



CIP4 xJdfLib

Version: 0.4

Date: 2013-02-25 17:37



Abstract

This document describes the structure and functionality of CIP4 xJdfLib and how to use it. The library is written in Java and manages the dealing with XJDF Documents and JMF Messages. It provides methods for creation, modification and analyzing.

Developer Team

CIP4 thanks the developer team of 'CIP4 xJdfLib' and their organizations for the great work!

Dr. Rainer Prosi

Heidelberger Druckmaschinen AG

<http://www.heidelberg.com>

Stefan Meissner

flyeralarm GmbH

<http://www.flyeralarm.com>

Legal Notice

Use of this document is subject to the following conditions which are deemed accepted by any person or entity making use hereof.

Copyright Notice

Copyright © 2000-2012, International Cooperation for the Integration of Processes in Prepress, Press and Postpress (CIP4) with registered office in Zurich, Switzerland. All Rights Reserved. CIP4 hereby grants to any person or entity obtaining a copy of the Specification and associated documentation files (the "Specification") a perpetual, worldwide, non-exclusive, fully paid-up, royalty-free copyright license to use, copy, publish, distribute, publicly display, publicly perform, and/or sublicense the Specification in whole or in part verbatim and without modification, unless otherwise expressly permitted by CIP4, subject to the following conditions. This legal notice must be included in all copies containing the whole or substantial portions of the Specification. Copies of excerpts of the Specification which do not exceed five (5) pages must include the following short form Copyright Notice: Copyright © 2000-2008, International Cooperation for the Integration of Processes in Prepress, Press and Postpress (CIP4) with registered office in Zurich, Switzerland.

Trademarks and Tradenames

International Cooperation for the Integration of Processes in Prepress, Press and Postpress, CIP4, Job Definition Format, JDF, Job Messaging Format, JMF, and the CIP4 logo are trademarks of CIP4. Rather than put a trademark symbol in every occurrence of other trademarked names, we state that we are using the names only in an editorial fashion, and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Except as contained in this legal notice or as allowed by membership in CIP4, the name of CIP4 must not be used in advertising or otherwise to promote the use or other dealings in this Specification without prior written authorization from CIP4.

Waiver of Liability

The JDF Specification is provided as is, without warranty of any kind, express, implied, or otherwise, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event will CIP4 be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of, or in connection with the JDF Specification or the use or other dealings in the JDF Specification.

CIP4 THANKS ITS PARTNER LEVEL MEMBERS



Table of Contents

1. Introduction	1
2. The xJdfLib Java Library	3
3. Components Description	4
3.1. XJDF Data Types	4
3.1.1. DateTime	4
3.1.2. Duration	5
3.1.3. IntegerList	6
3.1.4. Matrix	7
3.1.5. NMTokens	8
3.1.6. Rectangle	8
3.1.7. Shape	9
3.1.8. XYPair	10
3.2. XJdfNodeFactory	11
3.3. Builder Classes	12
3.3.1. XJdfBuilder	14
3.3.1.1. Partitioning of Parameter nodes	15
3.3.2. ProductBuilder	16
3.3.3. ContactBuilder	17
3.3.4. LayoutBuilder	18
3.4. XJdfParser	20
3.5. XJdfValidator	22
3.6. XJdfNavigator	23
3.6.1. XPath Expressions	23
3.6.2. Read and Modify Attributes	26
3.6.3. Extended XPath Functionality	27
3.7. XJdfPackager	29
3.8. XJdfConstants	30

3.9. Util Classes	31
3.10. Performance Optimizations	31
4. Developer Infos	33
4.1. Update XJDF Schema	33
5. Conclusion	34

List of Tables

3.1. XPath expressions	23
3.2. Java Constants of XPath Expressions	24
3.3. Overview XPathConstants	28
3.4. Overview xJdfLib Constants	30
3.5. Overview Util classes	31

Chapter 1. Introduction

The major conceptual change in JDF 2.0 (or XJDF) is that the technology no longer attempts to model the entire job as one large "job ticket" but rather specifies an interchange format between two applications that are assumed to have an internal data model that is not necessarily based on JDF. Thus each JDF ticket specifies a single transaction between two parties. A single job may be modeled as one or more JDF transactions.

This fact also requires a conceptual change of the JDF libraries as well. Among other things the focus of 1.x libraries is dealing with large XML files (XML files were used as data storage), managing concurrency file access (spawn and merge) and handling the workflow logic (references). Also there are implemented very common XML functionalities like creating Java objects from XSD Schema etc.

The fundamental concepts of JDF (and the library) were designed in the nineties. At this time XML was a very new technology and just a few early XML Tools had existed. Further, the XML technology itself was in a very early phase of development. Therefore, it was essential to include all the implementation specific details mentioned into the JDF Specification to get it working.

Over the last two decades XML has become very popular. Many tools have been designed and many conceptual enhancements in XML have been done. For example, one significant enhancement is XPath. More details about the XPath W3C Standard later in this document or on the W3C website: <http://www.w3.org/TR/xpath/>.

Due to the major change from JDF 1.x to JDF 2.0 a redesign of the JDF libraries is strongly recommended. The new CIP4 XJDF Library ("xJdfLib") is based on commonly used libraries and technologies like the Apache or JAXB framework. Supplementary functionality which has been removed from JDF Specification no longer makes the library unnecessarily complex for reasons of backwards compatibility. The goal of xJdfLib is to provide a lightweight,

modern and easy to use library optimized for actual requirements. This document describes the concepts and the usage of the CIP4 xJdfLib.

Chapter 2. The xJdfLib Java Library

The CIP4 xJdfLib Java Library is based on Java. The latest stable version always is being published on the CIP4 Website on page “Technical Resources -> Downloads -> Internal Source”. Furthermore, the library is an Apache Maven project. So the latest stable version also is available in the public Central Maven Repository:

```
<dependency>
  <groupId>org.cip4.lib.xjdf</groupId>
  <artifactId>xJdfLib</artifactId>
  <version>0.4</version>
</dependency>
```

Early Next-Version-Snapshots of the CIP4 xJdfLib Library are being published in the public OSS Sonatype Snapshot Repository:

```
<repository>
  <id>SnapshotOSS</id>
  <name>OSS Snapshot</name>
  <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
```

```
<dependency>
  <groupId>org.cip4.lib.xjdf</groupId>
  <artifactId>xJdfLib</artifactId>
  <version>0.5-SNAPSHOT</version>
</dependency>
```

Chapter 3. Components Description

3.1. XJDF Data Types

The XJDF Specification specifies several data types. At this time not all but the most significant data types already are implemented in CIP4 XJDF Java Library. The next versions of xJdfLib will complete the list step by step.

All XJDF Type classes are derived from `AbstractXJdfType` and provide a custom `toString()` method for converting a data type to a string expression. The way around is covered by a custom constructor. Each XJDF Data Type provides at least a default and several custom constructor for initializing.

3.1.1. DateTime

A `DateTime` object 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.

The XJDF `DateTime` object provides different constructors for initializing. The default constructor creates the current date as XJDF `DateTime` object whereas the custom constructors initialize the object by a custom value.

```
// current time
DateTime d1 = new DateTime();
System.out.println(d1.toString()); // 2013-02-24T18:34:01+01:00

// specify date only
DateTime d2 = new DateTime(2013, 02, 24);
System.out.println(d2.toString()); // 2013-02-24T23:59:00+01:00

// specify date and time
DateTime d3 = new DateTime(2013, 02, 24, 15, 30);
System.out.println(d3.toString()); // 2013-02-24T15:30:00+01:00

// specify date as string
DateTime d4 = new DateTime("2013-02-24T18:30:00+00:00");
System.out.println(d4.toString()); // 2013-02-24T18:30:00Z
```

```
// initialize DateTime from Calendar
DateTime d5 = new DateTime(new GregorianCalendar());
System.out.println(d5.toString()); // 2013-02-24T18:42:46+01:00
```

Internally the XJDF DateTime object holds the information as Calendar. This object can also be accessed using the Getter-Method:

```
// read DateTime
DateTime dateTime = new DateTime();
dateTime.getCalendar();
```

3.1.2. Duration

Durations are a component of time intervals and define the amount of intervening time in a time interval. Durations are represented by the format P[n]Y[n]M[n]DT[n]H[n]M[n]S.

The XJDF Duration object provides several constructors for initializing. The default constructor creates an empty XJDF Duration object whereas the custom constructors initialize the object by a custom value.

```
// empty duration
Duration d1 = new Duration();
System.out.println(d1.toString()); // [no output]

// two days
Duration d2 = new Duration(2);
System.out.println(d2.toString()); // P2D

// two days and twelve hours
Duration d3 = new Duration(2, 12);
System.out.println(d3.toString()); // P2DT12H

// full initialization
Duration d4 = new Duration(1, 2, 3, 4, 5, 6);
System.out.println(d4.toString()); // P1Y2M3DT4H5M6S

// string expression
Duration d5 = new Duration("P3DT4H5M");
System.out.println(d5.toString()); // P3DT4H5M
```

All duration values can easily be accessed using the Getter-Methods:

```
// read Duration
```

```
Duration duration = new Duration(0, 6);
int year = duration.getYear();
int month = duration.getMonth();
int day = duration.getDay();
int hour = duration.getHour(); // 6
int minute = duration.getMinute();
int second = duration.getSecond();
```

3.1.3. IntegerList

An IntegerList is an enumerated set of Integers, which is expressed as a list of space separated values.

The XJDF IntegerList object provides several constructors for initializing. The default constructor creates an empty XJDF IntegerList object whereas the custom constructors initialize the object by a custom value. That constructor which accepts a variable number of integer values is the most preferred one for creating IntegerList objects.

```
// empty list
IntegerList l1 = new IntegerList();
System.out.println(l1.toString()); // [no output]

// integer list with 2 elements
IntegerList l2 = new IntegerList(4, 4);
System.out.println(l2.toString()); // 4 4

// integer list with 4 elements
IntegerList l3 = new IntegerList(1, 2, 3, 4);
System.out.println(l3.toString()); // 1 2 3 4

// string expresssion
IntegerList l4 = new IntegerList("2 2");
System.out.println(l4.toString()); // 2 2
```

Internally the XJDF IntegerList data type holds all items as a List of Integers. This list can be addressed using the Getter-Method:

```
// read IntegerList
IntegerList integerList = new IntegerList(4, 0);
List<Integer> lst = integerList.getList();
```

3.1.4. Matrix

Coordinate transformation matrices are widely used throughout the whole printing Process, especially in Layout Resources. They represent two dimensional transformations as defined by [PS] and [PDF1.6]. For more information, refer to the respective reference manuals, and look for "Coordinate Systems and Transformations." The "identity matrix", which is "1 0 0 1 0 0", is often used as a default throughout this specification. When another matrix is factored against a matrix with the identity matrix value, the result is that the original matrix remains unchanged. Coordinate transformation matrices are primitive data types and are encoded as a list of six numbers (as doubles), separated by whitespace: "a b c d Tx Ty". The variables Tx and Ty describe distances and are defined in points.

The XJDF Matrix object provides several constructors for initializing. The default constructor creates a default XJDF Matrix object ("1 0 0 1 0 0") whereas the custom constructors initialize the object by a custom value.

```
// default matrix
Matrix m1 = new Matrix();
System.out.println(m1.toString()); // 1.0 0.0 0.0 1.0 0.0 0.0

// custom matrix
Matrix m2 = new Matrix(1, 2, 3, 4, 5, 6);
System.out.println(m2.toString()); // 1.0 2.0 3.0 4.0 5.0 6.0

// string expression
Matrix m3 = new Matrix("2 2 4 4 0 0");
System.out.println(m3.toString()); // 2.0 2.0 4.0 4.0 0.0 0.0
```

In case a detailed access of all elements is required, the XJDF Matrix class provides several Getter-Methods to achieve this.

```
// read matrix
Matrix matrix = new Matrix(1, 2, 3, 4, 5, 6);
double a = matrix.getA(); // 1.0
double b = matrix.getB(); // 2.0
double c = matrix.getC(); // 3.0
double d = matrix.getD(); // 4.0
double tx = matrix.getTx(); // 5.0
double ty = matrix.getTy(); // 6.0
```

3.1.5. NMTokens

NMTOKENS is an enumerated set of NMTOKEN, which is expressed as a list of space separated values.

The XJDF NMTokens object provides several constructors for initializing. The default constructor creates an empty XJDF NMTokens object whereas the custom constructors initialize the object by custom values.

```
// empty object
NMTokens n1 = new NMTokens();
System.out.println(n1.toString()); // [no output]

// multiple string expression
NMTokens n2 = new NMTokens("Val_1", "Val_2", "Val_3");
System.out.println(n2.toString()); // Val_1 Val_2 Val_3

// single string expression
NMTokens n3 = new NMTokens("Val_1 Val_2 Val_3");
System.out.println(n3.toString()); // Val_1 Val_2 Val_3
```

Internally the XJDF NMTokens data type holds all details as a List of Strings. This list can be addressed using the Getter-Method:

```
// read NMTokens
NMTokens nmTokens = new NMTokens("Val_1 Val_2 Val_3");
List<String> lst = nmTokens.getList();
```

3.1.6. Rectangle

Rectangles 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, lowerleft 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.

The XJDF Rectangle object provides several constructors for initializing. The default constructor creates an empty XJDF Rectangle object whereas the custom constructors initialize the object by custom values.

```
// empty rectangle
Rectangle r1 = new Rectangle();
System.out.println(r1.toString()); // 0.0 0.0 0.0 0.0

// customized rectangle
Rectangle r2 = new Rectangle(1, 2, 3, 4);
System.out.println(r2.toString()); // 1.0 2.0 3.0 4.0

Rectangle r3 = new Rectangle("2 4 6 8");
System.out.println(r3.toString()); // 2.0 4.0 6.0 8.0
```

All elements of the rectangle can be read using of the Getter-Methods:

```
// read Rectangle
Rectangle rectangle = new Rectangle(1, 2, 3, 4);
double llx = rectangle.getLlx();
double lly = rectangle.getLly();
double urx = rectangle.getUrx();
double ury = rectangle.getUry();
```

3.1.7. Shape

Shape data types are used to describe a three dimensional box. A shape is represented as an array of three (positive or zero) numbers x y z specifying the Width x, height y and depth z coordinates of the shape, in that order.

The XJDF Shape object provides several constructors for initializing. The default constructor creates an empty XJDF Shape object whereas the custom constructors initialize the object by custom values.

```
// empty shape
Shape s1 = new Shape();
System.out.println(s1.toString()); // 0.0 0.0 0.0

// definition of x and y only
Shape s2 = new Shape(4.5, 7.8);
System.out.println(s2.toString()); // 4.5 7.8 0.0

// definition of all elements
Shape s3 = new Shape(6.4, 5.7, 9.9);
System.out.println(s3.toString()); // 6.4 5.7 9.9

// string expression
```

```
Shape s4 = new Shape("3 5.6 8.0");  
System.out.println(s4.toString()); // 3.0 5.6 8.0
```

All elements of the shape can be read using of the Getter-Methods:

```
// read Shape  
Shape shape = new Shape(10.5, 15.2);  
double x = shape.getX(); // 10.5  
double y = shape.getY(); // 15.2  
double z = shape.getZ(); // 0.0
```

3.1.8. XYPair

XYPairs are used to describe sizes like Dimensions and StartPosition. They can also be used to describe positions on a page. All numbers that describe lengths are defined in points. XYPair Attributes are primitive data types and are encoded as a string of two numbers, separated by whitespace: "x y".

The XJDF XYPair object provides several constructors for initializing. The default constructor creates an empty XJDF XYPair object whereas the custom constructors initialize the object by custom values.

```
// empty object  
XYPair p1 = new XYPair();  
System.out.println(p1.toString()); // 0.0 0.0  
  
// definition of x and y  
XYPair p2 = new XYPair(4.4, 5.9);  
System.out.println(p2.toString()); // 4.4 5.9  
  
// string expression  
XYPair p3 = new XYPair("3 8.9");  
System.out.println(p3.toString()); // 3.0 8.9
```

All elements of the XYPair can be read using of the Getter-Methods:

```
// read XYPair  
XYPair xyPair = new XYPair(4.5, 5.0);  
double x = xyPair.getX();  
double y = xyPair.getY();
```

3.2. XJdfNodeFactory

The XJdfNodeFactory is the factory class for creating new instances of XJDF-Node-Objects. The class provides at least one simple Creation-Method per XJDF Node defined in XJDF Specification. Moreover, the class also provides extended Creation-Methods for commonly used nodes (e. g. GeneralID, RunList etc.) which also initializes the object after creation.

```
// new factory instance  
XJdfNodeFactory nf = new XJdfNodeFactory();
```

Following a demonstration about the difference between simple and extended Creation-Methods. For example, when working with GeneralIDs most of them consist only of the attributes "IDUsage" and "IDValue". Using the simple method, the creation of such a GeneralID Node would require three lines of code. The first line creates the new Node-Object whereas the next two lines initialize the object. Each attribute explicitly is set by a setter call:

```
// new factory instance  
XJdfNodeFactory nf = new XJdfNodeFactory();  
  
// New GeneralID XJDF Node Object using the simple method  
GeneralID generalId = nf.createGeneralID();  
generalId.setIDUsage("IDCatalog");  
generalId.setIDValue("42");
```

How ever, the creation of XJDF Documents only with the usage of simple Creation-Methods probably will consumes time and would raise code complexity. Extended Creation-Methods provides a more straighter way. Using these methods commonly used XJDF Nodes are able to be created and initialized by a single line of code:

```
// new factory instance  
XJdfNodeFactory nf = new XJdfNodeFactory();  
  
// New GeneralID XJDF Node Object using the extended method  
GeneralID generalId = nf.createGeneralID("IDCatalog", "42");
```

Both methods creates one and the same GeneralID Node. Simple methods raise the flexibility of attribute management whereas the extended ones decrease complexity, maintenance and produce a more clearly source code.

It is recommended to use the extended Creation-Methods whenever there is one available. If the extended method does not fit the needs, the preinitialized node object always can be modified in a further line of code. For instance, in some situations the */RunList/FileSpecNode* may contain an additional attribute *UserFileName* which is not supported by an extended method:

```
[...]
<xjdf:RunList>
  <xjdf:FileSpec URL="http://192.168.1.113:80/10496"
    UserFileName="myFileName.pdf"/>
</xjdf:RunList>
[...]
```

Here a demonstration how to realize such a modification in a further step:

```
// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

// best practice creating indivudal nodes
RunList runList = nf.createRunList("http://192.168.1.113:80/10496");
runList.getFileSpec().setUserFileName("myFileName.pdf");
```

3.3. Builder Classes

Most of the XJDF-Node-Objects can easily be created using the XJdfNodeFactory as described in the chapter before. How ever, more complex nodes like the XJDF-Root-Node or the Product-Node are nodes which contain a set of subnodes in a well defined structure. This requires additional logic to organize all child nodes within the parent node. Builder classes are designed to achieve this.

The XJDF-Root-Node for example is a parent node for all Product- and Parameter-Nodes. All *Product* items are listed under subelement *ProductList* whereas all *Parameter* nodes are listed in specific *ParameterSet* elements. So, when adding a new Parameter-Node the builder

class automatically checks whether the right *ParameterSet* element already exists. If not, the library creates a new one and finally puts the new parameter to the right position.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSchema_2_0" ID="XJDF_CCC45NIM"
  DescriptiveName="My lovely Poster" JobID="FA-SIG-123456"
  Category="Web2Print" Version="2.0">
  <xjdf:ProductList>
    <xjdf:Product DescriptiveName="FA-PRD-123456" Amount="1500"
      ProductType="Poster" ProductTypeDetails="PTD Value">
      <xjdf:Intent Name="MediaIntent">
        <xjdf:MediaIntent MediaQuality="IPM_90"/>
      </xjdf:Intent>
    </xjdf:Product>
  </xjdf:ProductList>
  <xjdf:ParameterSet Name="RunList">
    <xjdf:Parameter>
      <xjdf:RunList>
        <xjdf:FileSpec URL="test_file.pdf"/>
      </xjdf:RunList>
    </xjdf:Parameter>
  </xjdf:ParameterSet>
  <xjdf:ParameterSet Name="Contact">
    <xjdf:Parameter>
      <xjdf:Contact>
        <xjdf:Person FamilyName="Meissner" FirstName="Stefan"/>
        <xjdf:Company OrganizationName="flyeralarm GmbH"/>
        <xjdf:ComChannel ChannelType="Phone"
          Locator="tel:+49.931.465840"/>
        <xjdf:Address PostalCode="97082" City="Wuerzburg"
          Street="Alfred-Nobel-Strasse 15"/>
      </xjdf:Contact>
    </xjdf:Parameter>
  </xjdf:ParameterSet>
</xjdf:XJDF>
```

Following the associated Java code snippet which shows how to create such an XJDF Document as shown above:

```
// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

// create XJDF Document
ProductBuilder productBuilder = new ProductBuilder(1500, "Poster",
  "PTD Value", "FA-PRD-123456");
```

```
productBuilder.addIntent(nf.createMediaIntent("IPM_90"));
Product product = productBuilder.build();

ContactBuilder contactBuilder = new ContactBuilder();
contactBuilder.addCompany("flyeralarm GmbH");
contactBuilder.addPerson("Meissner", "Stefan", null);
contactBuilder.addAddress("Alfred-Nobel-Strasse 15", "97082", "Wuerzburg");
contactBuilder.addComChannel("Phone", "tel:+49.931.465840");
Contact contact = contactBuilder.build();

XJdfBuilder xJdfBuilder = new XJdfBuilder("FA-SIG-123456", "Web2Print",
    "My lovely Poster");
xJdfBuilder.addParameter(nf.createRunList("test_file.pdf"));
xJdfBuilder.addParameter(contact);
xJdfBuilder.addProduct(product);
XJDF xjdf = xJdfBuilder.build();

// parse document
byte[] bytes = new XJdfParser().parseXJdf(xjdf);

// output
System.out.println(new String(bytes));
```

The current version of CIP4 XJdfLib provides the following builder classes:

- **XJdfBuilder**

Creation of XJDF Documents. Manages the dealing with Products and Parameters.

- **ProductBuilder**

Creation of Product-Nodes. Handles all Intent nodes.

- **ContactBuilder**

Creation of Contact-Nodes. Organize the handling with contact details.

- **LayoutBuilder**

Creation of Layout-Nodes. Manages the handling with Layout elements.

3.3.1. XJdfBuilder

The XJdfBuilder is responsible for the creation and management of XJDF-Root-Nodes as well as the main structure in XJDF Documents. All child nodes can easily be appended by calling the associated "add-" methods. The logic where exactly a specific node has to be put is covered by the builder class.

For instance the XJDF Specification defines that all Parameter-Nodes have to be embedded within a specific ParameterSet element. This mechanism is fully supported by the builder class. So when adding a new Parameter item, the `addParameter()` method checks if the right ParameterSet-Node already exists. If not, a new one is created automatically. The following is a Java code snippet of how to use the XJdfBuilder.

```
// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

Contact contact = [...];
Product product = [...];

// create XJDF with builder
XJdfBuilder xJdfBuilder = XJdfBuilder.newInstance("FA-SIG-123456");
xJdfBuilder.addParameter(nf.createRunList("test_file.pdf"));
xJdfBuilder.addParameter(contact);
xJdfBuilder.addProduct(product);
XJDF xjdf = xJdfBuilder.build();

// build XJDF Doc
XJDF xJdf = xJdfBuilder.build();
```

3.3.1.1. Partitioning of Parameter nodes

Following a sample of a XJDF Document with a partitioned RunList. The XJDF Document references the PDF files for cover and body separately. This mechanism is called partitioning and requires an Part-Node per item.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSchema_2_0" ID="XJDF_2GYLJJJG"
  JobID="FA-SIG-123456" Category="Web2Print" Version="2.0">
  <xjdf:ProductList>
    <xjdf:Product Amount="1500"/>
  </xjdf:ProductList>
  <xjdf:ParameterSet Name="RunList">
    <xjdf:Parameter>
      <xjdf:RunList>
        <xjdf:FileSpec URL="cover.pdf"/>
      </xjdf:RunList>
      <xjdf:Part Run="cover"/>
    </xjdf:Parameter>
    <xjdf:Parameter>
```

```
<xjdf:RunList>
  <xjdf:FileSpec URL="body.pdf"/>
</xjdf:RunList>
<xjdf:Part Run="body"/>
</xjdf:Parameter>
</xjdf:ParameterSet>
</xjdf:XJDF>
```

Here a demonstration of how to create such partitioned XJDF Documents using the CIP4 XJDF Library. The XJdfBuilder class provides a further override of the addParameter() method to handle partitioning:

```
// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

// create XJDF Document
ProductBuilder productBuilder = new ProductBuilder(1500);
Product product = productBuilder.build();

Part partCover = nf.createPart();
partCover.setRun("cover");

Part partBody = nf.createPart();
partBody.setRun("body");

XJdfBuilder xJdfBuilder = new XJdfBuilder("FA-SIG-123456", "Web2Print");
xJdfBuilder.addParameter(nf.createRunList("cover.pdf"), partCover);
xJdfBuilder.addParameter(nf.createRunList("body.pdf"), partBody);
xJdfBuilder.addProduct(product);
XJDF xjdf = xJdfBuilder.build();
```

3.3.2. ProductBuilder

The Product-Node is another signification element in an XJDF Document. This node specifies the product configuration how desired by the customer. In XJDF, most product configurations are defined as Intent-Nodes. A Product-Node consists of at least itself, its attributes and a set of Intent-Node subelements. The ProductBuilder organizes the creation of a Product-Node as well as the handling of all its Intent-Nodes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSchemas_2_0" ID="XJDF_4Q1KC60M"
  JobID="FA-SIG-123456" Category="Web2Print" Version="2.0">
```

```
<xjdf:ProductList>
  <xjdf:Product Amount="1500">
    <xjdf:Intent Name="MediaIntent">
      <xjdf:MediaIntent MediaQuality="IPG_90"/>
    </xjdf:Intent>
    <xjdf:Intent Name="LayoutIntent">
      <xjdf:LayoutIntent Sides="TwoSidedHeadToHead" Pages="2"
        FinishedDimensions="297.63779528 419.52755906 0.0"/>
    </xjdf:Intent>
    <xjdf:Intent Name="ColorIntent">
      <xjdf:ColorIntent NumColors="4 4"/>
    </xjdf:Intent>
  </xjdf:Product>
</xjdf:ProductList>
</xjdf:XJDF>
```

The following the associated Java code snippet for the creation of such a Product-Node:

```
// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

// create product node
ProductBuilder productBuilder = new ProductBuilder(1500);
productBuilder.addIntent(nf.createMediaIntent("IPG_90"));
productBuilder.addIntent(nf.createLayoutIntent(2, "TwoSidedHeadToHead",
    new Shape(297.63779528, 419.52755906)));
productBuilder.addIntent(nf.createColorIntent(new IntegerList(4, 4)));
Product product = productBuilder.build();

XJdfBuilder xJdfBuilder = new XJdfBuilder("FA-SIG-123456", "Web2Print");
xJdfBuilder.addProduct(product);
XJDF xjdf = xJdfBuilder.build();
```

3.3.3. ContactBuilder

The ContactBuilder simplify the creation of Contact-Nodes. Contact-Nodes hold all the customers contact and delivery details and can be added as parameter to an XJDF Document. Usually, a contact parameter consists of the Contact-Node with at least an Address-Node, a Company-Node and a ComChannel-Node as subelements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSschema_2_0" ID="XJDF_C7C2Q04K"
  JobID="FA-SIG-123456" Category="Web2Print" Version="2.0">
  <xjdf:ParameterSet Name="Contact">
```

```
<xjdf:Parameter>
  <xjdf:Contact ContactTypes="Delivery Customer">
    <xjdf:Person FamilyName="Meissner" FirstName="Stefan"/>
    <xjdf:Company OrganizationName="flyeralarm GmbH"/>
    <xjdf:ComChannel ChannelType="Phone" Locator="tel:+49.931.465840"/>
    <xjdf:Address PostalCode="97082" City="Wuerzburg"
      Street="Alfred-Nobel-Strasse 15"/>
  </xjdf:Contact>
</xjdf:Parameter>
</xjdf:ParameterSet>
</xjdf:XJDF>
```

Using the ContactBuilder, all subnodes easily can be created and added by simple method calls:

```
// create contact node
ContactBuilder contactBuilder = new ContactBuilder();
contactBuilder.addCompany("flyeralarm GmbH");
contactBuilder.addPerson("Meissner", "Stefan", null);
contactBuilder.addAddress("Alfred-Nobel-Strasse 15", "97082", "Wuerzburg");
contactBuilder.addComChannel("Phone", "tel:+49.931.465840");
contactBuilder.addContactType("Delivery");
contactBuilder.addContactType("Customer");
Contact contact = contactBuilder.build();

XJdfBuilder xJdfBuilder = new XJdfBuilder("FA-SIG-123456", "Web2Print");
xJdfBuilder.addParameter(contact);
XJDF xjdf = xJdfBuilder.build();
```

3.3.4. LayoutBuilder

The Layout-Node contains imposition and stripping details. In many cases this node becomes complex very quickly. Thus, the CIP4 xJdfLib provides a LayoutBuilder class for simplification.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSchemas_2_0" ID="XJDF_BQS241K4"
  JobID="FA-SIG-123456" Category="Web2Print" Version="2.0">
  <xjdf:ParameterSet Name="Layout">
    <xjdf:Parameter>
      <xjdf:Layout SurfaceContentsBox="0.0 0.0 2976.3779 2125.9842">
        <xjdf:MarkObject ClipBox="0.0 0.0 2976.3779 2125.9842" Ord="0"
          CTM="1.0 0.0 0.0 1.0 0.0 0.0"/>
        <xjdf:ContentObject ClipBox="1731.9685 85.0393 2939.5275 374.1732"
```

```

        TrimCTM="0.0 -1.0 1.0 0.0 1734.8031 371.3385" Ord="0"
        TrimSize="813.5433 425.1968"
        CTM="0.0 -1.0 1.0 0.0 1731.968 374.1712"/>
<xjdf:ContentObject ClipBox="1731.9685 374.1732 2939.5275 663.307"
        TrimCTM="0.0 -1.0 1.0 0.0 1734.8031 660.4724" Ord="1"
        TrimSize="283.4645 1201.8897"
        CTM="0.0 -1.0 1.0 0.0 1731.968 663.3051"/>
<xjdf:MarkObject ClipBox="0.0 0.0 2976.3779 2125.9842" Ord="1"
        CTM="1.0 0.0 0.0 1.0 0.0 0.0"/>
    </xjdf:Layout>
</xjdf:Parameter>
</xjdf:ParameterSet>
</xjdf:XJDF>

```

Here, the associated Java code snippet how to create such a Layout-Node using the LayoutBuilder class:

```

Matrix ctm;
Rectangle clipBox;
Matrix trimCtm;
XYPair trimSize;

// new factory instance
XJdfNodeFactory nf = new XJdfNodeFactory();

// create layout
Rectangle surfaceContentBox = new Rectangle(0.0, 0.0, 2976.3779, 2125.9842);
LayoutBuilder layoutBuilder = new LayoutBuilder(surfaceContentBox);

ctm = new Matrix();
clipBox = new Rectangle(0.0, 0.0, 2976.3779, 2125.9842);
layoutBuilder.addPlacedObject(nf.createMarkObject(ctm, clipBox, 0));

ctm = new Matrix(0, -1, 1, 0, 1731.9680, 374.1712);
clipBox = new Rectangle(1731.9685, 85.0393, 2939.5275, 374.1732);
trimCtm = new Matrix(0, -1, 1, 0, 1734.8031, 371.3385);
trimSize = new XYPair(813.5433, 425.1968);
layoutBuilder.addPlacedObject(nf.createContentObject(ctm, clipBox,
                                                    0, trimCtm, trimSize));

ctm = new Matrix(0, -1, 1, 0, 1731.9680, 663.3051);
clipBox = new Rectangle(1731.9685, 374.1732, 2939.5275, 663.3070);
trimCtm = new Matrix(0, -1, 1, 0, 1734.8031, 660.4724);
trimSize = new XYPair(283.4645, 1201.8897);
layoutBuilder.addPlacedObject(nf.createContentObject(ctm, clipBox,
                                                    1, trimCtm, trimSize));

```

```
ctm = new Matrix();
clipBox = new Rectangle(0.0, 0.0, 2976.3779, 2125.9842);
layoutBuilder.addPlacedObject(nf.createMarkObject(ctm, clipBox, 1));

Layout layout = layoutBuilder.build();

XJdfBuilder xJdfBuilder = new XJdfBuilder("FA-SIG-123456", "Web2Print");
xJdfBuilder.addParameter(layout);
XJDF xjdf = xJdfBuilder.build();
```

3.4. XJdfParser

The XJdfParser writes an XJDF Document Object Tree either to a binary stream or to a byte array and vice versa. Cases of practical use are dealing with XJDF Documents and http transmissions or working on file system. When using binary streams, internally the parser is working with the Java interfaces *java.io.InputStream* and *java.io.OutputStream*. So it doesn't matter which kind of stream is used for reading or writing. The following is a sample of how to save an XJDF Document to a local file system.

```
// any XJDF Document
XJDF xJdf = [...];

// target file
File tmpFile = new File("/var/tmp/myXJdfDoc.xjdf");
OutputStream os = new FileOutputStream(tmpFile);

// write XJDF Document to file using XJdfParser
XJdfParser xJdfParser = new XJdfParser();
xJdfParser.parseXJdf(xJdf, os);

// close stream
os.close();
```

The other option is working with a byte array. Here a sample of how to get a Byte Array from a XJDF Document:

```
// any XJDF Document
XJDF xJdf = [...];

// write XJDF Document to file using XJdfParser
XJdfParser xJdfParser = new XJdfParser();
byte[] bytes = xJdfParser.parseXJdf(xJdf);
```

When parsing XJDF Document Object Trees to binary streams, the document is automatically being validated against the XJDF Schema. Internally the *XJdfValidator* class is used to achieve this. In case the document is invalid a *ValidationException* is thrown. The message of the exception lists all points making the document invalid. In order to skip the validation process during parsing there is an optional parameter *skipValidation* in *parseXJdf()* method.

```
// skip validation when parsing
xJdfParser.parseXJdf(xJdf, os, true);
```

In order to create an XJDF Object Tree from a binary stream the *XJdfParser* class contains a method *parseStream()*. This method accepts an *InputStream* as input parameter. Out of the box the Java framework provides many different implementations of this interface. All *InputStreams* can easily be parsed to an XJDF Document Object Tree by calling the method *parseStream()*. The sample below, for example, uses the *FileInputStream* implementation which is responsible for creating an *InputStream* from a local file. An *InputStream* created from an *HttpRequest* also would be suitable and often is being used when working with http transmissions.

```
// open XJDF Document as InputStream
File tmpFile = new File("/var/tmp/myXJdfDoc.xjdf");
InputStream is = new FileInputStream(tmpFile);

// parse stream to XJDF Document Object Tree
XJdfParser xJdfParser = XJdfParser.newInstance();
XJDF xJdf = xJdfParser.parseStream(is);
```

As before, there also is a second method for using Byte Arrays:

```
// XJDF Byte Array
byte[] bytes = [...];

// write XJDF Document to file using XJdfParser
XJdfParser xJdfParser = new XJdfParser();
XJDF xJdf = xJdfParser.parseBytes(bytes);
```

Note

In order to analyze or extract details from an XJDF Document it is recommended to work with XPath expressions. Parsing the whole document and working with the DOM Tree Objects is no longer state of the art. This mechanism consumes time and raises code complexity. Besides, parsing an InputStream is also prone to errors because it requires fully conform documents. CIP4 xJdfLib provides an extra class *XJdfNavigator* for dealing with XPath expressions in XJDF Documents. XJDF is designed for XPath so the preferred way of reading XJDF Documents is XPath.

3.5. XJdfValidator

The XJdfValidator class validates an XJDF Binary Stream against the latest XJDF Schema. A new instance is required for each validation process. So when validating an XJDF Document, first of all a new validator object has to be created. The method *isValid()* runs the validation process and finally returns the result as Boolean.

Out of the box all XJDF Documents created with the library are automatically being validated during the parsing process. This mechanism can be explicitly switched off. For more details about that see XJdfParser.

```
// get binary stream
InputStream xJdfStream = [...]

// new instance of XJdfValidator
XJdfValidator xJdfValidator = new XJdfValidator(xJdfStream);

// get validation result
boolean result = xJdfValidator.isValid();
```

The validation procedure also can be done in a single line:

```
// get binary stream
InputStream xJdfStream = [...]

// process validation in a single line
boolean result = new XJdfValidator(xJdfStream).isValid();
```

If a more detailed output is desired all errors are available as a list of Strings by calling the *getMessages()* method. In order to simplify user output (e. g. for the creation of exception messages) there is an additional method *getMessagesText()*, which converts all messages into a single String value.

```
// get binary stream
InputStream xJdfStream = [...]

// validate
XJdfValidator xJdfValidator = XJdfValidator.newInstance(xJdfStream);
boolean result = xJdfValidator.isValid();

// message output
List<String> messages = xJdfValidator.getMessages();
String msgText = xJdfValidator.getMessagesText();
```

3.6. XJdfNavigator

The XJdfNavigator class provides functionality for reading, modifying and analyzing XJDF Documents using XPath. XPath is a very powerful XML Technology for working with XML Documents. More details about the XPath W3C Standard can be found here: <http://www.w3.org/TR/xpath/>.

XJdfNavigator directly works on InputStream objects or Byte Arrays, so there is no need to parse the document before hand. This mechanism saves time, code complexity and performance. One XJdfNavigator instance is required for each XJDF Document processed. There are several methods for reading, modifying and analyzing the document.

3.6.1. XPath Expressions

The following is a short XPath overview of expressions which are significant to XJDF Documents. The XJDF snippet after is used for extracting these attribute values:

Table 3.1. XPath expressions

XPath Expression	Attribute Value
/XJDF/GeneralID/@IDUsage	"CatalogID"

XPath Expression	Attribute Value
/XJDF/ParameterSet[@Name='RunList']/Parameter[./Part/@Run='Cover']/RunList/FileSpec/@URL	"cover.pdf"
/XJDF/ProductList/Product/@Amount	"1500"

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by CIP4 xJdfLib 0.4 -->
<xjdf:XJDF xmlns:xjdf="http://www.CIP4.org/JDFSschema_2_0" ID="XJDF_2GYLJJJG"
  JobID="FA-SIG-123456" Category="Web2Print" Version="2.0">
  <xjdf:GeneralID IDUsage="CatalogID" IDValue="46" />
  <xjdf:ProductList>
    <xjdf:Product Amount="1500"/>
  </xjdf:ProductList>
  <xjdf:ParameterSet Name="RunList">
    <xjdf:Parameter>
      <xjdf:RunList>
        <xjdf:FileSpec URL="cover.pdf"/>
      </xjdf:RunList>
      <xjdf:Part Run="cover"/>
    </xjdf:Parameter>
    <xjdf:Parameter>
      <xjdf:RunList>
        <xjdf:FileSpec URL="body.pdf"/>
      </xjdf:RunList>
      <xjdf:Part Run="body"/>
    </xjdf:Parameter>
  </xjdf:ParameterSet>
</xjdf:XJDF>
```

Commonly used XPath Expression also are provided as Java Constants in XJdfNavigator as well. Here a list of such predefined XPath Expressions:

Table 3.2. Java Constants of XPath Expressions

Java Constant Name	Expression
JOB_ID	/XJDF/@JobID
CATEGORY	/XJDF/@Category

Java Constant Name	Expression
GENERAL_CATALOG_ID	/XJDF/GeneralID[@IDUsage='CatalogID']/ @IDValue
GENERAL_LINE_ID	/XJDF/GeneralID[@IDUsage='LineID']/ @IDValue
FILE_SPEC_URL	/XJDF/ParameterSet[@Name='RunList']/ Parameter/RunList/FileSpec/@URL
MIN_APPROVALS	/XJDF/ ParameterSet[@Name='ApprovalParams']/ Parameter/ApprovalParams/ @MinApprovals
CUSTOMER_ID	/XJDF/ ParameterSet[@Name='CustomerInfo']/ Parameter/CustomerInfo/@CustomerID
AMOUNT	/XJDF/ProductList/Product/@Amount
MEDIA_QUALITY	/XJDF/ProductList/Product/ Intent[@Name='MediaIntent']/ MediaIntent/@MediaQuality
LAYOUT_FINISHED_DIMENSIONS	/XJDF/ProductList/Product/ Intent[@Name='LayoutIntent']/ LayoutIntent/@FinishedDimensions
LAYOUT_DIMENSIONS	/XJDF/ProductList/Product/ Intent[@Name='LayoutIntent']/ LayoutIntent/@Dimensions

Java Constant Name	Expression
PRODUCTION_PRINT_PROCESS	/XJDF/ProductList/Product/ Intent[@Name='ProductionIntent']/ ProductionIntent/@PrintProcess
FOLDING_CATALOG	/XJDF/ProductList/Product/ Intent[@Name='FoldingIntent']/ FoldingIntent/@FoldingCatalog
COLOR_NUM_COLORS	/XJDF/ProductList/Product/ Intent[@Name='ColorIntent']/ColorIntent/ @NumColors

3.6.2. Read and Modify Attributes

All attributes in document easily can be addressed and read using the method *readAttribute()* and the specific XPath expression as parameter. Modifications also can be done. The method *updateAttribute()* accepts a XPath expression plus the (new) attribute value. When all modifications are done finally the new XJDF Document is returned as *InputStream* by calling the *getXJdfStream()* method or as *Byte Array* by calling the *getXJdfBytes()* method.

```
// load XJDF as InputStream
InputStream is = XJdfLibSamples.class.getResourceAsStream(RES_SIMPLE_PRODUCT);
XJdfNavigator nav = new XJdfNavigator(is);

// define xPath
String xPath = "/XJDF/ProductList/Product/Intent[@Name='LayoutIntent']";
xPath += "/LayoutIntent/@FinishedDimensions";

// read finished dimensions
String att = nav.readAttribute(xPath);

// modify attribute
att = att.replace("0.0", "2.222");

// update finished dimensions
nav.updateAttribute(xPath, att);
```

```
// output
byte[] bytes = nav.getXJdfBytes();
System.out.println(new String(bytes));
```

The sample above is working with String objects. The XJdfNavigator also provides a second way for reading and modifying based on XJDF Data Types. The following a Java code snippet which has quite exactly the same functionality but is using XJDF Data Types and Java Constants for XPath expressions:

```
// load XJDF as InputStream
InputStream is = XJdfLibSamples.class.getResourceAsStream(RES_SIMPLE_PRODUCT);
XJdfNavigator nav = new XJdfNavigator(is);

// define xPath
String xPath = XJdfNavigator.LAYOUT_FINISHED_DIMENSIONS;

// read finished dimensions
Shape attShape = (Shape) nav.readAttribute(xPath, Shape.class);

// modify attribute
attShape = new Shape(attShape.getX(), attShape.getY(), 2.222);

// update finished dimensions
nav.updateAttribute(xPath, attShape);

// output
byte[] bytes = nav.getXJdfBytes();
System.out.println(new String(bytes));
```

3.6.3. Extended XPath Functionality

Specific XPath expressions can be evaluated using the method *evaluate()*. The return type of this method is *Object* but in specific it depends on the XPath expression and the resultant return type. Both parameters must be defined when using this method. The following a table of all supported return types:

Table 3.3. Overview XPathConstants

XPathConstants	Description
XPathConstants.BOOLEAN	Returns a <i>Boolean</i> object. This constant must be used in case the XPath expressions return a Boolean value.
XPathConstants.NUMBER	Returns a <i>Double</i> object. This constant must be used in case the XPath expressions return a Double value.
XPathConstants.STRING	Returns a <i>String</i> object. This constant must be used in case the XPath expressions return a String value.
XPathConstants.NODE	Returns an <i>org.w3c.dom.Node</i> object. This constant must be used in case the XPath expression returns a single node.
XPathConstants.NODESET	Returns an <i>org.w3c.dom.NodeList</i> object. This constant must be used in case the XPath expression returns multiple nodes.

The next few samples demonstrate how to use the *evaluate()* method in detail:

```
// load XJDF as InputStream
InputStream is = XJdfLibSamples.class.getResourceAsStream(RES_SIMPLE_PRODUCT);
XJdfNavigator nav = new XJdfNavigator(is);

// get number of Intent nodes
String xpath = "count(/XJDF/ProductList/Product/Intent)";
double cnt = (Double) nav.evaluate(xpath, XPathConstants.NUMBER);

// print result
System.out.println("Count: " + cnt);
```

```
// load XJDF as InputStream
InputStream is = XJdfLibSamples.class.getResourceAsStream(RES_SIMPLE_PRODUCT);
XJdfNavigator nav = new XJdfNavigator(is);
```

```
// read names of all Intent nodes
String xpath = "/XJDF/ProductList/Product/Intent";
NodeList nodes = (NodeList) nav.evaluate(xpath, XPathConstants.NODESET);

for (int i = 0; i < nodes.getLength(); i++) {
    Node node = nodes.item(i);
    String name = node.getAttributes().getNamedItem("Name").getTextContent();
    System.out.println(name);
}
```

```
// load XJDF as InputStream
InputStream is = XJdfLibSamples.class.getResourceAsStream(RES_SIMPLE_PRODUCT);
XJdfNavigator nav = new XJdfNavigator(is);

// update node
String xpath = "/XJDF/ProductList/Product/Intent[@Name='MediaIntent']/MediaIntent";
Node node = (Node) nav.evaluate(xpath, XPathConstants.NODE);
node.getAttributes().getNamedItem("MediaQuality").setNodeValue("IPM_170");

// output
byte[] bytes = nav.getXJdfBytes();
System.out.println(new String(bytes));
```

3.7. XJdfPackager

The XJdfPackager is responsible for the packaging of an XJDF Document including all its references into a ZIP Package. One instance is required for each XJDF Document processed. If necessary, the compression level can be adjusted manually using the method `setCompressionLevel()`. The optional parameter "docName" of method `packageXJdf()` defines the name of the XJDF Document file within the ZIP Package. Out of the box the documents name is the XJDFs JobID plus the extension ".xjdf". The following there is a demonstration how to use the XJdfPackager class:

```
// path to artwork
String pathArtwork = XJdfLibSamples.class.getResource(RES_PDF_ARTWORK).getFile();

// build XJDF Document
XJdfNodeFactory nf = new XJdfNodeFactory();

ProductBuilder productBuilder = new ProductBuilder(1500);
Product product = productBuilder.build();
```

```
XJdfBuilder xJdfBuilder = new XJdfBuilder("JOB-12345678");
xJdfBuilder.addProduct(product);
xJdfBuilder.addParameter(nf.createRunList(pathArtwork));
XJDF xJdf = xJdfBuilder.build();

byte[] bytes = new XJdfParser().parseXJdf(xJdf, true);

// package to temporarily file
File tmp = File.createTempFile("XJdfPackage", "zip");
tmp.deleteOnExit();
OutputStream os = new FileOutputStream(tmp);

XJdfPackager packager = new XJdfPackager(bytes);
packager.setCompressionLevel(CompressionLevel.BEST_SPEED);
packager.packageXJdf(os);

os.close();
```

3.8. XJdfConstants

When working with XJDF, there are several constants which are required in some cases. So the CIP4 xJdfLib also provides a static class *XJdfConstants*, where most common constants are already defined. Here is a list of all items in this class:

Table 3.4. Overview xJdfLib Constants

Constant	Value	Use
NAMESPACE_JDF20	"http://www.cip4.org/JDFSchema_2_0"	JDF Default Namespace
NAMESPACE_W3_XML	"http://www.w3.org/2001/XMLSchema"	W3C XML Namespace
XJDF_CURRENT_VERSION	"2.0"	Current JDF Version Number
XJDF_LIB_VERSION	"0.4"	xJdfLib Version Number
XJDF_LIB_BUILD_DATE	"2013-02-24 13:19"	xJdfLib Build Date
JMF_ICS_VERSION	"JMF_L1-2.0"	JMF ICS Version Number

Constant	Value	Use
MEDIA_TYPE_VND_JMF	"application/vnd.cip4-jmf+xml"	MIME Type JMF
MEDIA_TYPE_VND_JDF	"application/vnd.cip4-jdf+xml"	MIME Type JDF

All constants are static and public. So they can be easily accessed by typing the class name XJdfConstants and the specific name of the constant:

```
// get XJDF default namespace
String defaultNamespace = XJdfConstants.NAMESPACE_JDF20;

// get current version of XJDF
String currentVersion = XJdfConstants.XJDF_CURRENT_VERSION;
```

3.9. Util Classes

Util classes provide tools and other simplifications for simplifying working with XJDF. The following is a list of all utils with a short description:

Table 3.5. Overview Util classes

Util Class	Description
IDGeneratorUtil	Generation of randomly created alphanumeric 8-digits IDs with or without customized prefix.
DimensionUtil	Conversion from millimeter to dtp and the way around.

3.10. Performance Optimizations

The CIP4 XJDF Library is based on the Java JAX-B Framework. The framework brings a lot of benefits to the CIP4 xJdfLib but unfortunately the initialization takes a few seconds. Usually this happens automatically when the first XJDF Document is being parsed. In some

cases this is not acceptable, thus the library provides additional functionality for manual initialization. The XJdfFactory class is responsible for such a manual initialization process. The *init()* method starts the process synchronously. By setting the parameter "initAsync", the same process starts asynchronously. A recommendation is using the asynchronous method when the application starts. By the way the manual initialization process also initializes some other components.

```
// synchronous initialization
XJdfFactory.init();

// asynchronous initialization
XJdfFactory.init(true);
```

Chapter 4. Developer Infos

This chapter describes the internals of the XJDF Library. This is useful when you are interested in getting involved in development. Otherwise feel free to skip it.

4.1. Update XJDF Schema

Generating new sources after the XJDF Schema has changed is a very common task. The XSD is located at "src/main/resources/org/cip4/lib/xjdf/xsd/JDF20.xsd". In order to update sources, just overwrite the schema file and run the "updateXJdfXSD.bat" batch script. All files located in the java package "org.cip4.lib.xjdf.schema.jdf" are automatically replaced.

Chapter 5. Conclusion

Currently the CIP4 xJdfLib project is still in a very early stadium. In addition the library is based on a specification which has not been completely finished yet. So when using the library please keep in mind that concepts or even the XJDF structure may change from one version to the next. The first official stable version is indicated by starting with 1 in version number (1.x).

Feature requests and bug reports are always very welcome. Please do not hesitate to create a new ticket for your issues. In CIP4 JIRA Ticket System there you will find the project "xJdfLib", where you can place new posts.