

Lecture on

Job Definition Format

„Specification and Schema (XML)“

Walter Kriha

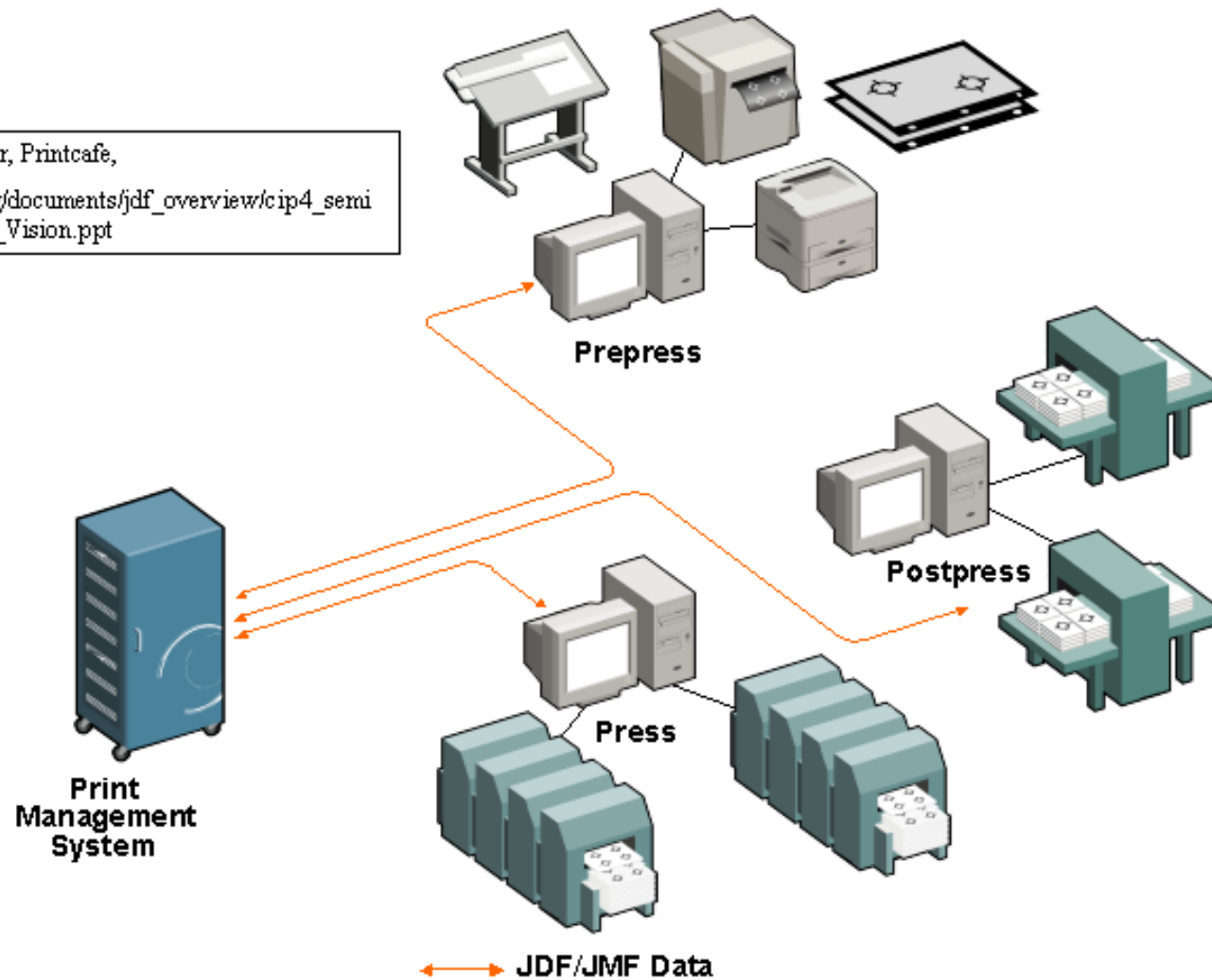
Goals

- Understand the main elements of the Job Definition Format
- Learn how linking is done using ID/IDREF and other mechanisms
- Understand how these elements are used in instances of JDF
- Understand how the elements themselves are modelled using XML-Schema
- Understand the main process ideas behind JDF
- Learn how extension mechanisms are used to adapt JDF.

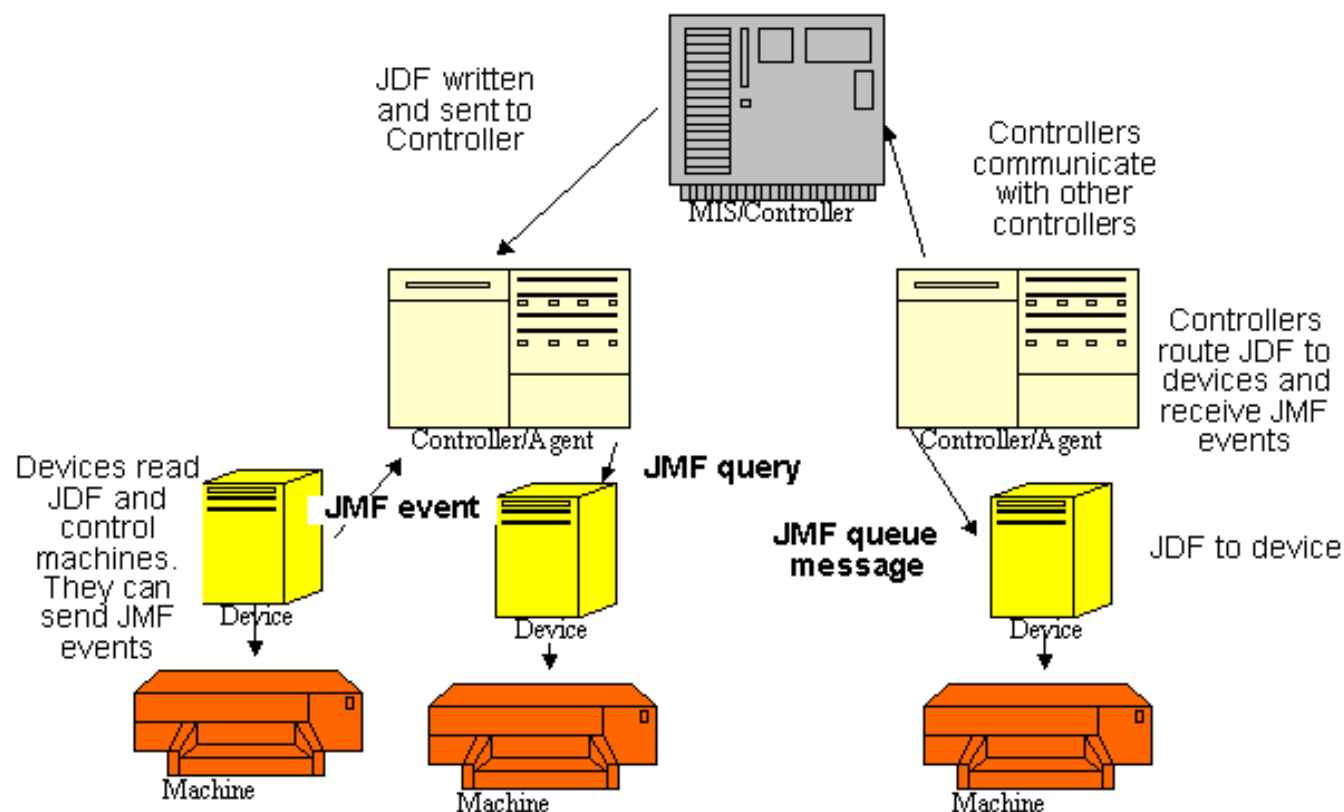
JDF is a large and complex industry schema. Make yourself familiar with its features by going through the documentation at www.cip4.org

The Vision of JDF

from: Doug Belkofer, Printcafe,
http://www.cip4.org/documents/jdf_overview/cip4_seminars_ipex2002/JDF_Vision.ppt



JDF System Components



Controller, Agent and device are purely logical functions. Agents produce JDF. Controller route JDF and devices interpret and execute JDF. Devices can send events and respond to queries about their state. MIS collect this information and control the whole job. See: JDF 1.1 specification, pages 10-13

Tools

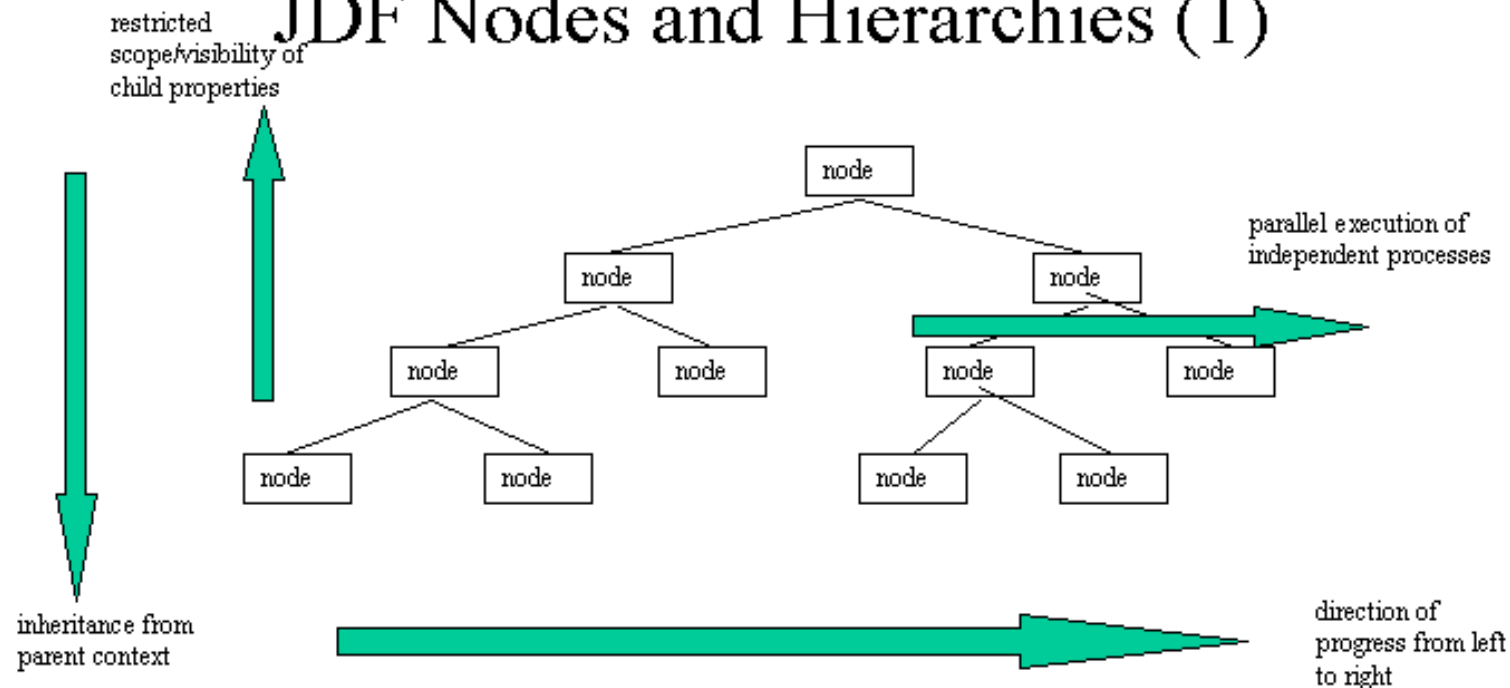
A good schema editing tool is a necessity if you want to learn more about JDF since the schema itself is very large. The use of XML-Schema also creates some problems as some well known XML tools cannot handle it yet (they still expect DTDs).

For all practical purposes XML-Spy seems to be a good tool to work with. You need different views on the schema (tree view, schema view, source view) which is supported nicely.

The menu-item „Extract XML“ is useful to extract single elements from this large schema, e.g. to look at how inheritance is used by JDF.

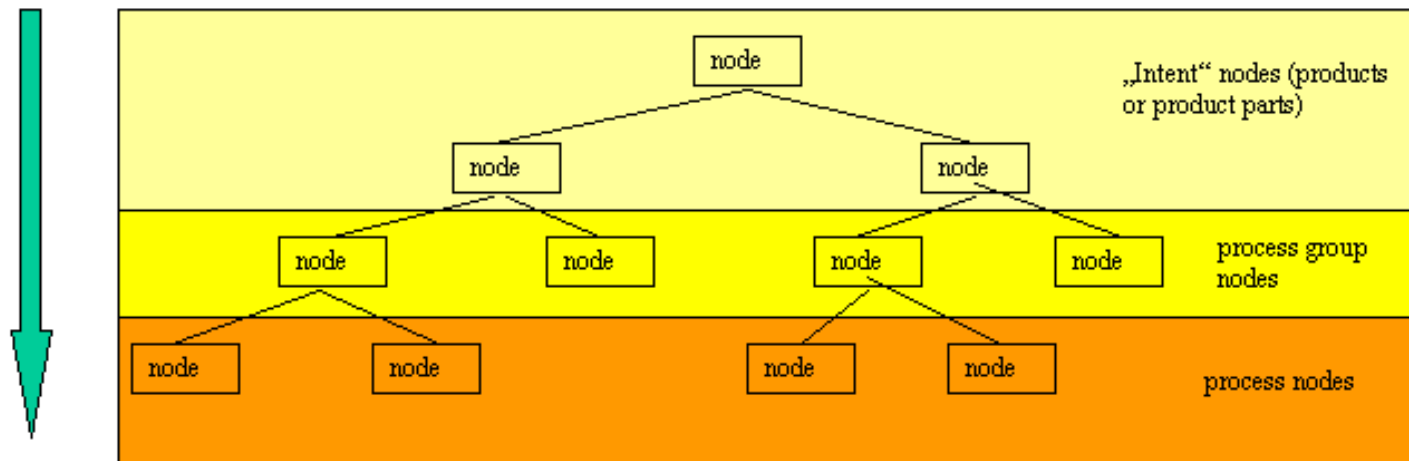
ToDo: what other tools could be useful? I believe a hyperbolic tree viewer would be very nice.
--

JDF Nodes and Hierarchies (1)



The parent/child relationship in a tree can be semantically overloaded in various ways: JDF uses this e.g. to make child elements „inherit“ values from a parent context to prevent child nodes from accessing resources not in their ancestors path. It looks like JDF would also use the horizontal relations to indicate processing flow but this is not true: the node internal resource definitions (input/output) control process flow through a producer/consumer mechanism.

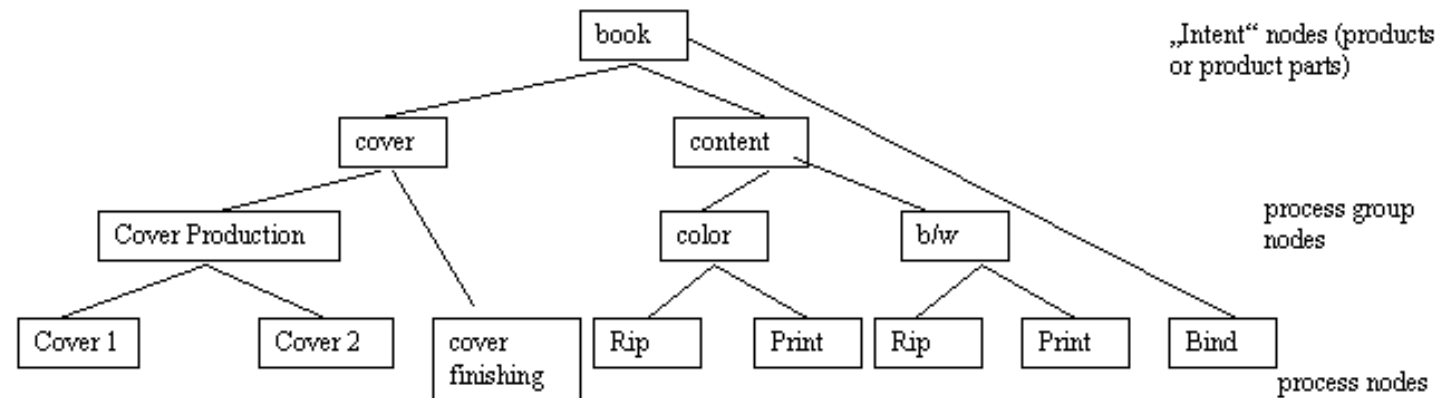
JDF Nodes and Hierarchies (2)



JDF tree grows
during
processing

JDF owns two very peculiar mechanisms as well: The JDF tree usually grows during processing. This is like writing a book by starting with a rough structure and refining it step by step. The advantage is clear: all the information about product and process is contained in one place. The other important feature is a change in node semantic from top to leaf nodes: The top nodes are customer facing, they express product features. The leaf nodes express pure processing.

JDF Nodes and Hierarchies (3)

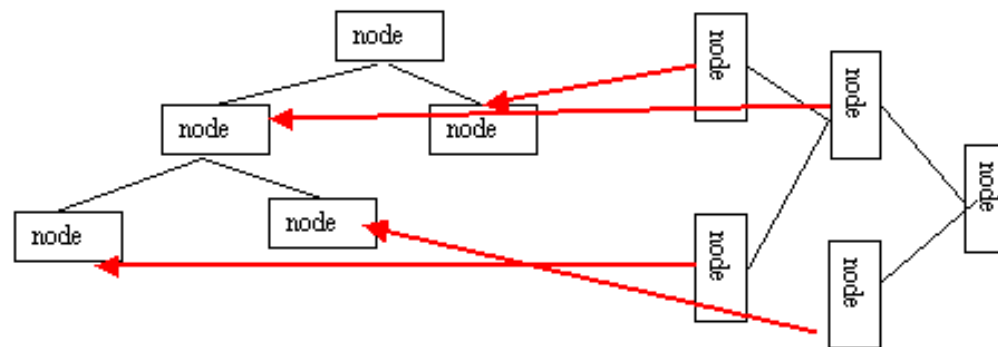


More specific steps are at the bottom of the hierarchy. The parent/child relation controls the visibility of resources for children. „Bind“ e.g. can only use resources for input that are defined at the top node „book“.

JDF Nodes and Hierarchies (4)

„product or product parts

workflow or
process parts



An aspect of a thing or process can be represented as a tree. Several related aspects can be represented as orthogonal trees. JDF decided to NOT encode these relationships explicitly by using separate trees. Instead, JDF uses attributes and types to encode several views in ONE tree representation. The workflow relation e.g. between processes – controlled by resource production and consumption – needs to be created by a JDF processor. It is not explicitly expressed in the JDF tree. Again, the advantage is that everything is contained in one document – everything except the document content itself (text, pictures etc.)

Main JDF Elements (1)

JDF Node:

expresses either products or product parts (e.g. cover, insert) or a production process (e.g. binding). A node can contain child nodes and usually defines resources which are exported or imported from child nodes)

Resource:

anything that is necessary to produce an output: consumables, parameter, components, intents, implementation related things etc. Resources are the central elements of process flow in JDF – even though they are passive.

Resource Link:

JDF distinguishes the definition of a resource within some node and its reference from child nodes. The reference link is a separate construct within JDF and can be qualified as an input or output link.

Resource[Link]Pool:

Container elements which hold either resources or resource links.

Nodes can have more pools, not just resource related pools (e.g. audit pools)

Main JDF Elements (2)

Audit/Audit Pool:

an in-document database which tracks execution flow and process status. Alternatively JDF processors can use JMF messages to report/track progress. Audit elements will contain copies of JDF elements which are referenced but have changed during processing.

Node Info:

contains information about scheduling and routing of messages. Used by MIS for planning, scheduling, invoicing of jobs. (Specification pg. 49)

Customer Info:

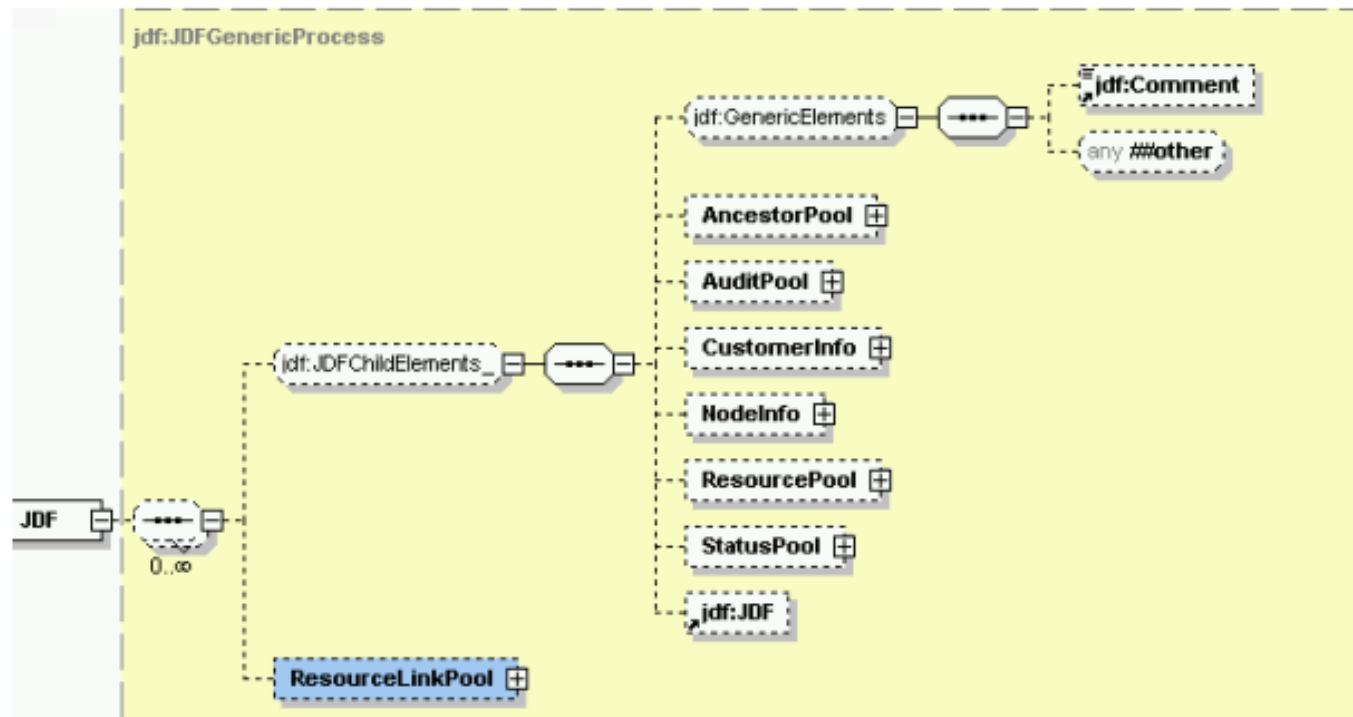
holds information about the company that ordered the job, billing data, customer id and job name etc. Seems to serve as an anchor for resource references in case of spawning jobs.

Status Pool:

Some nodes contain several partitioned resources and can therefore have a different status in these resources. Individual parts are addressed using attributes as parameters (e.g. PartIDKey)

JDF specifies a lot of different process or resource elements, actually, most of the spec. is about those two elements.

JDF Node



The JDF Node is the core element of JDF. It contains other core elements and can be used recursively to extend a specific JDF instance by adding more and more node elements. Rectangles with rounded corners are abstract types which cannot be direct elements of JDF instance files. Diagram generated with XMLSpy.

JDF Node Instance Example

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM21" Type="Product" JobID="HDM2002" Status="Waiting" Version="1.0">
  <ResourcePool>
    <SomeInputResource ID="Link0017" Class="Parameter" Locked="false" Status="Available"/>
    <Component ID="Link0018" Class="Quantity" Locked="false" Status="Unavailable"
      ComponentType="PartialProduct" DescriptiveName="SomeOutputResource"/>
  </ResourcePool>
  <ResourceLinkPool>
    <SomeInputResourceLink rRef="Link0017" Usage="Input"/>
    <ComponentLink rRef="Link0018" Usage="Output" />
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="foo" TimeStamp="2002-04-24T17:21:26+02:00"/>
  </AuditPool>
  <Comment>
    This is a comment
  </Comment>
  <JDF ID="HDM22" ..... </JDF>
</JDF>
```

Every node has an unique ID and several other important attributes of which „status“ is very important.

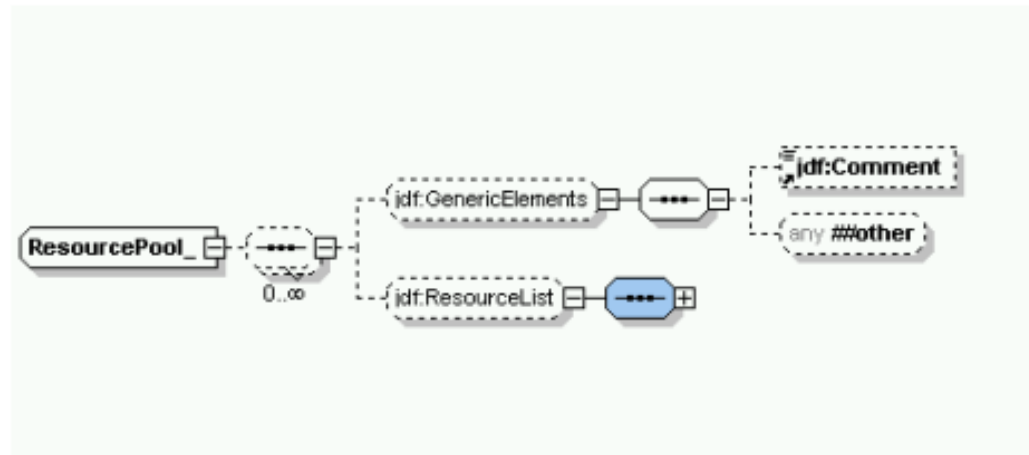
Main JDF node Attributes

Name	Type	Use	Default
CommentURL	jdf:URL	optional	
DescriptiveName	xsd:string	optional	
any ##other			
Activation	jdf:eActivation_	optional	
ID	xsd:ID	required	
JobID	xsd:string	optional	
JobPartID	xsd:string	optional	
Status	jdf:eNodeStatus_	required	
Types	xsd:NMTOKENS	optional	
Version	xsd:string		1.0
Type	xsd:NMTOKEN	required	

ID is the unique identifier of a node, JobID and JobPartID are set by an MIS application to identify jobs. Node types are Product, ProcessGroup, Combined and individual process elements. Status can be „Ready“, „waiting“, „inprogress“ etc.

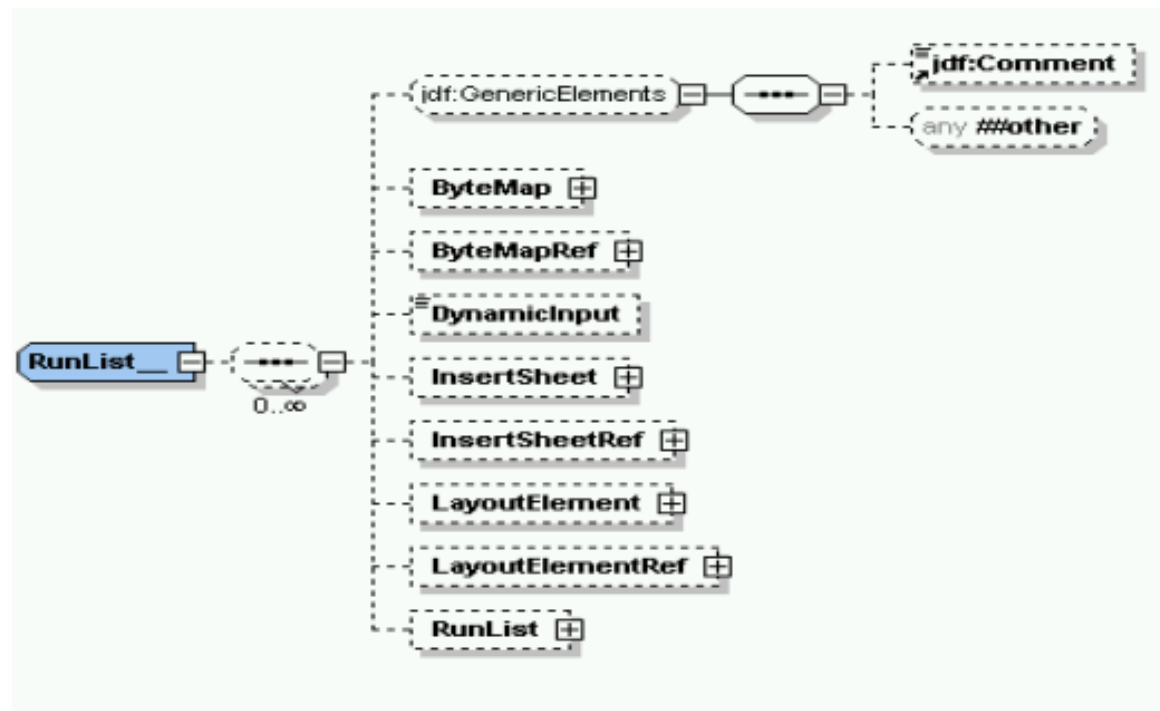
Activation sets the run mode for a whole subtree of nodes. The value of activation is inherited by child nodes.

JDF Resource



Resources contains elements of the following classes: Intent, Parameter, Placeholder, Implementation (Device/Employee), Handling, Quantity, Consumable. All Resource Elements are kept in a ResourceList Element.

Resource Example Runlist (1)



Resource Example Runlist(2)

```
<RunList Run="Run0126" NPage="1">
  <Comment>No Magenta, the missing sep does not exist as a page</Comment>
  <RunList SkipPage="3" FirstPage="10" Separation="Cyan">
    <LayoutElementRef rRef="Link0124"/>
  </RunList>
  <RunList SkipPage="3" FirstPage="11" Separation="Yellow">
    <LayoutElementRef rRef="Link0124"/>
  </RunList>
  <RunList SkipPage="3" FirstPage="12" Separation="Black">
    <LayoutElementRef rRef="Link0124"/>
  </RunList>
  <RunList SkipPage="3" FirstPage="13" Separation="Green">
    <LayoutElementRef rRef="Link0124"/>
  </RunList>
</RunList>
<LayoutElement ID="Link0124" Class="Parameter" Locked="false" Status="Available">
  <FileSpec URL="PreSepCMYKG.pdf" /></LayoutElement>
```

Runlist resources can form a tree with embedded runlist elements. Note that there is a resource internal reference to the LayoutElement with ID=„Link0124“

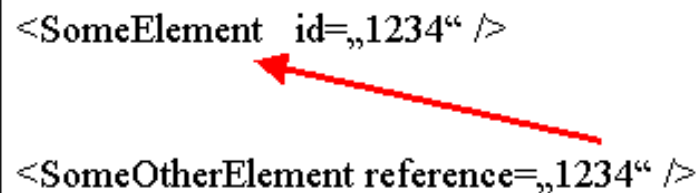
JDF Resource Instance Example

```
<?xml version='1.0' encoding='utf-8' ?>
<JDF ID="HDM21" Type="Product" JobID="HDM2002" Status="Waiting" Version="1.0">
  <ResourcePool>
    <SomeInputResource ID="Link0017" Class="Parameter" Locked="false" Status="Available"/>
    <Component ID="Link0018" Class="Quantity" Locked="false" Status="Unavailable"
      ComponentType="PartialProduct" DescriptiveName="SomeOutputResource"/>
  </ResourcePool>
  <ResourceLinkPool>
    <SomeInputResourceLink rRef="Link0017" Usage="Input"/>
    <ComponentLink rRef="Link0018" Usage="Output" />
  </ResourceLinkPool>
  <AuditPool>
    <Created Author="foo" TimeStamp="2002-04-24T17:21:26+02:00"/>
  </AuditPool>
  <Comment>
    This is a comment
  </Comment>
  <JDF ID="HDM22" ..... </JDF>
</JDF>
```

Every node has an unique ID and several other important attributes of which „status“ is very important.

Linking with xml: Link Attributes

„id“ must be of type „ID“, a special attribute type which is known by parsers. The name of the attribute does not matter, only the type ID is important.



The diagram shows two XML elements within a rectangular box. The top element is `<SomeElement id=„1234“ />` and the bottom element is `<SomeOtherElement reference=„1234“ />`. A red arrow originates from the value „1234“ in the `reference` attribute of the bottom element and points diagonally upwards to the value „1234“ in the `id` attribute of the top element, illustrating a reference link.

```
<SomeElement id=„1234“ />  
  
<SomeOtherElement reference=„1234“ />
```

„reference“ must be of type „IDREF“, a special attribute type which is known by parsers. The name of the attribute is not important, only the type

By using attributes of type ID and IDREF(S) one can refer to elements from WITHIN a document. The xml parser will ensure the following features:

- all id's must be unique within a document
- a referenced element must exist

It is the applications job to know what to do with those links or references. The parser checks only uniqueness. RDF uses this mechanism to link resource elements and resource element links.

Making IDs unique (e.g. merging of jobs)

<JMF id=„1“>.....

<JMF id=„1“>.....

composite ID = id_one.id_two.id_three etc.

The xml attribute type ID must be unique within a document. This means if independent jobs are merged that some IDs will have to be renamed. JDF does this by constructing a composite ID through connecting several pure IDs with the „.“ (dot) operator. In JDF nodes and resources use IDs a lot.

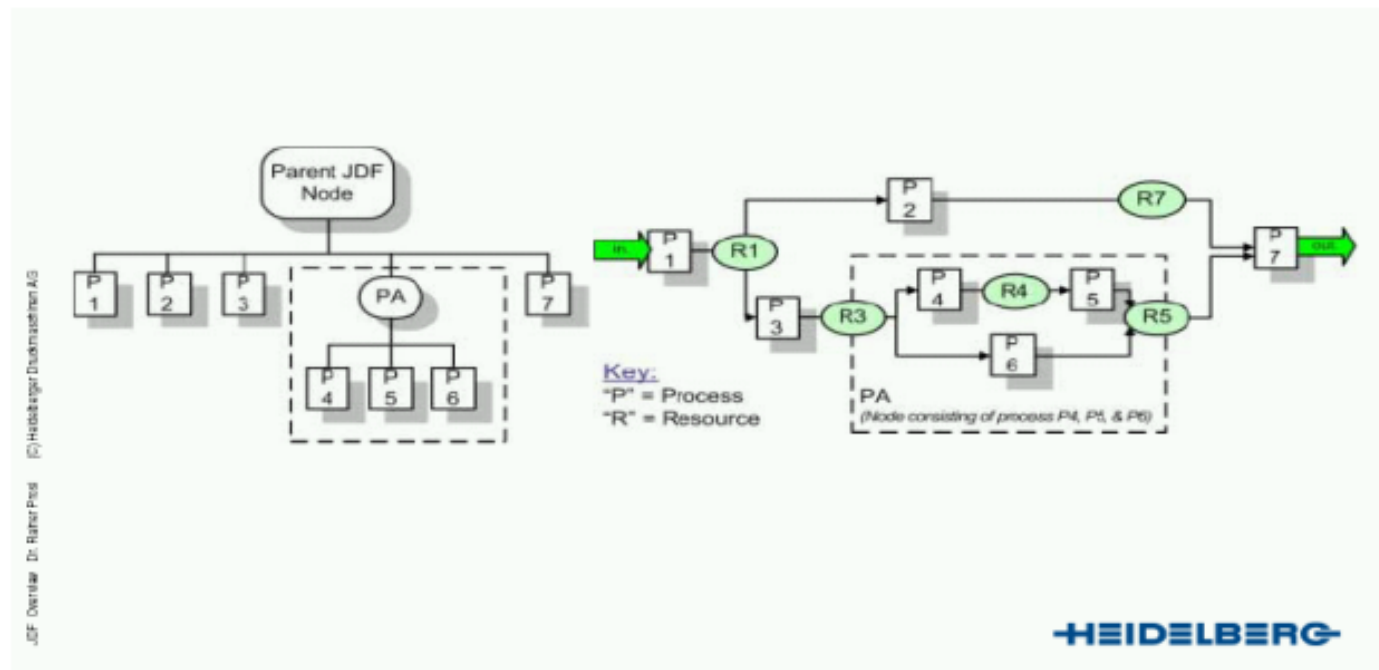
Application defined link elements

```
<Component ID="Link0010" Class="Quantity" Locked="false"  
Status="Unavailable" DescriptiveName="SomeOutputResource"/>  
  
<ComponentLink rRef="Link0010" Usage="Output"/>
```

Every JDF Resource has an associated link element which can be used to define input and output characteristics of that resource. Please note that the xml parser has no idea about the semantics of „component“ or „componentlink“. Both are just xml elements for the parser. The real workflow in a JDF driven system is defined by

- a) where resources are defined (the resourcepool section of a node)
- b) what resources are defined as input or output to this node.

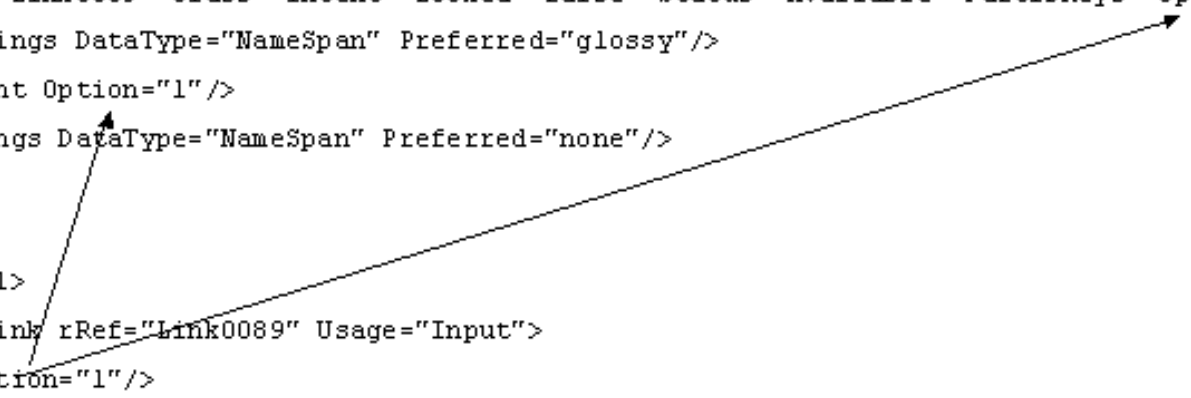
Resources control workflow



Please note that process nodes (4,5) and 6 can run in parallel because their resource definitions are independent.

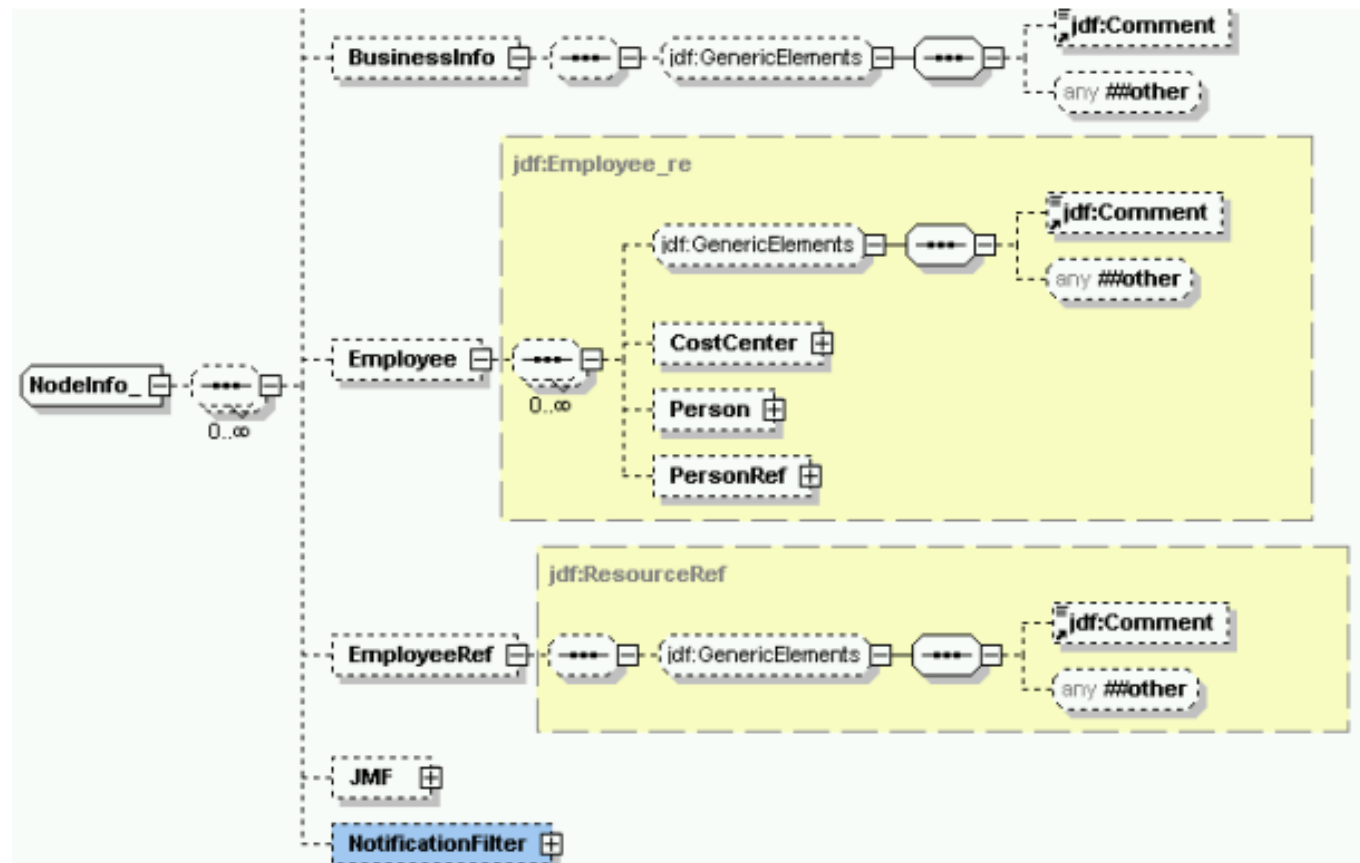
Partitioned Resources

```
<ResourcePool>
  <MediaIntent ID="Link0089" Class="Intent" Locked="false" Status="Available" PartIDKeys="Option">
    <FrontCoatings DataType="NameSpan" Preferred="glossy"/>
    <MediaIntent Option="1" />
    <BackCoatings DataType="NameSpan" Preferred="none"/>
  </MediaIntent>
</ResourcePool>
<ResourceLinkPool>
  <MediaIntentLink rRef="Link0089" Usage="Input">
    <Part Option="1" />
  </MediaIntentLink>
</ResourceLinkPool>
```



A link can use a Part element to select a specific part of a partitionable resource. Please note that the name „option“ is arbitrary. Any token will do as long as the resource uses it as the value of attribute „PartIDKeys“

NodeInfo Elements



Nodeinfo elements deal with job tracking. The embedded JMF element allows the node to establish a persistent notification channel to an MIS system.

NodeInfo Instance Example

```
<NodeInfo LastEnd="2001-08-26T07:14:01+02:00">  
  <BusinessInfo>  
    <RFQ Currency="DEM" LastQuote="2001-08-03T03:40:41+02:00" BusinessID="RFQ_ID"/>  
  </BusinessInfo>  
</NodeInfo>
```

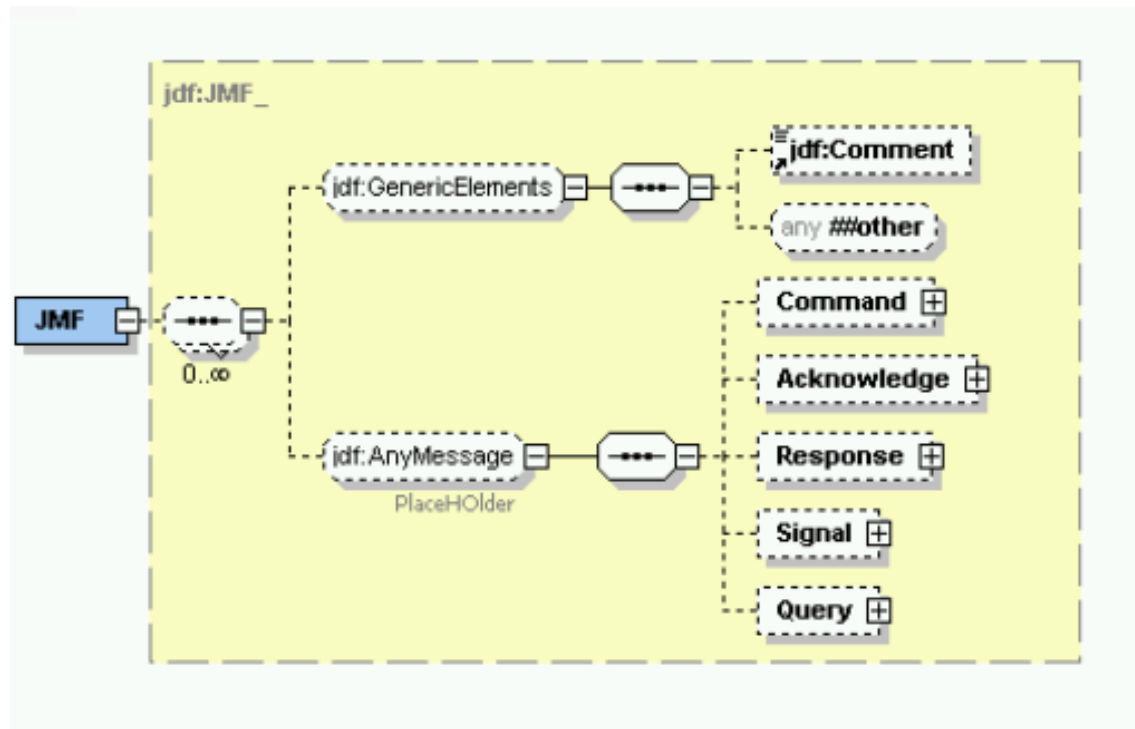
The attribute „LastEnd“ specifies the deadline for the node where this nodeinfo element is a child of. BusinessInfo is supposed to be used with e-commerce standards and serves as a container for business related information

JDF Example files

- testjdf12.jdf (Runlist and layout element resources)
- testjdf13.jdf (request for quote)
- testjdf16.jdf /testjdf18.jdf (query/response JMF)
- testjdf20.jdf (JMF Signal)
- testjdf31.jdf (proofing process node with extension elements)
- testjdf33.jdf (process group example)
- book.jdf (book job with partitioned jobs)

These are files from the C++ based development kit for JDF which can be downloaded from CIP4.org. You can find the jdf files in directory \JDF_CPP_API_B01\Projects\Win32\VC6\TestWrapper

JMF Node



The JMF node is the core element of the messaging system.

Main JMF Elements (1)

JMF Node:

The root element of the XML fragment that encodes a message.

Message:

This is the abstract element is the base type of concrete messages. It provides some attributes.

Query:

A family of messages which retrieve data from controllers without changing controller state. A query contains a placeholder for a descriptive element which further describes the query (QueryTypeObject)

Response:

A family of answer messages to queries. Contains notifications elements with further explanations. An ID attribute references the original query message. Long running queries might require a separate Acknowledge message.

Signal:

A unidirectional message sent to other controllers. Broadcast of events.

Acknowledge:

Response to a command.

JMF Query

```
<?xml version='1.0' encoding='utf-8' ?>  
<JMF SenderID="JMFCClient" TimeStamp="2001-07-27T17:21:28+02:00">  
  <Query ID="Q0177" Type="KnownMessages">  
    <KnownMsgQuParams ListQueries="true" ListSignals="false" ListCommands="true"/>  
  </Query>  
</JMF>
```

JMF Response

```
<?xml version='1.0' encoding='utf-8' ?>
<JMF SenderID="JMFCClient #2" TimeStamp="2001-07-27T17:21:28+02:00">
  <Response ID="R0181" Type="KnownMessages" refID="Q0177">
    <KnownMessages>
      <MessageService Type="KnownMessages" Query="true"/>
      <MessageService Type="Status" Query="true" Persistent="true"/>
      <MessageService Type="StopPersistentChannel" Command="true"/>
    </KnownMessages>
  </Response>
</JMF>
```

JMF Signal

```
<?xml version='1.0' encoding='utf-8' ?>  
<JMF SenderID="JMFCClient #2" TimeStamp="2001-07-27T17:21:28+02:00">  
  <Signal ID="S0190" Type="Status" refID="Q0182"/>  
</JMF>
```

There are 3 ways for a controller to receive a signal: A controller can subscribe for a signal by using a Query message through a message channel including a subscription element. Or through a NodeInfo element in a JDF element which contains a subscription. Or through a list of hardwired signal channels read from a file. The refID attribute refers to a persistent channel (not hardwired case)

JDF as an XML Schema

- Namespace
- Extensibility
- Types and Elements (example)
- Open questions

Extensibility

```
<xsd:complexType name="BaseElement_">
  <xsd:attribute name="CommentURL" type="jdf:URL" use="optional"/>
  <xsd:attribute name="DescriptiveName" type="xsd:string" use="optional"/>
  <xsd:anyAttribute namespace="##other"/>

```

```
<xsd:group name="GenericElements">
  <xsd:sequence>
    <xsd:element ref="jdf:Comment" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>

```

An industry schema needs to be extensible to allow vendors to support special machines or software. One way to do this with XML Schema is to allow arbitrary attributes or elements in certain places. This is done through „anyAttribute“ or „any“ elements. Namespace=„##other“ means to accept any wellformed xml from a different namespace and processContents=„lax“ tells the parser not to insist on finding a schema for those elements. Please note the inclusion of the „any“ features in base types of JDF so that other types and elements can inherit the extensibility by extending those base types.

JDF Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.CIP4.org/JDFSchema_1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jdf="http://www.CIP4.org/JDFSchema_1"
  xmlns:jdfP="http://www.CIP4.org/JDFSchema_1/JDFParser"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
```

.....

If elementFormDefault is „qualified“, local elements in an JDF instance document have to be qualified with a namespace prefix or the namespace must be defined through a default namespace declaration at the beginning. The jdf namespace itself is: "http://www.CIP4.org/JDFSchema_1"

Base Types and Base Elements (1)

```
<xsd:complexType name="BaseElement_">
  <xsd:attribute name="CommentURL" type="jdf:URL" use="optional"/>
  <xsd:attribute name="DescriptiveName" type="xsd:string" use="optional"/>
  <xsd:anyAttribute namespace="##other"/>
</xsd:complexType>
<xsd:complexType name="JDFBaseType_" abstract="true">
  <xsd:complexContent>
    <xsd:extension base="jdf:BaseElement_">
      <xsd:attribute name="Activation" type="jdf:eActivation_" use="optional"/>
      <xsd:attribute name="ID" type="xsd:ID" use="required"/>
      <xsd:attribute name="JobID" type="xsd:string" use="optional"/>
      <xsd:attribute name="JobPartID" type="xsd:string" use="optional"/>
      <xsd:attribute name="Status" type="jdf:eNodeStatus_" use="required"/>
      <xsd:attribute name="Types" type="xsd:NMTOKENS" use="optional"/>
      <xsd:attribute name="Version" type="xsd:string" default="1.0"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Please note how the complex type JDFBaseType_ extends BaseElement_ type and by doing so inherits the extensibility for attributes

Generic and Child Elements (2)

```
<xsd:group name="GenericElements">
  <xsd:sequence>
    <xsd:element ref="jdf:Comment" minOccurs="0"/>
    <xsd:any namespace="##other" processContents="lax" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>
<xsd:group name="JDFChildElements_">
  <xsd:sequence>
    <xsd:group ref="jdf:GenericElements" minOccurs="0"/>
    <xsd:element name="AncestorPool" type="jdf:AncestorPool_" minOccurs="0"/>
    <xsd:element name="AuditPool" type="jdf:AuditPool_" minOccurs="0"/>
    <xsd:element name="CustomerInfo" type="jdf:CustomerInfo_" minOccurs="0"/>
    <xsd:element name="NodeInfo" type="jdf:NodeInfo_" minOccurs="0"/>
    <xsd:element name="ResourcePool" type="jdf:ResourcePool_" minOccurs="0"/>
    <xsd:element name="StatusPool" type="jdf:StatusPool_" minOccurs="0"/>
    <xsd:element ref="jdf:JDF" minOccurs="0"/>
  </xsd:sequence>
</xsd:group>
```

The **GenericElements** uses a **Comment** element (not shown) and allows any other elements. **JDFChildElements** includes **GenericElements** thereby inheriting the extensibility. Additionally the main JDF node child elements are defined.

Node Types and Element (3)

```
<xsd:complexType name="JDFGenericProcess">
  <xsd:complexContent>
    <xsd:extension base="jdf:JDFBaseType_">
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:group ref="jdf:JDFChildElements_" minOccurs="0"/>
        <xsd:element name="ResourceLinkPool" type="jdf:GenericResourceLinkPool_"
minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="Type" type="xsd:NMTOKEN" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="JDF" type="jdf:JDFGenericProcess">
  <xsd:annotation>
    <xsd:appinfo>
      <Constraint type="jdf:JDFBaseType_" />
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

Finally the main JDF node is defined by reference to the JDFGenericProcess type which in turn extends JDFBaseType_ and so on...

Open Questions

- What is the purpose of all those types ending in
__, _r, _e, _re, _rp ???

The JDF Developer mailing list might be able to answer those questions. Please subscribe by sending email to listserv@igd.fhg.de. Then complete the email body by replacing <your full name> with your name, e.g.

"subscribe JDFDEV Hans Mueller".

JDF Resources

- www.cip4.org is the main portal for jdf information
- JDF Specification Version 1.0 (December 2001)
http://www.cip4.org/documents/jdf_specifications/JDFSpec1.pdf
- jdf schema in XML-Schema language http://www.cip4.org/Schema/JDF_1.xsd
- very good overview from IPEX 2002: jdf technology overview
http://www.cip4.org/documents/jdf_overview/cip4_seminars_ipex2002/JDF_Technology.pdf
- Graham Mann, XML Schema for Job Definition Format

This lesson assumes a general familiarity with JDF (basically what it is and what the print industry wants to achieve through its use).