessIndex ?*	IntegerList	New
	CombinedProcessType ?	NMTOKEN	Deprecated. Replaced by *Com-binedProcessIndex.*
	PipeProtocol ?	NMTOKEN	New
	AmountPool ?	element	New
Table 3-18 Contents of the	PartAmount *	element	New

Location	Name	Data Type	Comment
AmountPool element			
Table 3-19 General contents of the PartAmount element	*DraftOK ?*	boolean	New
	PipeURL ?	URL	New
	Part	element	New
Table 3-20 Contents of the abstract ImplementationLink or PartAmount element	*Duration ?*	duration	New and modified.
	Recommendation ?	boolean	New (PartAmount)
	Start ?	dateTime	New and modified.
	StartOffset ?	duration	New and modified.
Table 3-21 Additional contents of the abstract physical ResourceLink and PartAmount or AmountPool element	*Amount ?*	number	New (PartAmount and Amount-Pool)
	Orientation ?	enumeration	New
	PipePause ?	number	New (PartAmount and Amount-Pool)
	PipeResume ?	number	New (PartAmount and Amount-Pool)
	RemotePipeEndPause ?	number	New (PartAmount and Amount-Pool)
	RemotePipeEndRe-sume ?	number	New (PartAmount and Amount-Pool)
	Transformation ?	matrix	New
Table 3-23 Contents of the abstract ResourceRef element	Part ?	element	New
Table 3-25 Contents of the Part element	*BlockName ?*	NMTOKEN	New
	LayerIDs ?	IntegerRange List	New
	PageNumber ?	Integer-RangeList	Data type modified.
	PreviewType ?	enumeration	New
	Run ?	string	Data type modified.
	RunTags ?	NMTOKENS	New
	RunPage ?	integer	New
	SetIndex ?	Integer-RangeList	New
Table 3-26 Locations within Printers	*Top, Middle, Bottom, Side, Left, Right, Center, Rear, FaceUp, FaceDown, FitMedia, LargeCapacity, Mailbox-N, Stacker-N, Tray-N, SystemSpecified*		New
Contents of the Selector resource	Part +	element	Deleted
Table 3-28 Contents of the abstract Audit type	*SpawnID ?*	NMTOKEN	New
	TimeStamp	dateTime	Data type modified.
Table 3-29 Contents of the ProcessRun element	*Duration ?*	duration	Data type modified.
	End	dateTime	Data type modified.
	Start	dateTime	Data type modified.
	Part *	element	New
Table 3-30 Contents of the Notification element	Part *	element	New

Location	Name	Data Type	Comment
Table 3-32 Contents of the PhaseTime element	*End*	dateTime	Data type modified.
	Start	dateTime	Data type modified.
	ResourceLink *	element	New
Table 3-33 Contents of the ModulePhase element	*End*	dateTime	Data type modified.
	Start	dateTime	Data type modified.
Table 3-34 Contents of the ResourceAudit element	*Reason ?*	enumeration	New
Table 3-37 Contents of the Spawned element	*NewSpawnID*	NMTOKEN	New
	Status ?	enumeration	New
	URL ?	URL	New
Table 3-38 Contents of the Merged element	*MergeID*	NMTOKEN	New
	URL ?	URL	New

M.6.2 JDF Messaging with the Job Messaging Format

Location	Name	Data Type	Comment
Table 5-1 Contents of the JMF root	*TimeStamp*	dateTime	Data type modified.
	xmlns ?	URI	New
Table 5-2 Contents of the abstract Message element	*Time ?*	dateTime	Data type modified.
Table 5-10 Contents of the Command message element	*AcknowledgeType ?*	enumerations	New
Table 5-11 Contents of the Acknowledge message element	*AcknowledgeType ?*	enumerations	New
Table 5-22 Contents of the DeviceFilter element	*DeviceDetails ?*	enumeration	New
Table 5-25 Contents of the JDFService element	*CombinedMethod ?*	enumeration	New
	TypeOrder ?	enumeration	New
Table 5-27 Contents of the KnownMsgQuParams element	*Exact ?*	boolean	New
Table 5-28 Contents of the MessageService element	*Acknowledge ?*	boolean	New
	DevCaps *	element	New
Table 5-30 Contents of the MsgFilter element	*After ?*	dateTime	Data type modified.
	Before ?	dateTime	Data type modified.
Table 5-40 Contents of the ResourceCmdParams element	*Activation ?*	enumeration	New
	UpdateIDs ?	NMTOKENS	New
Table 5-44 Contents of the DeviceInfo element	*HourCounter ?*	duration	Data type modified.
	PowerOnTime ?	dateTime	Data type modified.
Table 5-45 Contents of the JobPhase element	*Activation ?*	enumeration	New
	RestTime ?	duration	New
	StartTime ?	dateTime	New
	TotalAmount ?	number	New
	Waste ?	number	New
	Part *	element	Modified to *.
Table 5-81 Contents of the QueueEntry element	*JobID ?*	string	Modified to optional.
	StartTime ?	dateTime	New
	SubmissionTime ?	dateTime	Data type modified.

M.6.3 Processes

Location	Name	Comment
6.2.2 Buffer	BufferParams	New section.
	Resource	
Output Resources	Resource	
6.2.5 ManualLabor: Input Resources	Resource *	New section.
	ManualLaborParams	
Output Resources	Resource	
6.2.6 Ordering: Output Resources	Resource +	Modified name; allow multiple output resources.
6.2.7 Packing		Deprecated. Replaced by the individual processes defined in Section 6.5.44.5 *Packaging Processes*.
6.2.8 ResourceDefinition: Input Resources	Resource *	Modified to optional multiple.
Output Resources	Resource +	Modified to required.
6.3.1 ColorCorrection Input Resources	ColorCorrectionParams	New
6.3.3 ContactCopying:		New section.
6.3.4 ContoneCalibration: Input Resources	ScreeningParams ?	Modified to optional.
	TransferFunctionControl ?	Modified to optional.
6.3.7 FilmToPlateCopying		Deprecated. Replaced by *ContactCopying*
6.3.8 FormatConversion		New section.
6.3.9 ImageReplacement: Input Resources	ImageCompressionParams ?	New
6.3.10 ImageSetting: Input Resources	DevelopingParams ?	New
	ImageSetterParams ?	Modified to optional.
	TransferCurvePool ?	New
6.3.12 InkZoneCalculation: Input Resources	Layout ?	New
	Sheet ?	Deleted
6.3.13 Interpreting: Input Resources	ColorantControl ?	Modified to optional.
6.3.15 LayoutPreparation		New section.
6.3.18 PreviewGeneration: Input Resources	ColorantControl ?	New
	Preview ?	New
	TransferCurvePool ?	New
6.3.21 Rendering Input Resources	Media	Deprecated.
6.3.24 Screening: Input Resources	ScreeningParams ?	Modified to optional.
6.3.28 Trapping: Input Resources	FontPolicy ?	New
6.4.1 ConventionalPrinting: Input Resources	Ink ?	Modified to optional
	Layout ?	New
	Sheet *	Deprecated
	TransferCurvePool ?	New

Location	Name	Comment
	Layout ?	New
	Sheet *	Deprecated
	TransferCurvePool ?	New
6.4.3 IDPrinting		Deprecated section.
Input Resources	Ink ?	New
6.5.1 AdhesiveBinding		Deprecated section. Split into: CoverApplication, Gluing, Spine-Preparation, SpineTaping.
6.5.2 BlockPreparation		New section. Modifies Block Pro-duction.
6.5.3 BoxPacking		New section.
6.5.4 CaseMaking		New section.
6.5.5 CasingIn		New section.
6.5.9 CoverApplication		New section
6.5.10 Creasing		New section.
6.5.11 Cutting : Input Re-sources	CuttingParams ?	New. Replaces CutBlock * and CutMark *.
6.5.12 Dividing		Deprecated
6.5.13 Embossing		New section.
6.5.15 Folding: Output Re-sources	Component	Modified from required.
6.5.17 Gluing		New section
6.5.18 HeadBandApplication		New section.
6.5.21 Jacketing		New section.
6.5.22 Labeling		New section
6.5.23 Laminating		New section
6.5.24 LongitudinalRibbon-Operations		Deprecated. In-line finishing is de-scribed using the "standard" finish-ing processes.
6.5.26 Palletizing		New section.
6.5.27 Perforating		New section.
6.5.30 SaddleStitching		Deprecated. Replaced by *Stitch-ing*.
6.5.31 ShapeCutting		New section.
6.5.32 Shrinking		New section.
6.5.33 SideSewing		Deprecated. Replaced by *Thread-Sewing*.
6.5.34 SpinePreparation		New section
6.5.35 SpineTaping		New section
6.5.36 Stacking		New section.
6.5.38 Strapping		New section.
6.5.39 StripBinding		New section. Renamed from Velo-Binding.
6.5.40 ThreadSealing		New section.
6.5.44 Wrapping		New section.

M.6.4 Resources

Location	Name	Data Type	Comment
7.1.1.2 Structure of the Dura-tionSpan Subelement		element	New section.
7.1.1.8 Structure of the ShapeSpan Subelement		element	New section
7.1.1.10 Structure of the Time-Span Subelement	*Actual ?*	dateTime	Modified data type.
	Preferred ?	dateTime	Modified data type.
7.1.2 ArtDeliveryIntent: Re-source Structure	*ArtDeliveryDate ?*	TimeSpan	New
	ArtDeliveryDuration ?	DurationSpan	New
	ArtHandling ?	Enumera-tionSpan	New
	DeliveryCharge ?	Enumera-tionSpan	New
	PreflightStatus ?	enumeration	New
	ReturnList ?	NMTOKENS	New
	ReturnMethod ?	NameSpan	New
	Transfer ?	Enumera-tionSpan	New
	ArtDelivery +	element	Modified to required (+).
	Company ?	refelement	Deprecated
	Contact *	refelement	New
Structure of ArtDelivery Ele-ments	*ArtDeliveryDate ?*	TimeSpan	New
	ArtDeliveryDuration ?	DurationSpan	New
	ArtDeliveryType	NMTOKEN	Modified data type.
	ArtHandling ?	Enumera-tionSpan	New
	DeliveryCharge ?	Enumera-tionSpan	New
	PreflightOutput ?	URL	New
	PreflightStatus ?	enumeration	New
	ReturnMethod ?	NameSpan	New
	Transfer ?	Enumera-tionSpan	New
	Company ?	refelement	Deprecated
	Component ?	refelement	Deprecated
	Contact *	refelement	New
	Tool ?	refelement	New
7.1.3 BindingIntent: Re-source Structure	*BackCoverColor ?*	Enumera-tionSpan	New
	BindingOrder ?	enumeration	New
	AdhesiveBinding ?	element	Deprecated
	BindList ?	element	New
	BookCase ?	element	Deprecated
	EdgeGluing ?	element	New
	HardCoverBinding ?	element	New
	SoftCoverBinding ?	element	New
	Tape ?	element	New

Location	Name	Data Type	Comment
	StripBinding ?	element	New
	VeloBinding ?	element	Renamed to StripBinding.
Structure of BindList Subelement			New section.
Structure of BindItem Subelement			New section
Structure of the AdhesiveBinding Subelement			Deprecated section.
Structure of the BookCase Subelement			Deprected section.
Structure of the EdgeGluing Subelement			New section.
Structure of the HardCover-Binding Subelement			New section.
Structure of the RingBinding Subelement	*HoleType*	Enumera-tionSpan	New
	RingSystem	NameSpan	Deprecated
Structure of the PlasticComb Subelement	*PlasticCombType ?*	NameSpan	modified
Structure of the SaddleStitch-ing Subelement	*StitchNumber ?*	IntegerSpan	New
Structure of the SoftCover-Binding Subelement			New section.
Structure of the Tape Subelement			New section.
Structure of the VeloBinding Subelement			Renamed to StripBinding
7.1.4 ColorIntent: Resource Properties	Partition		Modified
Resource Structure	*Coatings ?*	StringSpan	Modified data type.
	ColorPool ?	refelement	New
7.1.5 DeliveryIntent: Re-source Structure	*DeliveryCharge ?*	Enumera-tionSpan	New
	Pickup ?	boolean	Deprecated
	ReturnMethod ?	NameSpan	New
	SurplusHandling ?	Enumera-tionSpan	New
	Transfer ?	Enumera-tionSpan	New
	Company ?	refelement	Deprecated
	Contact *	refelement	New
Structure of DeliveryIntent Elements: DropIntent	*Pickup ?*	boolean	Deprecated
	ReturnMethod ?	NameSpan	New
	SurplusHandling ?	Enumera-tionSpan	New
	Transfer ?	Enumera-tionSpan	New
	Company ?	refelement	Deprecated
	Contact *	refelement	New

Location	Name	Data Type	Comment
Structure of the DropItemIntent Subelement	*Proof ?*	string	New
	Component	refelement	Deleted. Replaced by Physical-Resource, which has Component as an instance.
	PhysicalResource ?	refelement	New
Contents of the CreditCard Subelement			New section.
Contents of the Payment Subelement			New section
Contents of the Pricing Subelement	Payment ?	element	New
7.1.6 EmbossingIntent			New section
7.1.7 FoldingIntent: Resource Structure	*Folds ?*	XYPair	Deprecated
	Fold *	element	New
7.1.8 HoleMakingIntent: Resource Structure	*HoleReferenceEdge ?*	enumeration	New
	HoleType	StringSpan	Modified data type; some values are deprecated.
Structure of the HoleList Subelement	Hole *	refelement	Modified data type.
	HoleLine *	refelement	New
Structure of the Hole Subelement			Deleted section. Moved to **Hole** resource.
7.1.9 InsertingIntent: Structure of Insert Subelement	*SheetOffset ?*	XYPair	Deprecated
	WrapPages ?	Integer-RangeList	New
	GlueLine *	element	New
7.1.10 LaminatingIntent: Resource Structure	*Laminated*	OptionSpan	Deprecated
7.1.11 LayoutIntent: Resource Structure	*Dimensions ?*	XYPairSpan	New
	FinishedDimensions ?	ShapeSpan	New
	FinishedPageOrientation ?	enumeration	Deprecated. Page orientation is implied by the value of *Dimensions* and *FinishedDimensions.*
	FolioCount ?	enumeration	New
	Pages ?	IntegerSpan	New
	PageVariance ?	IntegerSpan	New
	Layout ?	refelement	New
7.1.12 MediaIntent: Resource Properties	Resource reference by		Modified
Resource Structure	HoleType ?	StringSpan	New.
	HoleCount ?	IntegerSpan	Deprecated
	Thickness ?	NumberSpan	New
7.1.16 ProofingIntent: Resource Properties	Partition		Modified
	ProofItem *	element	New
	PageIndex ?	Integer-RangeList	New
	ProofName ?	string	New
Structure of the ProofItem Element	SeparationSpec *	Enumeration-Span	New

Location	Name	Data Type	Comment
	Amount ?	IntegerSpan	Moved
	BrandName ?	StringSpan	Moved
	ColorType ?	Enumera-tionSpan	Moved
	Contract ?	boolean	Moved
	HalfTone ?	OptionSpan	Moved
7.1.17 ShapeCuttingIntent: Structure of ShapeCut Subele-ment	CutType ?	Enumera-tionSpan	Modified data type
	ShapeDepth ?	NumberSpan	New
	ShapeType	Enumera-tionSpan	New
7.1.18 SizeIntent			deprecated section
7.2.3 AdhesiveBinding-Params			Deprecated section. Split into: CoverApplicationParams, GlueAp-plication, SpinePreparationParams, SpineTapingParams.
7.2.4 ApprovalParams: Re-source Properties	Output of processes		Modified
7.2.5 ApprovalSuccess: Resource Properties	Partition, Output of proc-esses		Modified
7.2.7 BlockPreparation-Params			New section.
7.2.8 BoxPackingParams			New section.
7.2.9 BufferParams			New section.
7.2.10 Bundle			New section.
7.2.12 CaseMakingParams			New section
7.2.13 CasingInParams			New section.
7.2.15 CIELABMeasuringField: Resource Properties	Resource referenced by, Output of processes		Modified
Resource Structure	Diameter ?	double	Modified to optional
	DensityStandard ?	enumeration	Deprecated
	Light	NMTOKEN	Deprecated
	Observer	integer	Deprecated
	Setup ?	string	Deprecated
	Tolerance ?	double	Modified to optional
	ColorMeasurement-Conditions ?	refelement	New
7.2.18 Color: Resource Struc-ture	ColorName ?	NamedColor	New
	ColorMeasurement-Conditions ?	refelement	New
	TransferCurve *	refelement	modified data type to refelement, removed TransferCurve subele-ment which is now a resource.
7.2.19 ColorantControl: Resource Properties	Resource referenced by, Partition, Input of Proc-esses, Output of processes		Modified
7.2.20 ColorControlStrip: Resource Properties	Output of processes		Modified

Location	Name	Data Type	Comment
Resource Structure	CIELABMeasuringField *	refelement	New
	DensityMeasuringField *	refelement	New
7.2.21 ColorCorrectionParams: Resource Properties	Partition		Modified
Resource Structure	FileSpec ? (assumed characterization of CMYK, RGB, and Gray)	refelement	Deprecated
7.2.22 ColorMeasurement-Conditions			New section.
7.2.24 ColorSpaceConversion-Params:: Resource Properties	Partition		Modified
Resource Structure	FileSpec ? (assumed characterization of CMYK, RGB, and Gray)	refelement	Deprecated
	PreserveBlack ?	boolean	New
7.2.26 Company: Resource Properties	Resource referenced by		Modified
Resource Structure	Contact *	refelement	Deprecated
7.2.27 Component : Resource Structure	*Overfold ?*	double	New
	OverfoldSide ?	enumeration	New
	ReaderPageCount ?	integer	New
	SurfaceCount ?	integer	New
	Transformation ?	matrix	Deprecated
	Bundle ?	refelement	New
7.2.28 Contact: Resource Properties	Resource referenced by		Modified
Resource Structure	Company ?	refelement	New
7.2.29 ContactCopyParams			New section.
7.2.30 Conventional-PrintingParams: Resource Properties	Partition		Modified
Resource Structure	*ModuleAvailableIndex ?*	Integer-RangeList	New
	PerfectingModule ?	integer	New
7.2.32 CoverApplicationParams:			New. Replaces CoverApplication.
Resource Structure	GlueApplication *	refelement	New
7.2.33 CreasingParams			New section. Replaces *Crease* Subelement of *FoldingParams*.
7.2.34 CutBlock: Resource Properties	Resource referenced by, Input of processes, Output of processes		Modified
7.2.35 CutMark: Resource Properties	Resource referenced by, Input of processes, Output of processes		Modified
Resource Structure	*Blocks ?*	NMTOKENS	Modified to optional

Location	Name	Data Type	Comment
7.2.36 CuttingParams			New section. Replaces *Cut* Subelement of *FoldingParams*.
7.2.41 DeliveryParams: Re-source Structure	*Earliest ?*	dateTime	Modified data type
	Required ?	dateTime	Modified data type
	Company ?	refelement	Deprecated
	Contact *	refelement	New
Structure of the Drop Subele-ment	*Earliest ?*	dateTime	Modified data type
	Required ?	dateTime	Modified data type
	Company ?	refelement	Deprected
	Contact *	refelement	New
7.2.42 DensityMeasuringField: Resource Properties	Output of processes		Modified
Resource Structure	ColorMeasurement-Conditions ?	refelement	New
7.2.43 DevelopingParams			New section
7.2.44 Device: Resource Structure	*DeviceFamily ?*	string	Deprecated. Replaced by the ap-propriate *ModelXXX* attributes
	Directory ?	URL	New
	FriendlyName ?	string	New
	JDFVersions ?	string	New
	JMFSenderID ?	string	New
	JMFURL ?	URL	New
	Manufacturer ?	string	New
	ManufacturerURL ?	string	New
	ModelDescription ?	string	New
	ModelName ?	string	New
	ModelNumber ?	string	New
	ModelURL ?	string	New
	SerialNumber ?	string	New
	PresentationURL ?	string	New
	UPC ?	string	New
	DeviceCap *	element	New
	IconList ?	element	New
Structure of the IconList Subelement			New section.
Structure of the Icon Subele-ment			New section.
7.2.45 DigitalPrintingParams Re-source Properties	Partition		Modified
Resource Structure	*Collate ?*	enumeration	New
	OutputBin ?	MNTOKEN	New
	ManualFeed ?	boolean	New
	PageDelivery ?	enumeration	New
	PrintQuality ?	enumeration	Deprecated
	PrintingType ?	enumeration	Modified to optional

Location	Name	Data Type	Comment
	Component ?	refelement	New
	Disjointing ?	refelement	New
	Media ?	refelement	New
	MediaSource ?	refelement	Deprecated.
7.2.46 Disjointing: Resource Properties	Resource referenced by		Modified
Resource Structure	*Overfold ?*	double	Deprecated
	IdentificationField *	element	Modified to optional multiple
7.2.47 DividingParams			Deprecated section.
Resource Properties	Partition		Modified
7.2.48 EmbossingParams			New section.
7.2.51 ExposedMedia: Re-source Properties	Partition, Input of proc-esses, Output of processes		Modified
7.2.52 FileSpec: Resource Structure	*CheckSum ?*	integer	New
	FileVersion ?	string	New
	UID ?	string	New
7.2.53 FitPolicy			New section.
7.2.54 Fold			New section. Replaces *Fold* Subelement of *FoldingParams*
7.2.55 FoldingParams: Re-source Properties	Partition, Output of proc-esses		Modified. Split into CuttingPar-ams, CreasingParams, Fold, Glu-ingParams, PerforatingParams, ThreadSealingParams.
Resource Structure	*FoldSheetIn ?*	XYPair	Deprecated
	Fold *	element	Modified to new section.
	FoldOperation *	element	Deprecated
7.2.56 FontParams: Re-source Properties	Partition		Modified
7.2.57 FontPolicy: Resource Properties	Partition, Input of proc-esses		Modified
7.2.58 FormatConversionParams			New section.
7.2.60 GlueApplication			New section.
7.2.61 GluingParams			New section. Replaces *Glue* Subelement of *FoldingParams*.
7.2.62 GlueLine: Resource Properties	Resource referenced by		Modified
Resource Structure	*AreaGlue ?*	boolean	New
7.2.63 HeadBand-ApplicationParams			New section
7.2.64 Hole			New section
7.2.65 HoleLine			New section
7.2.66 HoleMakingParams: Resource Properties	Input of processes		Modified
Resource Structure	*Center ?*	XYPair	Modified to optional
	CenterReference ?	enumeration	New
	HoleReferenceEdge ?	enumeration	New
	HoleType	enumerations	New. Some values are deprecated.

Location	Name	Data Type	Comment
	Shape ?	enumeration	Modified to optional.
	HoleLine *	element	New
	RegistrationMark ?	refelement	New
7.2.67 IdentificationField: Resource Properties	Resource referenced by, Output of processes		Modified
Resource Structure	*Value ?*	string	New
7.2.68 IDPrintingParams			Deprecated section.
Resource Properties	Partition		Modified
Structure of the Cover Subelement			Deprecated section.
Properties of the IDPFinishing Subelement			Deprecated section.
Structure of IDPFolding Subelement			Deprecated section.
Structure of IDPHoleMaking Subelement			Deprecated section
Structure of the IDPLayout Subelement			Deprecated section.
Structure of IDPStitching Subelement			Deprecated section.
Structure of IDPTrimming Subelement			Deprecated section.
Structure of the ImageShift Subelement			Deprecated section.
Structure of the JobSheet Subelement			Deprecated section.
7.2.69 ImageCompression-Params : Resource Properties	Partition, Input of processes		Modified
Resource Structure	*EncodeColorImages ?*	boolean	Deprecated
	EncodeImages ?	boolean	New
7.2.70 ImageReplacement-Params**:** Resource Properties	Partition, Input of processes		Modified
Resource Structure	*MaxResolution ?*	double	Deprecated. Replaced with a link to *ImageCompressionParams* in the process.
	ResolutionReduction-Strategy ?	enumberation	Deprecated. Replaced with a link to *ImageCompressionParams* in the process.
	IgnoreExtensions ?	NMTOKENS	Modified to optional.
	FileSpec +	refelement	New
	SearchPath +	telem	Deprecated
7.2.71 ImageSetterParams: Resource Structure	*BurnOutArea ?*	XYPair	New
	Media ?	refelement	New
7.2.72 Ink: Resource Structure	InkName ?	string	Modified to optional.
7.2.75 InsertingParams: Resource Structure	*SheetOffset*	XYPair	Deprecated
7.2.76 InsertSheet: Resource Properties	Resource referenced by		Modified
Resource Structure	*MarkList ?*	NMTOKENS	New

Location	Name	Data Type	Comment
	SheetFormat ?	NMTOKEN	New
	SheetType	enumeration	New
	SheetUsage	enumeration	New
	Usage	enumeration	Renamed to *SheetUsage* and modified.
7.2.78 InterpretingParams: Resource Properties	Partition		Modified
Resource Structure	*FitToPage ?*	boolean	Deprecated
	PrintQuality ?	enumeration	New
	FitPolicy ?	refelement	New
	Media ?	refelement	New
	PDFInterpretingParams ?	refelement	New
7.2.79 JacketingParams			New section.
7.2.80 JobField			New section.
7.2.81 LabelingParams			New section.
7.2.82 LaminatingParams			New section.
7.2.83 Layout: Resource Properties	Input of processes		Modified
Resource Structure	*MaxDocOrd ?*	integer	New
	MaxSetOrd ?	integer	New
	Name ?	string	New
	LayerList ?	element	New
	Media ?	refelement	New
	MediaSource ?	refelement	Deprecated
	TransferCurvePool ?	refelement	New
Structure of LayerList Subelement			New section.
Structure of LayerDetails Subelement			New section
Structure of Signature Subelement	Media ?	refelement	New
	MediaSource ?	refelement	Deprecated
7.2.84 LayoutElement: Resource Properties	Output of processes		Modified
Resource Structure	*IgnorePDLCopies ?*	boolean	New
	IgnorePDLImposition ?	boolean	New
7.2.85 LayoutPreparation-Params			New section.
7.2.86 Longitudinal-RibbonOperationParams			Deprecated section.
Resource Properties	Partition		Modified
Structure of LongitudinalRibbonOperationParams Elements	LROperation		Deprecated section.
	LongFold		Deprecated section.
	LongGlue		Deprecated section.
	LongPerforate		Deprecated section.
	LongSlit		Deprecated section.

Location	Name	Data Type	Comment
7.2.87 ManualLaborParams			New section.
7.2.88 Media: Resource Properties	Resource reference by, Input of processes		Modified
Resource Structure	*ColorName ?*	string	New
	Dimension ?	XYPair	Modified to optional
	GrainDirection ?	enumeration	New
	HoleCount ?	integer	Deprecated
	HoleType ?	enumerations	New
	MediaColorName ?	NamedColor	Modified data type.
	ShrinkIndex ?	XYPair	New
	StockType ?	NMTOKEN	New
	Texture ?	NMTOKEN	New
	UserMediaType ?	NMTOKEN	Deprecated
	Color ?	refelement	Deprecated, replaced by *Color-Name*.
7.2.89 MediaSource			Deprecated section.
Resource Structure	*SheetLay ?*	enumeration	New
	Component ?	refelement	New
7.2.91 ObjectResolution: Resource Properties	Resource referenced by		Modified
7.2.92 OrderingParams: Resource Structure	Company ?	refelement	Deprecated
	Contact *	refelement	New
7.2.93 PackingParams			Deprecated section.
7.2.94 PalletizingParams			New section.
7.2.95 Pallet			New section.
7.2.96 PDFToPSConversion-Params: Resource Properties	Partition		Modified
Resource Structure	*IgnoreBG ?*	boolean	New
	IgnoreDeviceExtGState ?	boolean	Deprecated
	IgnoreOverprint ?	boolean	New
	IgnoreTransfers ?	boolean	New
	IgnoreUCR ?	boolean	New
7.2.98 PerforatingParams			New section. Replaces *Perforate* Subelement of *FoldingParams*.
7.2.101 PlasticCombBindingPar-ams	*Type ?*	enumeration	Modified list
7.2.102 PlateCopyParams			Deprecated section.
7.2.103 PreflightAnalysis: Structure of PreflightInstance Subelement	*PageRefs*	Integer-RangeList	Modified data type.
	PreflightInstanceDetail	element	Properties renamed to Pre-flightInstanceDetail
7.2.106 Preview: Resource Properties	Resource referenced by, Partition		Modified
Resource Structure	*CTM ?*	matrix	New

Location	Name	Data Type	Comment
	Directory ?	URL	New
7.2.107 PreviewGeneration-Params: Resource Properties	Partition		Modified
Resource Structure	*AspectRatio ?*	enumeration	New
	PreviewType	enumeration	Deleted. Replaced by *PreviewUsage ?*
	PreviewUsage ?	enumeration	New
	ImageSetterParams ?	refelement	New
7.2.108 ProofingParams: Resource Properties	Partition		Modified
Resource Structure	*ManualFeed?*	boolean	New
	ProofRenderingIntent ?	enumeration	New
	Media ?	refelement	New
7.2.109 PSToPDFConversionParams: Resource Properties	Partition, Input of processes		Modified
Resource Structure	*InitialPageSize ?*	XYPair	New
	IntialResolution ?	XYPair	New
Structure of AdvancedParams Subelement	*PreserveHaltoneInfo ?*	boolean	New
	PreserveOverprintSettings ?	boolean	New
	TransferFunctionInfo ?	enumeration	New
	UCRandBGInfo ?	enumeration	New
7.2.110 RegisterMark: Resource Properties	Output of processes		Modified
Resource Structure	*MarkUsage ?*	enumerations	New
7.2.111 RegisterRibbon			New section.
7.2.112 RenderingParams: Resource Properties	Partition, Input of processes		Modified
Resource Structure	Media ?	refelement	New
7.2.113 Resource-DefinitionParams: Resource Structure	*DefaultID ?*	NMTOKEN	Deprecated
	ResourceParam +	refelement	New
Structure of the ResourceParam Subelement			New section.
7.2.114 RingBindingParams: Resource Structure	*RingSystem ?*	enumeration	Deprecated
7.2.115 RunList: Resource Properties	Partition		Modified
Resource Structure	*DocCopies ?*	integer	New
	EndOfDocument ?	boolean	New
	EndOfSet ?	boolean	New
	NDoc ?	integer	New
	NSet ?	integer	New
	PageCopies ?	integer	New
	RunTag ?	NMTOKEN	New
	SetCopies ?	integer	New

Location	Name	Data Type	Comment
	SetNames ?	NameRange-List	New
	Sets ?	IntegerRan-geList	New
7.2.116 SaddleStitchingParams			Deprecated section
7.2.118 ScavengerArea			New section.
7.2.119 ScreeningParams: Resource Properties	Input of processes		Modified
Resource Structure	ScreenSelector *	element	Modified to optional multiple
Structure of ScreenSelector Subelement	*AngleMap ?*	string	New
	DotSize ?	double	New
7.2.122 ShapeCuttingParams			New section.
7.2.123 Sheet: Resource Properties	Input of processes		Modified
Resource Structure	Media ?	refelement	New
	MediaSource ?	refelement	Deprecated
7.2.124 ShrinkingParams			New section.
7.2.125 SideSewingParams			Deprecated section
7.2.126 SpinePreparationParams			New section. Replaces BackPrepa-ration.
7.2.127 SpineTapingParams			New. Replaces SpineTaping.
Resource Structure	GlueApplication *	refelement	New
7.2.128 StackingParams			New section.
7.2.129 StitchingParams: Resource Properties	Resource referenced by		Modified
Resource Structure	*ReferenceEdge ?*	enumeration	New
7.2.130 Strap			New section
7.2.131 StrappingParams			New section
7.2.132 StripBindingParams			New section
7.2.133 Surface: Structure of the Abstract PlacedObject Subelement	*LayerID ?*	integer	New
	OrdID ?	integer	New
	Trim CTM ?	matrix	New
	Type	enumeration	Deprecated
Structure of ContentObject Subelement	*DocOrd ?*	integer	New
	SetOrd ?	integer	New
Structure of MarkObject Ele-ments	*LayoutElement-PageNum ?*	integer	New
	ColorControlStrip *	refelement	Modified to optional multiple
	CutMark *	refelement	Modified to optional multiple
	DensityMeasuringField *	refelement	Modified to optional multiple
	DeviceMark ?	refelement	New
	JobField *	refelement	New
	RegisterMark *	refelement	Modified to optional multiple

Location	Name	Data Type	Comment
	ScavengerArea *	refelement	New
Structure of DeviceMark Subelement			New section
Structure of DynamicField Subelement	*InputField ?*	string	Deprecated
	DeviceMark ?	refelement	New
7.2.134 ThreadSealingParams			New section. Replaces *Thread-Seal* Subelement of *FoldingParams*.
7.2.135 ThreadSewingParams: Resource Structure	*Offset ?*	double	New
7.2.137 Tool			New section.
7.2.138 TransferCurve			Moved from Structure of TransferCurvePool Subelement and made resource
	CTM ?	matrix	New
	Name	NMTOKEN	Moved from Structure of TransferCurvePool Subelement
7.2.139 TransferCurvePool: Structure of TransferCurvePool Subelement			Deleted section. Contents moved to Structure of TransferCurveSet Subelement.
Structure of TransferCurve Subelement	*Curve*	Transfer-Function	Moved from Structure of TransferCurveSet Subelement
	Separation ?	string	Moved from Structure of TransferCurveSet Subelement
7.2.140 TransferFunctionControl: Resource Properties	Resource referenced by, Input of processes		Modified
7.2.141 TrappingDetails: Resource Properties	Resource referenced by, Partition, Input of processes		Modified
Resource Structure	ObjectResolution *	refelement	New
7.2.142 TrappingParams: Resource Properties	Resource referenced by, Partition		Modified
7.2.143 TrapRegion	Input of processes		Modified
7.2.144 TrimmingParams: Resource Structure	*TrimmingType*	enumeration	New
VeloBindingParams			Deleted section. Renamed to StripBindingParams
7.2.146 WireCombBindingParams: Resource Structure	*FlipBackCover ?*	boolean	New
7.2.147 WrappingParams			New section.
7.3 Device Capability Definitions			New section.

Appendix N Table of Tables

Appendix O Terminology Usage

This document contains many terms specific to its interpretation and intent. Many of the terms are described in relation to various processes, components, and values throughout the document. The more prominent terms are listed below to make it easier for the casual user to locate precise definitions and usage.

Term	Term Type	Glossary of Terminology (Sect. 1.4)	Data Structures (Sect. 1.5)	Job Components (Sect. 2.1.1)	Workflow Components (Sect. 2.1.2)	Relation ships (Sect. 2.1.1.4)	Other
Acknowledge	message						Section 5.2.1.5
Activation	enumeration						Table 3.3, Table 5.38
Agent(s)	consumer	X			Section 2.1.2.3		
Ancestor	element					X	
AncestorPool	element						Sect. 3.3 Table 3.4
Attribute(s)	attribute	X		Section 2.1.1.3			Sect. 3.1.2
AuditPool	elements						Sect. 3.10
Big job		X					
boolean	data type		X				Table A.1
Branch	node					X	
Child	element					X	
Class	data type	X					
CMYK color	data type		X				A.2.1
Command	message						Section 5.2.1.4
Controllers	consumer	X			Section 2.1.2.4		
Coordinate systems							Section 2.5
Customer	node						Section 3.4
Date	data type		X				Table A.1
DateTime	data type		X				Table A.1
Default	value	X					Sect. 1.4.2.1
Deprecated		X					
Descendent	element					X	
Devices	consumer	X			Section 2.1.2.2		
Document set		X					
Double	data type		X				Table A.1
Duration	data type		X				Table A.1
DurationRange	data type		X				A. 2.2
Element(s)	job component	X	X	Section 2.1.1.2			
Enumeration(s)	data type		X				
Finished page	job component	X					
gYearMonth	data type		X				Table A.1
ID/IDREF(s)			X				Table A.1
IfraTrack modeling							App. E
Instance document	job component	X					
Integer	data type		X				Table A.1
IntegerList	data type		X				A.2.3

Term	Term Type	Glossary of Terminology (Sect. 1.4)	Data Structures (Sect. 1.5)	Job Components (Sect. 2.1.1)	Workflow Components (Sect. 2.1.2	Relation ships (Sect. 2.1.1.4)	Other
IntegerRange	data type		X				A.2.4
IntegerRangeList	data type		X				A.2.5
intent resources							3.2.1, 7.1.1.1
IPP mapping							App. F
iterative processing							2.3
JDF consumer		X					
JMF		X					Chapt. 5
Job(s)	job component	X		Section 2.1.1.1			
Job part	node	X					
LabColor	data type		X				A.2.6
Language	data type		X				Table A.1
Leaf	element					X	
Links	job components	X		Section 2.1.1.5			A.3.1
Machines	job components	X			Section 2.1.2.1		
Matrix	data type		X				A.2.7
Merging	*process*						Section 4.4
MIME File Packaging							A.4.1
MIS		X			Sesction 2.1.2.5		
NamedColor	data type		X				A.2.8
NameRange	data type		X				A.2.9
NameRangeList	data type		X				A.2.10
NMTOKEN(S)	data type		X				Table A.1
Node(s)	element	X		Section 2.1.1.1			Table 3.3
Number	data type		X				
NumberList	data type		X				A.2.11
NumberRange	data type		X				A.2.12
NumberRangeList	data type		X				A.2.13
Parent	element					X	
Partitioned resource	**resource**	X					
Path	data type		X				A.2.14
PDL		X					
PJTF conversion							App. C
PNG format							A.4.3
PPF conversion							App. D
Process	consumer	X					
Process nodes							Section 3.2 Chapter 6
Product intent nodes	node						Section 3.2.1
Query	message						Section 5.2.1.1
Queue	consumer	X					
Reader page	value	X					
Rectangle	data type		X				A.2.15
Refelement	data type		X				
Relationships	job components			Section 2.1.1.4			

Term	Term Type	Glossary of Terminology (Sect. 1.4)	Data Structures (Sect. 1.5)	Job Components (Sect. 2.1.1)	Workflow Components (Sect. 2.1.2	Relation ships (Sect. 2.1.1.4)	Other
Resource(s)	job component	X					
Response	message						Section 5.2.1.2
Root	element					X	
Shape	data type		X				
ShapeRange	data type		X				A.2.16
ShapeRangeList	data type		X				A.2.17
Sibling	element					X	
Signal	message						Section 5.2.1.3
Small job		X					
Spawning	*process*						Section 4.4
sRGBcolor	data type		X				A.2.18
String	data type		X				Table A.1
Support	value	X					
System interaction	job components				Section 2.1.2.6		
Tag	value	X					
Telem	data type		X				
Text	data type		X				
TimeRange	data type		X				A.2.19
TransferFunction	data type		X				A.2.20
URI	data type		X				Table A.1
URL	data type		X				Table A.1
Work center		X					
Workflow components	job components				Section 2.1.2		
XYPair	data type		X				A.2.21
XYPairRange	data type		X				A.2.22
XYPair/RangeList	data type		X				A.2.23

Appendix P Errata

The following section summarizes errata that were found after publication of JDF 1.1. Note that trivial changes such as font changes are not tracked in this table. Although the table may seam quite long, the authors spent a great deal of effort in ensuring that changes were as transparent as possible to implementations of JDF 1.1. The bulk of changes consists of clarifications of ambiguities. Modifications that require a change to the XML schema are the exception and limited to situations where implementation would have otherwise been inhibited.

Location	Date	Comments
Table 3-19 General contents of the PartAmount element	May 2 2002-	Table grid formatting
3.9.2 Description of Partitionable Resources		Missing quotes in example added
M.1 New Items		*RunTag* has data type NMTOKENS
Table 3-3 Contents of a JDF node		*Activation, Status.* added Modified in JDF 1.1 flag.
3.7 Resources		*UpdateID* has data type NMTOKEN
Table 3-31 Redundant table removed	May 3 2002-	Removed redundant table (it was part of table 3-30) The heading was kept to avoid renumbering
7.2.88 Media	May 6 2002	Put deprecated **Color** refelement back. It had accidentally been removed instead of deprecated.
	June 24 2002	*Grade*: Definition modified to refer to ISO 12647-2 ff
K.5 Conversion of PPF to JDF	May 8 2002	Modified example **ColorantControl** to include *ProcessColorModel* and ColorantParams
3.9.2 Description of Partitionable Resources	May 8 2002	Modified example #4 of the ResourceRef to contain explicit Part elements.
		Added clarifying text on how the Part elements are combined.
3.8.6 Inter-Resource Linking Using ResourceRef	May 15 2002	Added a restriction that the Part element in a ResourceRef must reference a resource leaf.
Table 3-25 Contents of the Part element	May 16 2002	*PreviewType.* added New in JDF 1.1 flag.;
		Reordered *DocCopies* and *DocIndex*
7.2.127 SpineTapingParams	May 16 2002	Added a ? to *HorizontalExcess, StripLength, TopExcess*
A.2.17 ShapeRange A.2.23 XYPairRange	May 21 2002	Clarified definition of reverse order and changed < to <= in the algorithm definition.
4.4.1 Case 1: Standard Spawning and Merging	May 22 2002	Clarified the attributes that must be left when removing the spawned node from the parent node.
Table 3-9 Contents of the PartStatus element Table 3-19 General contents of the PartAmount element	May 22 2002	Clarified that the Part element in a PartStatus or PartAmount element must refer to a leaf resource.
7.2.122 ShapeCuttingParams	June 3 2002	Data type of *ShapeDepth* changed from NumberSpan to double. (Copy/Paste error from ShapeCut)
7.1.3 BindingIntent	June 3 2002	Removed default reference in BindList
7.2.133 Surface	June 6 2002	PlacedObject::*ClipPath* Replaced clip path with clipping rectangle in the description.
	July 9 2002	PlacedObject::*Ord* Clarified usage and added example tables.
	August 9 2002	PlacedObject::*Ord* Clarified zero based.
	August 21 2002	Added remark that partitioning is discouraged.

Location	Date	Comments
7.3.3 Structure of the DevCaps Subelement	June 11 2002	DevCaps::*DevNS*. Data type changed to URI. Added *GenericAttributes*.
	August 28 2002	*GenericAttributes now has a ?*
7.3.4 Structure of the DevCap Subelement	June 11 2002	Removed *Restricted, Supported*
	August 28 2002	Added *DevNS*
7.3.5 Structure of the Abstract State Subelement	June 11 2002	Removed *Restricted, Supported* Added *Span*
	August 5 2002	Removed *DataType*
	August 28 2002	Added *DevNS*
7.3.5.1 Structure of the BooleanState Subelement	June 11 2002	Added AllowedValueList.
7.3.6 Examples of Device Capabilities	June 26 2002	Modified example to reflect the modifications in the JDF device capabilities.
7.2.80 JobField	June 11 2002	JobField::*ShowList FriendlyName* removed blank from enumeration.
7.2.85 LayoutPreparation-Params	June 13 2002	*CreepValue* is optional.
	June 21 2002	PageCell::*TrimSize* Clarified default to be **Layout-PreparationParams**:*SurfaceContentsBox.*
5.5.1.3 KnownDevices	June 14 2002	Replace ResponseTypeObj **Device *** with DeviceList ?
	August 14 2002	Replace contents of DeviceList with DeviceInfo.
7.2.107 PreviewGeneration-Params	June 19 2002	Synchronized *PreviewType* value list with the *PreviewType* partition key.
5.6.3.5 QueueEntryStatus	June 21 2002	The QueryTypeObj was changed from QueueEntryDef to QueueEntryDefList in order to resolve a type collision in the XML schema.
6.5.26 Palletizing	June 21 2002	Fixed copy-paste error in the description of the input **Component**.
7.1.12 MediaIntent	June 24 2002	*Grade*: Definition modified to refer to ISO 12647-2 ff
3.8.6.2 Alignment of Re-sourceLink and Re-sourceRef	June 26 2002	Added Section.
7.2.52 FileSpec	June 26 2002	*CheckSum*: "RSA MD" now completed to "RSA MD5" Changed data type to hexBinary.
	July 10 2001	Clarified usage of *FileTemplate* and *FileFormat* when UID is present.
Table 1-2 JDF data types	June 26 2002	Added data type hexBinary.
A.1 XML Schema Data Types	June 26 2002	Added data type hexBinary.
7.2.78 InterpretingParams	June 27 2002	Clarified usage of *Center* and *Scaling* in conjunction with **FitPolicy**.
6.3.17 Preflight	June 27 2002	Added warning that **Preflight** is under construction.
7.2.53 FitPolicy	June 27 2002	Added clarification for use of *ClipOffset*. Added clarification on aspect ratios for *SizePolicy*. Removed *ResourceUsage*. It had been removed fro 1.1 and reappeared in the editing process. Its functionality is achieved by evaluating the context of **FitPolicy**.

Location	Date	Comments
7.2.42 DensityMeasuring-Field	June 28 2002	*Density* data type modified to NumberList.
K.3 Spawning and Merging	July 1 2002	Added an example of partitioned Spawning and Merging.
7.2.115 RunList	July 9 2002	Modified the 2. example to use **RunList**::*Directory* Added a reference to *Sets* in *Pages*. Removed erroneous flag stating that *EndOfDocument* was new in JDF 1.1. Added a clarifying sentence on documents, pages and sets in the introduction.
6.1 Process Template	July 9 2002	Added **Preview** to the list of optional input resources.
6.5.4 CaseMaking	August 5 2002	Removed ? from **Media (CoverBoard).**
7.2.106 Preview	August 9 2002	Clarified usage of RGB PNG files in previews.
A.4.3 PNG Image Format	August 9 2002	Clarified usage of RGB PNG files in previews.
5.2.1.5 Acknowledge	August 9 2002	Added a ? to Notification.
5.4.1 Pure Event Messages	August 9 2002	Added section.
6.3.1 ColorCorrection	August 9 2002	**ColorantControl** is now an optional input.
6.3.2 ColorSpaceConversion	August 9 2002	**ColorantControl** is now an optional input.
6.3.19 Proofing	August 9 2002	**ColorantControl** is now an optional input.
6.3.25 Separation	August 9 2002	**ColorantControl** is now an optional input.
6.3.26 SoftProofing	August 9 2002	**ColorantControl** is now an optional input.
6.3.28 Trapping	August 9 2002	**ColorantControl** is now an optional input. Wording change for the output **RunList**.
7.2.109 PSToPDFConversionParams	August 14 2002	Renamed misspelled *AutoPostitionEPSInfo* to *Auto-PositionEPSInfo*
7.2.19 ColorantControl	August 16 2002	Clarified usage of *ColorantOrder*.
7.2.83 Layout	August 21 2002	Added remark that partitioning is discouraged.
7.2.123 Sheet	August 21 2002	Added remark that partitioning is discouraged.
8.2.4 MIME Types and File Extensions	August 23 2002	Clarified use of file extensions and renamed MIME type.
3.10 AuditPool	August 23 2002	Clarified use of Audits when creating / modifying a JDF.
5.6.4 Queue-Handling Elements	August 27 2002	Clarified QueueEntry elements for running queues.
5.3 JMF Messaging Levels	August 27 2002	Added an integer level number to the messaging levels.
3.11.1.1 JDF Namespace	August 27 2002	Inserted section header.
3.11.1.2 JDF Extension Namespace	August 27 2002	Inserted section.
3.10.1 Audit Elements	August 27 2002	Inserted extensions for *AgentName*, *AgentVersion*
4.4.5 Case 5: Spawning and Merging of Independent Jobs	August 30 2002	Added some clarifications and removed ambiguous naming of nodes and jobs. Added disclaimer for using case 5.
4.4.3 Case 3: Parallel Spawning and Merging of Partitioned Resources	August 30 2002	Added some clarifications.