



# ***CIP4 xJdfLib***

***Version: 0.3***

*Date: 2012-08-23*



## **Abstract**

This document describes the structure and functionality of CIP4 xJdfLib and how to use it. The library was written in Java and simplify dealing with JDF / 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 .....	2
3. Core Components .....	3
3.1. XJdfNodeFactory .....	3
3.2. Builder Classes .....	4
3.2.1. XJdfBuilder .....	6
3.2.1.1. Partitioning of Parameter nodes .....	7
3.2.2. ProductBuilder .....	8
3.2.3. ContactBuilder .....	9
3.3. XJdfParser .....	10
3.4. XJdfValidator .....	11
3.5. XJdfNavigator .....	12
3.6. XJdfConstants .....	14
3.7. Converter Classes .....	15
4. Developer Infos .....	17
4.1. Update XJDF Schema .....	17
5. Conclusion .....	18

---

## List of Tables

3.1. XPath expressions .....	14
3.2. Overview xJdfLib Constants .....	15
3.3. Overview Converter classes .....	16

# Chapter 1. Introduction

The major conceptual change in JDF 2.0 (or XJDF) is the technology does 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 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). Further there even are implemented some very common XML functionalities like creating Java objects from XSD Schema and so on.

Remember, 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 already had existed. Further the XML technology itself was in a very early phase of development. So it was essential to include all this stuff mentioned getting JDF working.

Over the last two decades XML has become very popular. Many tools were designed and many conceptual enhancements in XML has been done. For example one significant enhancement has been 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 XJDF library now 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, up to date 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 7. The latest stable version is being published on the CIP4 Website on page “Technical Resources -> Downloads -> Internal Source”. Further xJdfLib 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>0.3</version>
</dependency>
```

Early Next-Version-Snapshots of 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>0.4-SNAPSHOT</version>
</dependency>
```

# Chapter 3. Core Components

## 3.1. XJdfNodeFactory

The XJdfNodeFactory is the fundamental factory class for creating new instances of XJDF Nodes. The class provides at least one Creation-Method per XJDF Node. For plain commonly used XJDF Nodes (e. g. GeneralID, RunList etc.) there also are extended Creation-Methods which additionally do initialize the node object after creation.

A new instance of XJdfNodeFactory can be created by calling the static method *newInstance()*:

```
// New Factory Instance
XJdfNodeFactory xJdfNodeFactory = XJdfNodeFactory.newInstance();
```

The next two code snippets demonstrate the difference and usage of normal and extended Creation-Methods. For example when working with GeneralIDs most of them only consist of the attributes "IDUsage" and "IDValue". Using the flexible way the creation of GeneralID instances takes three lines of code. The first line creates the new object whereas the following two lines initialize the object by attribute values. Each attribute explicitly can be set and modified:

```
// New GeneralID XJDF Node Object using the flexible way
GeneralID generalId = xJdfNodeFactory.createGeneralID();
generalId.setIDUsage("IDCatalog");
generalId.setIDValue("42");
```

But when creating larger XJDF Documents the flexible way unnecessarily does consume time for writing and raise complexity enormously. Hence the factory class besides provides extended Creation-Methods for a straighter way. Using this methods plain commonly used XJDF Node structures can be created and initialized by a single line of code:

```
// New GeneralID XJDF Node Object using the straight way
GeneralID generalId = xJdfNodeFactory.createGeneralID("IDCatalog","42");
```

Both ways produce one and the same GeneralID Node. The flexible way raises flexibility of attribute management whereas the straight one decreases complexity, maintenance and produces more clearly source code.

Best practice is the combination of both ways. It is strongly recommended using the extended Creation-Methods whenever one is available. If the extended method does not fully fit the needs the result always can be individualized in a further step after. All attributes always are able being modified directly.

Following example showcases how to modify an XJDF Node after creation. In some situations the */RunList/FileSpec* Node may contain an additional attribute *UserFileName*:

```
[...]
<RunList>
  <FileSpec URL="http://192.168.1.113:80/4daction/Poster/10496"
    UserFileName="myFileName.pdf"/>
</RunList>
[...]
```

In the latest version of XJdfNodeFactory for example there is no extended Creation-Method for the creation of a *RunList* node which also defines a *UserFileName* attribute in *FileSpec*. The best practice is using the extended method despite and finally individualize the result:

```
// Best practice creating individual nodes
RunList runList = xJdfNodeFactory
    .createRunList("http://192.168.1.113:80/4daction/Poster/10496");
runList.getFileSpec().setUserFileName("myFileName.pdf");
```

## 3.2. Builder Classes

Plain XJDF Node elements easily can be created using the XJdfNodeFactory. Complex elements (e. g. XJDF-Root-Node, Product-Node, etc.) are elements which contains multiple subnodes in a defined manner. For example you can say the XJDF-Root-Node is the container for Product and Parameter nodes. All *Product* items are listed in subelement *ProductList*. Also all *Parameter* nodes are listed in specific *ParameterSet* elements.

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

```
<XJDF xmlns="http://www.CIP4.org/JDFSSchema_2_0" Category="Web2Print"
  DescriptiveName="My lovely Poster" JobID="FA-SIG-123456">
  <ProductList>
    <Product DescriptiveName="1234_56" Amount="1500" ProductType="Poster"
      ProductID="FA-SIG-123456" ProductTypeDetails="PTD Value" >
      <Intent Name="MediaIntent">
        <MediaIntent MediaQuality="IPM_90"/>
      </Intent>
    </Product>
  </ProductList>
  <ParameterSet Name="RunList">
    <Parameter>
      <RunList>
        <FileSpec URL="test_file.pdf"/>
      </RunList>
    </Parameter>
  </ParameterSet>
  <ParameterSet Name="Contact">
    <Parameter>
      <Contact ContactTypes="Customer Delivery">
        <Address PostalCode="97080" City="Würzburg"
          Country="Germany" Street="Alfred-Nobel-Straße 18"/>
        <ComChannel ChannelType="Email" Locator="info@flyeralarm.de"/>
        <ComChannel ChannelType="Phone" Locator="+49.931.465840"/>
        <Company OrganizationName="flyeralarm GmbH"/>
      </Contact>
    </Parameter>
  </ParameterSet>
</XJDF>
```

Builder classes are designed and optimized for simplify creating the container element and dealing with all subnodes in a correct manner. In case of XJdfBuilder when adding a new *Parameter* node the class automatically checks whether the right *ParameterSet* element already does exist. If not so the library creates a new one and finally embeds the new parameter.

The CIP4 XJdfLib provides several element builders:

- **XJdfBuilder**

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

- **ProductBuilder**

Creation of Product nodes. Handles Intent nodes.

- **ContactBuilder**

Creation of Contact nodes. Simplify handling with contact details.

### 3.2.1. XJdfBuilder

The builder class XJdfBuilder covers logic to simplify the creation of XJDF Root Nodes and the handling of their Product and Parameter definitions. In order to add subelements there are several "add"-methods to achieve this. Secondary these methods are responsible keeping the structure of the XJDF Document in a well defined manner. For instance the XJDF Specification defines all Parameter nodes have to be embedded within a specific ParameterSet element. This mechanism fully is supported by the builder class. So when adding a new Parameter item the addParameter() method checks whether or not the right ParameterSet node already does exist. In case the ParameterSet node does not exist yet a new one is created automatically. Following a Java Code Snippet how to use the XJdfBuilder for creating the XJDF document above:

```
// new xJdfBuilder
XJdfBuilder xJdfBuilder = XJdfBuilder.newInstance("FA-SIG-123456",
                                                "Web2Print", "My lovely Poster");

// create Product node
ProductBuilder productBuilder = ProductBuilder.newInstance();
[...]
Product product = productBuilder.build();

// create Contact node
ContactBuilder contactBuilder = ContactBuilder.newInstance();
[...]
Contact contact = contactBuilder.build();

// create RunList
RunList runList = XJdfNodeFactory.newInstance().createRunList("test_file.pdf");

// append product
xJdfBuilder.addProduct(product);

// append parameters
xJdfBuilder.addParameter(contact);
xJdfBuilder.addParameter(runList);

// build XJDF Doc
```

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

### 3.2.1.1. Partitioning of Parameter nodes

The XJDF Specification defines a partitioning mechanism of Parameters. Here an example of a partitioned RunList in a XJDF Document:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XJDF xmlns="http://www.CIP4.org/JDFSchema_2_0">
  [...]
  <ParameterSet Name="RunList">
    <Parameter>
      <Part Run="Cover"/>
      <RunList>
        <FileSpec URL="./artwork/cover.pdf"/>
      </RunList>
    </Parameter>
    <Parameter>
      <Part Run="Body"/>
      <RunList>
        <FileSpec URL="./artwork/body.pdf"/>
      </RunList>
    </Parameter>
  </ParameterSet>
  [...]
</XJDF>
```

In XJdfBuilder class there is a particular addParameter() method override for partitioning. Following the appendant code snippet for the creation of a partitioned RunList ParameterSet:

```
// new factory
XJdfNodeFactory xJdfNodeFactory = XJdfNodeFactory.newInstance();

// new XJdfBuilder
XJdfBuilder xJdfBuilder = XJdfBuilder.newInstance();

// parameter cover
RunList runListCover = xJdfNodeFactory.createRunList("./artwork/cover.pdf");

Part partCover = xJdfNodeFactory.newInstance().createPart();
partCover.setRun("Cover");

// parameter body
RunList runListBody = xJdfNodeFactory.createRunList("./artwork/body.pdf");
```

```
Part partBody = XJdfNodeFactory.newInstance().createPart();
partBody.setRun("Body");

// add parameters with partitioning
xJdfBuilder.addParameter(runListCover, partCover);
xJdfBuilder.addParameter(runListBody, partBody);

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

### 3.2.2. ProductBuilder

The main task of ProductBuilder is the creation and initialization of new product instances. Further the classes proper is the handling of product Intents. In XJDF each product intent has to be embedded within a separate Intent node. This mechanism also is covered by the ProductBuilder. When adding a new concrete Intent node (e. g. MediaIntent, LayoutIntent etc.) the class manages the structure.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XJDF xmlns="http://www.CIP4.org/JDFSchema_2_0">
  [...]
  <ProductList>
    <Product ProductTypeDetails="MyJob" Amount="1500" ProductType="Poster"
      ProductID="FA-SIG-123456" >
      <Intent Name="MediaIntent">
        <MediaIntent Weight="135.0" />
      </Intent>
      <Intent Name="LayoutIntent">
        <LayoutIntent FinishedDimensions="595.27559055 822.04724409" />
      </Intent>
      <Intent Name="BindingIntent">
        <BindingIntent BindingType="SaddleStitch" />
      </Intent>
    </Product>
  </ProductList>
  [...]
</XJDF>
```

The snippet above shows a Product definition within a XJDF Document. Using the ProductBuilder class there is no longer a need managing the document structure manually. The only thing to do is creating the particular Intent elements and add them to the builder:

```
// new ProductBuilder instance
ProductBuilder productBuilder = ProductBuilder.newInstance(1500,
    "FA-SIG-123456", "Poster", "MyJob");

// create Intents
MediaIntent mediaIntent = [...];
LayoutIntent layoutIntent = [...];
BindingIntent bindingIntent = [...];

// add Intents
productBuilder.addIntent(mediaIntent);
productBuilder.addIntent(layoutIntent);
productBuilder.addIntent(bindingIntent);

// build product
Product product = productBuilder.build();
```

### 3.2.3. ContactBuilder

The Contact Node in XJDF keeps all customers contact details. There are several types of subnodes for holding the information. In order to simplify the creation and initialization of these elements the ContactBuilder class has been designed and developed.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XJDF xmlns="http://www.CIP4.org/JDFSchema_2_0">
    [...]
    <ParameterSet Name="Contact">
        <Parameter>
            <Contact ContactTypes="Customer Delivery">
                <Address PostalCode="97080" City="Würzburg"
                    Country="Germany" Street="Alfred-Nobel-Straße 18"/>
                <ComChannel ChannelType="Email" Locator="info@flyeralarm.de"/>
                <ComChannel ChannelType="Phone" Locator="+49.931.465840"/>
                <Company OrganizationName="flyeralarm GmbH"/>
            </Contact>
        </Parameter>
    </ParameterSet>
    [...]
</XJDF>
```

Using the ContactBuilder Person-, Company-, Address- and ComChannel nodes easily can be created and added by plain method calls:

```
// new ContactBuilder instance
ContactBuilder contactBuilder = ContactBuilder.newInstance();
```

```
// set contact details
contactBuilder.addAddress("Alfred-Nobel-Straße 18", "97080", "Würzburg");
contactBuilder.addComChannel("Email", "info@flyeralarm.de");
contactBuilder.addComChannel("Phone", "+49.931.465840");
contactBuilder.addCompany("flyeralarm GmbH");

// build contact
Contact contact = contactBuilder.build();
```

### 3.3. XJdfParser

The XJdfParser writes an XJDF Document Object Tree to a binary stream and vice versa. Practical use cases are dealing with XJDF Documents and http transmissions or working on file system. Internally the parser class 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. Following a sample how to save a XJDF Document to local file system. For writing an XJDF Document to a http stream just use the specific implementation to achieve this.

```
// 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 = XJdfParser.newInstance();
xJdfParser.parseXJdf(xJdf, os);

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

When parsing XJDF Document Object Trees to a binary streams the document automatically is 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 input streams easily can 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 a *InputStream* from a local file. An *InputStream* created from a *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);
```

## Note

In order to analyze or extract details from a XJDF Document it is strongly recommended working with XPath expressions. Parsing the whole document and working with the DOM Tree Objects is no longer state of the art. This mechanism does consume time and raises code complexity. Besides parsing an *InputStream* also prones 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 reading XJDF Documents is XPath.

## 3.4. XJdfValidator

The XJdfValidator class validates an XJDF Binary Stream against the latest XJDF Schema. A new instance is required per validation process. So when validating an XJDF Document first of all a new validator object has to be created using the static method *newInstance()*. 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 automatically are being validated during parsing process. This mechanism explicitly can be switched off. For more details about that see XJdfParser.

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

// new instance of XJdfValidator
XJdfValidator xJdfValidator = XJdfValidator.newInstance(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 = XJdfValidator.newInstance(xJdfStream).isValid();
```

If a more detailed output is desired all errors are available as a list of Strings by calling the method *getMessages()*. In order to simplify user output (e. g. for the creation of exception messages) additionally there is an 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();
```

### 3.5. XJdfNavigator

The XPathNavigator class provides functionality for reading and modifying 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/>.

XPathNavigator directly works on `InputStream` objects. So there is no need parsing the document before. This mechanism saves time, code complexity and performance. One XPathNavigator instance is required per XJDF Document processed. The method `newInstance()` creates a new object and initializes it by the XJDF Document `InputStream`. 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 stream is returned as `InputStream` by calling the method `getXJdfStream()`:

```
// load XJDF Document as InputStream and create new XPathNavigator.
InputStream is = [...]
XPathNavigator xPathNavigator = XPathNavigator.newInstance(is);

// read JobID
String xJobId = "/XJDF/@JobID";
String jobId = xPathNavigator.readAttribute(xJobId);

// read GeneralID IDUsage
String xIdUsage = "/XJDF/GeneralID/@IDUsage";
String idUsage = xPathNavigator.readAttribute(xIdUsage);

// read FileSpec URL of partition "Cover"
String xFileCover = "/XJDF/ParameterSet[@Name='RunList']/"
    + "Parameter[./Part/@Run='Cover']/RunList/FileSpec/@URL";
String fileCover = xPathNavigator.readAttribute(xFileCover);

// update Product Amount to 1500
String xAmount = "/XJDF/ProductList/Product/@Amount";
xPathNavigator.updateAttribute(xAmount, "1500");

// get modified XJDF Document as InputStream
InputStream is = xPathNavigator.getXJdfStream();
[...]
```

Following 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"
/XJDF/ParameterSet[@Name='RunList']/Parameter[./Part/@Run='Cover']/RunList/FileSpec/@URL	"cover.pdf"
/XJDF/ProductList/Product/@Amount	"1000"

```

<?xml version="1.0" encoding="UTF-8"?>
<XJDF xmlns="http://www.CIP4.org/JDFSSchema_2_0" Category="Web2Print"
  DescriptiveName="PrintJob 123456" JobID="FA-SIG-123456">
  <GeneralID IDUsage="CatalogID" IDValue="46" />
  <ProductList>
    <Product Amount="1000" />
  </ProductList>
  <ParameterSet Name="RunList">
    <Parameter>
      <Part Run="Cover" />
      <RunList>
        <FileSpec URL="cover.pdf" />
      </RunList>
    </Parameter>
    <Parameter>
      <Part Run="Body" />
      <RunList>
        <FileSpec URL="body.pdf" />
      </RunList>
    </Parameter>
  </ParameterSet>
</XJDF>

```

## 3.6. XJdfConstants

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

Table 3.2. 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
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 easily can be 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;
```

## 3.7. Converter Classes

In order to convert from Java datatypes to JDF or ISO datatypes and the way around the library provides several converter classes. Following a list of all converters with a short description:

Table 3.3. Overview Converter classes

Converter Class	Description
DateConverter	Conversion between the Java Calendar object and the ISO Date String how specified in XJDF Specification.
DimensionConverter	Conversion from millimeter to dtp and the way around.

## Chapter 4. Developer Infos

This chapter tells about the internals of the XJDF Library. This is useful when you are interest in getting involved to development. Otherwise fell 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" automatically are being 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 always are 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.