jadice server

Juli 2013

jadice® server
Version 4.5.1.0

Developer's Guide

jadice server

Table of Contents

1. General

1.1. About this documentation

This guide in hand is an introduction to the technical coherences of the jadice server®.

This documentation is basically limited to the areas which are interesting to developers (subsequently called integrators) in order to integrate jadice server® in their own applications.

An API reference in *javadoc* format is made available as a separate document.

1.2. Feedback

If you come across any errors when using this documentation or if you like to suggest any improvements, please send a possibly detailed message to **solutions@levigo.de**.

Your feedback helps us in further developing this documentation. Thanks a lot.

1.3. Online-Service

For developers and integrators we offer the **levigo support center** at **https://levigo.de/support/** where you can report your requests, problems, suggestions etc concerning the jadice document platform and address them to the jadice developers. You will be automatically informed about all responses, the status of the problem and its solution via email. Of course you can add additional information or consideration, too. If interested please contact us at **solutions@levigo.de**.

In the **jadice knowledgebase** at **https://levigo.de/info/display/JKB/Home** you can find different background articles, tips and tricks. They provide valuable information beyond the contents of this documentation and contribute towards a better understanding of the topics surrounding the jadice product family.

Additionally, we provide a variety of open source projects and code examples in our **github repositories** at **https://github.com/levigo/**. You can use them for your own projects freely. Of course we are glad if you wish to contribute to the development and send us a pull request.

1.4. About the jadice product family

The **jadice document platform** is a technology developed on JAVA with a central component for document processing. Due to its flexibility as an easy-to-integrate toolbox it may be applied in different ways and it offers the basis for individual archiving client and server solutions in the professional document management.

The document viewer **jadice viewer**, the former starting product of the jadice family, has remained an essential part of the jadice document platform – however, with an advanced and expanded functional range.

Based on the jadice document platform there also exist the products jadice server and jadice web toolkit.

jadice server is a component for any server-sided processing of data streams. This all-in-one solution conceived for the documents' processing is operated by simple workflow instructions and cascaded commands. It is possible at any time to change the way of processing since both the origin of the incoming data stream and the output of the processing result may be freely selected.
jadice server is able to react dynamically on data and disposes of powerful modules for format recognition. Its open interfaces allow the integration of further functions, of any software and thus make it possible to cover new formats.

The **jadice web toolkit** is an easy-to-use and highly customizable portal solution for document viewing and document editing. It runs entirely in the web browser

and thus does not require a Java installation on the client. The jadice web toolkit provides easy access and the best usability of the jadice features by using the Google Web Toolkit.

2. jadice server

2.1. The product's concept

jadice server's main task is not to convert a document from type X to type Y. Instead it falls back amongst others on the functionality of the underlying **jadice document platform** which provides many possibilities to convert formats encountered in long time archives.

Then it provides a flexible **interface to applications** which carry out the conversion. This advantage relies amongst others on jadice server's platform independence. That means that jadice server may be installed on different platforms and so it may communicate with a large number of applications.

For this jadice server offers interfaces to externally controlled third-party software. Its control may either be realised directly in the JAVA interface or you can use the COM-interface or write a batch-file.

Depending on the original's format the converting may be carried out either by a software **function** (like e.g. Photoshop) which can display the original format or by using an available **image print driver**. In both cases it is only important to define jadice server's **appropriate interfaces** with the respective software.

Due to jadice server's two-staged architecture nearly any scalability may be reached. It may be extended to any number of plug-ins to different applications. Thus lots of clients may profit from the programmes installed on few computers (on which **jadice server** is installed, too).

2.2. Possible applications of jadice server

The possible applications listed in this chapter describe the typical use cases covered by jadice server. For the precise technical realisation it is referred to the appropriate code examples in the chapter "Application scenarios including code examples" from page 19 on.

Evidently it is not possible at this point to describe all applications offered by jadice server. Besides the respective subtasks may be combined in such a way that they cover precisely the given problem.

2.2.1. Unification and long time archiving

jadice server is suited for working with data which can't be directly displayed by a client like e.g. office formats or special technical formats.

Since the server is responsible for the documents' converting the single clients on the work stations are discharged and thus working on older, less powerful computers is made possible. Additionally the programmes which are necessary for the converting do not have to be installed on all clients but only on the jadice server.

Since these data are now provided in a standard format (e. g. as PDF/A or TIFF) the programme which these files originally were made with does not have to be available any longer. Thus these documents are also appropriate for the long time archiving.

Relevant code examples:

- Converting unknown input data in a unified format (PDF) (page 27)
- Converting Office-documents to PDF (page 28)
- Merging of multiple PDF documents (page 24)
- Converting to TIFF (page 25)

2.2.2. Tiling

A further possible application is the processing and editing of documents with a large number or also a particularly large size of pages.

When working with very large documents in the jadice viewer or in the jadice web toolkit jadice server offers the possibility to analyse documents by tiles, i.e. to load and display in single sections. This is an advantage when large pages, like e.g. construction plans or even only parts of a document with hundreds of pages, are to be displayed.

2.2.3. Virtual documents

In contrast to this jadice server may also combine multiple documents to a large, logical document. So it is possible to combine staff files with time registration documents, sick notes and fuel receipts of a staff member in a unified file.

On the other hand it is also possible to divide large documents in single sub-documents in order to make them accessible for single departments.

Relevant code example:

- Merging of multiple PDF documents (page 24)

2.2.4. Permanent anchoring of annotations

If annotations are brought up on documents, it may be necessary to fix them permanently in the document, if the respective documents are to be forwarded to external places.

This may be necessary for different reasons. For one reason it is not granted that external places may process the annotations' data format, so that a standard format (like PDF) has to be accessed. It is also possible that some document's parts must be "blackened" since they contain business confidential information or protected private data. In this case it must be made sure that the external place does not have the means to make the "blackened" information legible again.

Relevant code example:

- Converting to TIFF (page 25)
- Permanent anchoring of annotations (page 25)

2.2.5. Extraction of meta data

For a fast survey in a client it is not always necessary to transmit the complete document. Instead it is sufficient if it gets meta data which characterize the document.

These may be used e.g. to decide from client-side which further processing steps should be set off.

Relevant code example:

- Extraction of document information (page 23)

2.2.6. Unification of e-mails

For a legally accepted archiving of e-mails it is not only necessary to archive the pure e-mail text but also its attachments. In order to make sure that the divers file formats may be read in future too, these formats have to be brought into a standardised format, though. For this the PDF standard is very convenient.

Apart from this it is desirable that a survey of attachments is created automatically. Both is performed by the jadice server.

Relevant code example:

- Converting e-mails to PDF (page 28)

2.2.7. Central document printing

Print jobs may be created on staff's working places. For this the API provides a large number of possible settings. This configuration is transmitted to jadice server which functions as central print server. In the data centre it takes over the print job and passes it without loss on to the print cluster.

Since the processing done by the print driver takes centrally place with – depending on the setting and desired print quality – possibly large print data streams, not only

the network load may be reduced but it is also possible to continue working on the staff's working place immediately after starting the print job.

2.2.8. Processing of packed files

For transporting large files via internet it is sensible to pack and compress them as archive files , e.g. as a zip file. However, this archive file must be unpacked before processing. In order to automatise this procedure, jadice server may be used.

Relevant code example:

- Controlling of external programmes (page 26)

2.2.9. Data validation

All customer documents which are stored in a long time archive must meet the requirements to be still readable in decades without any problems. For this an appropriate document format must be chosen.

When archiving it must additionally be made sure that the documents are complete and sound and that they correspond formally to the technical specification.

With jadice server a processing step independent of the files' creation may be introduced when importing which recognises file formats and calls up appropriate validation mechanisms against defined standards.

Thus defective files may be recognised in time and sorted out for being checked and re-edited. So it is provided that - even in decades - documents in long time archives will be available for editing.

3. System architecture

Jadice server may be used in a clustered way. Thus a high availability along with a high performance may be realised. Jadice server is performed per server instance in a JVM and administrates a pool of LibreOffice processes respectively different COM servers. Using a messaging system (MOM) which serves as s transport layer clients order the performance of jobs. The access by clients is done via a client library (see chapter 4.2).
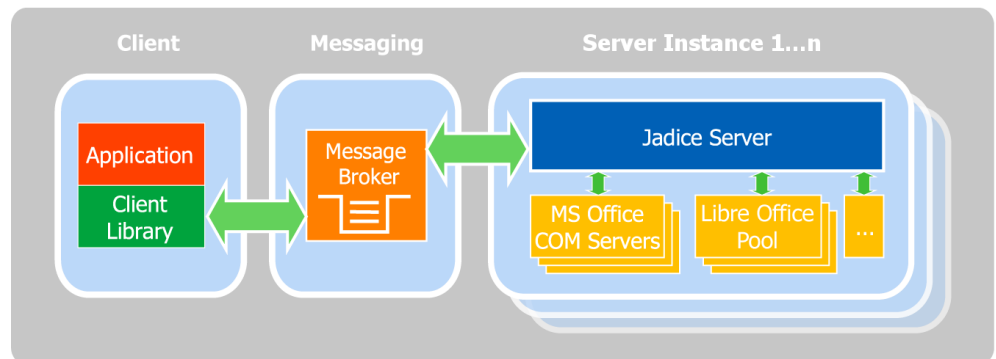
Chart 1: Messaging system as transport layer between clients and jadice server

Since the messaging system is connected by the standardised JMS interface (Java Message Service), message broker already used in the company may be embedded into the system architecture.

Due to this architecture an automated load-balancing along with a high availability may be realised with little effort.

3.1. Functionality

Jadice server is designed in such a way that the processing of documents and document data is separated in jobs[1] which are separated in single processing steps (nodes[2]) thus describing a workflow.

Clients command the jobs' performance and transmit them by messaging system to the jadice server.
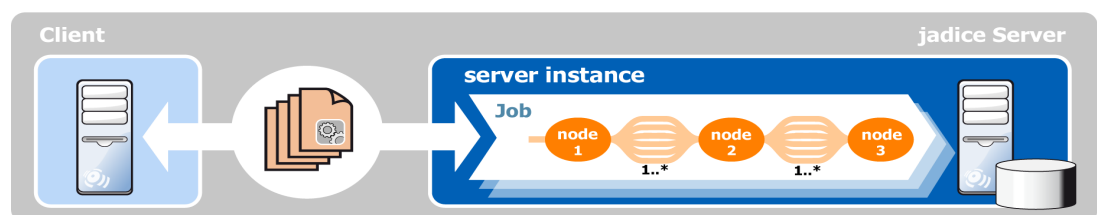
Chart 2: Description of a job with single nodes

Nodes are the single, individually defined processing steps which a job consists of. They are connected amongst each other by bundles of data streams which transport user data and meta data. Consequently the processing steps depend in regard of content and order on the performing job.

The nodes differ from each other in their given task. Thus documents may be converted in one node e.g. with LibreOffice, MS Office or the functionalities of the jadice document platform, whereas in a different node documents are parted or merged. Further nodes may edit meta data and format data. The print processing of documents, the rasterization and providing of data as tile server and the classification of data streams are further examples for nodes. Even the simple packing or unpacking into or out from archives happens in such a processing step.

---

1   com.levigo.jadice.server.Job
2   com.levigo.jadice.server.Node

Here it is not firmly prescribed that each node has got exactly a predecessor or successor. So there is already a predefined node[3] which multiplies a data stream and passes it on to various successor nodes and a node[4] which has got various predecessor nodes and which passes all incoming data streams on to only one successor. The only condition to be respected when configuring a work flow is to avoid cycles in the compiling of nodes.

---

3   com.levigo.jadice.server.nodes.MultiplexerNode
4   com.levigo.jadice.server.nodes.DemultiplexerNode

4. Installation and configuration

4.1. Server

jadice server runs from Java version 1.6.0 update 26 / Java version 1.7.0 Update 3 oder newer. For the installation first the distribution file (jadice-server-4.4.x.x-dist.zip) has to be unpacked into an empty directory on the server's local disk. Any following files or folder names listed in this developer's guide are relative to this directory.

The installation / starting files are in the directory /**bin**.

To install and manage jadice server as a Windows service the following files are provided:

install.bat   Installs jadice server as Win32-service "levigo jadice server 4" .

start.bat   Starts the service.

stop.bat   Stops the service.

Executing these batch files requires administrative rights.

Alternatively the server may be started with the script **jadice-server.bat.**

For Linux-based operating systems the script **jadice-server.sh** is to be used.

At the start a console is opened; using the key combination CTRL-C ends the server.

All jar-files required by the server are in the directory **/server-lib**.

4.1.1. Licence file

Beside jadice server's distribution file you get a separate licence file (**JadiceServer-License.properties**). This file must be inserted in jadice server's configuration directory (**/server-config**).

If this file does not exist or if the licence has expired or become invalid, an error report is written in the server and client log at the start of jadice server and at each requirement of a job.

In case of a temporarily limited evaluation licence a report will be shown only at the server's start. Beyond this jadice server will not be restricted in its functionality.

4.1.2. Manual download for hyphenation support

jadice server uses Apache FOP for converting XML-documents and e-mails into PDF. Due to legal licencing reasons the optional package for hyphenation is not attached to jadice server's distribution package. However, this hyphenation package is available for free under **http://offo.sourceforge.net/**. For installation just copy the file **hyphenation.jar** into the folder **server-lib**.

4.1.3. Configuration of the messaging system

Three variants are provided for the building of a messaging system:

- Embedded broker (standard)
  - Broker working within jadice server's VM
  - Use of Apache ActiveMQ
  - Cluster competent by „network of brokers"
  - Configuration see chapter 4.1.4
- Separate broker
  - Separate broker is used on jadice server's infrastructure
  - Use of any compatible MOM
  - Eventually cluster competent
- Separate MOM-infrastructure
  - Use of separate MOM-infrastructure
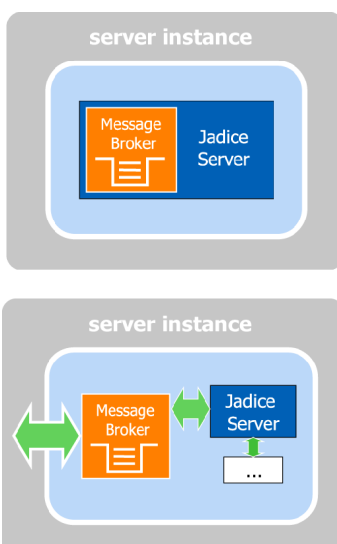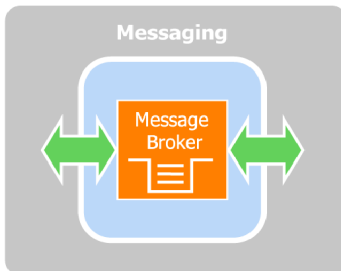
- ◦ Use of any compatible MOM
- ◦ Availability must be granted by infrastructure

Which of these three variants is used, is configured in the file **server-config/application/server.xml**. In the entry <bean id="jms-connection-factory" …> the connector to be used is configured as well as in the paragraph <property name="properties" …> under jadice.server.queue-name the name of the message queue and under jadice.server.activemq-port the port for ActiveMQ to which jadice server is to be connected to. Due to the fact the class ActiveMQConnectionFactory implements both QueueConnectionFactory[5] and TopicConnectionFactory[6], it is referenced with the aliases "jms-queue-connection-factory" and "jms-topic-connection-factory" at the appropriate place.

An example configuration how to connect to IBM Websphere MQ is listed in the file **server-config/application/example/jms.xml**. The adequate configuration replaces the entries <bean id="jms-queue-connection-factory" …> and <bean id="jms-topic-connection-factory" …> in the file **server-config/application/server.xml**.

4.1.4. Configuration of embedded message broker

Jadice server provides by default in the server instance the messaging system (i.e. the message broker) to be used. In this case Apache ActiveMQ[7] is used as messaging system. It is configured in the file **server-config/application/activemq-broker.xml**. In order to encode the transmission between client and server by SSL, there are instructions under http://activemq.apache.org/ssl-transport-reference.html and http://activemq.apache.org/how-do-i-use-ssl.html how this may be realised.

The port and the name of the message queue in use are centrally configured in the file **server-config/application/server.xml**, see paragraph 4.1.3.

To avoid starting this internal broker the paragraph <bean id="broker" …> in the file **server-config/application/server.xml** must be commented out.

4.1.4.1. Clustering

The following configuration change is necessary to operate jadice server via ActiveMQ in a cluster.

- Delete in file **activemq-broker.xml** the comment signs around one of the two elements <networkConnector …>.

The cluster has got the following structure:

- On each node runs an instance of jadice server each with an embedded broker under Apache ActiveMQ.
- The clustering is based on an ActiveMQ Network-of-Brokers[8], i. e. the embedded broker of each node participates equally in a shared broker.
- The brokers find each other either by Auto-Discovery[9] which is realised by Multicast or the Network-of-Brokers is defined statically.

In order to render the clustering effective the clients have to be configured correspondingly. If the brokers find each other by auto-discovery, the method Multicast-Discovery[10] may be used for client connections. For this the group name of the cluster built by jadice server instances has to be known. This cluster is set in file **/server-config/application/server.xml** under jadice.server.activemq-group and is named jadice-server.cluster by default.

---

5 javax.jms.QueueConnectionFactory
6 javax.jms.TopicConnectionFactory
7 See http://activemq.apache.org/
8 See http://activemq.apache.org/networks-of-brokers.html
9 See http://activemq.apache.org/discovery.html
10 See http://activemq.apache.org/discovery-transport-reference.html

The URL by which a client may connect in this case to a cluster is (compare code example in chapter 5.3.1):

```
discovery:(multicast://default)?
  group=jadice-server.cluster&initialReconnectDelay=100
```

If a static Network-of-Brokers has been defined, the connecting URL has to be defined statically, too.

```
failover:(<URL_Server_A>,<URL_Server_B>)
// example:
failover:(tcp://serverA:61616,tcp://serverB:61616)
```

4.1.5. Configuration wrapper

The server is started by a platform independent wrapper. Here JAVA VM and class path parameters are defined.

The wrapper configuration file **wrapper.conf** is situated in the directory **/wrapper**.

Path to JAVA VM:

```
# Java Application
wrapper.java.command=java
```

Definition of further class path elements:

```
# Java Classpath (include wrapper.jar).  Add class path
# elements as needed starting from 1
wrapper.java.classpath.1=../wrapper/lib/wrappertest.jar
wrapper.java.classpath.2=../wrapper/lib/wrapper.jar
wrapper.java.classpath.3=./bin/server-console.jar
wrapper.java.classpath.4=../msoffice-lib

# Example definition of a further class path element:
wrapper.java.classpath.N[11]=<additional element>
```

If your Java Virtual Machine (JVM) requires additional parameters, e.g. setting the temporary directory for the VM, these parameters can be applied in the following manner:[12]

```
# Example definition of additional JVM parameters
wrapper.java.additional.1=-server
wrapper.java.additional.2=-Djava.io.tmpdir=C:\tmp
wrapper.java.additional.N[13]=<another parameter>
```

Definition of JAVA VM's saving properties, wrapper.java.initmemory corresponds to parameter -Xms, wrapper.java.maxmemory corresponds to parameter -Xmx:[14]

```
# Initial Java Heap Size (in MB)
#wrapper.java.initmemory=3

# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=512
```

Apart from this it is possible to change the class path and in doing so the loaded Java-libraries as well, in order to add e.g. own worker implementations to the jadice server. This happens in the files **/server-config/jadice-server.options** and

---

11 The numbering must be consecutive and must not clash already existing entries.
12 If jadice server is started in multi-VM-mode, these parameters only take effect on the central instance (compare chapter 4.1.12)
13 See foot note 11
14 See foot note 12

/server-config/jadice-server-local.options. The following entries are possible here:

`-cp` `<Jar-library>`	Adds a single Jar-library to jadice server's class path.
`--classpath` `<Jar-library>`	
`-ld` `<directory>`	All Jar-libraries provided in the given directory are added to jadice server's classpath in alphabetical order.
`--library-dir` `<directory>`	

With that it has to be noted that a line break must be done between the option and the parameter. The effective class path is built in the following way:

1. Entries under -cp / --classpath from jadice-server.options
2. Entries under -cp / --classpath from jadice-server-local.options
3. Entries under -ld / --library-dir from jadice-server.options
4. Entries under -ld / --library-dir from jadice-server-local.options

Additionally the following options are possible:

`-xo` `<configuration file>`	Embeds another configuration file. This file has to correspond to the syntax shown here.
`--extra-options` `<configuration file>`	
`-dCL` `--debug-classpath`	When starting a list of the effective class path and of the classloader-hierarchy is shown.
`-dX` `--debug-using-exceptions`	Eventual exceptions and stack traces which are thrown during the start are displayed.

4.1.6. Configuration LibreOffice

It must be referred to the directory **<LibreOffice-directory>/program** to enable the server's accessing on LibreOffice's programme file. The configuration is located in the file **/server-config/jadice-server-local.options**, e.g.

```
# LibreOffice directory
-cp
C:\Program Files (x86)\LibreOffice 4.0\program\
```

Using MS Windows Vista, Windows 7 and Server 2008 the symbolic file paths must not be used; instead of e.g. „C:\Programme (x86)\" (as in the German version of MS Windows) „C:\Program Files (x86)\" must be indicated.

For the use of LibreOffice 4.0.x it is necessary in addition to embed **<LibreOffice-directory>/URE/java/jurt.jar** in the class path. Apart from this JVM- and LibreOffice have to be binary compatible (x86 / x64).

If the paths are not configured properly, an error will be reported when converting.

On operating systems not belonging to Windows (Linux, Unix, and similar) the package **Xvfb** (X window virtual framebuffer) must be installed so that a headless and thus automated operation of LibreOffice may be performed.

4.1.7. Configuration MS Office

If jadice server is installed on Windows NT, 2000, Server 2003 or an earlier version and if the conversion is to be performed with the MSWord- / MSExcel- /... nodes,

first the „Microsoft Visual C++ 2005 SP1 Redistributable Package (x86)"[15] must be installed.

In MS Office 2007 Service Pack 2 and newer it is possible to export PDFs natively. In MS Office 2007 versions before Service Pack 2 the „2007 Microsoft Office Add-in: Microsoft Save as PDF"[16] must be installed to use the native PDF export.

Additionally in MS Office's Trust Center it has to be set how macros have to be dealt with. Since during the server's operation no user queries are possible only the options „Deactivate all macros without reporting" and „Activate all macros" are sensible (see 3). ActiveX-settings are to be dealt the same way.
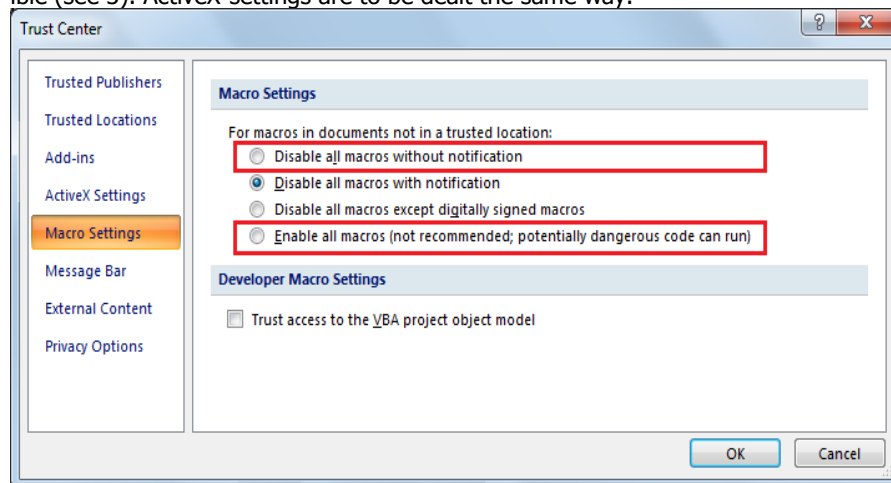
Chart 3: Allowed settings for macros in MS Office

4.1.8. Configuration MS Outlook

In order to avoid security instructions which forbid jadice server's accessing on MS Outlook, the programme „Advanced Security for Outlook"[17] must be installed and configured.

Configuration:

1. Log on the server computer with the user name jadice server runs with.
2. Start a conversion job that effects the MSOutlookNode[18].
3. Confirm the „Advanced Security for Outlook"-dialogue with „Allow Access" and „Always perform (…)"

Besides the option "At programme's ending empty folder "Deleted objects"[19] must be activated and the option "Show warning before elements are finally deleted"[20] must be deactivated.

4.1.9. Configuration MS Project

In order to be able to convert MS project files which have been created by previous versions "Allow loading files with legacy or non default file formats" must be set as permitted in MS Project Trust Center.

---

15 See http://www.microsoft.com/downloads/details.aspx?FamilyID=200b2fd9-ae1a-4a14-984d-389c36f85647&displaylang=en
16 See http://www.microsoft.com/downloads/details.aspx?FamilyID=f1fc413c-6d89-4f15-991b-63b07ba5f2e5&displaylang=en
17 See http://www.mapilab.com/de/outlook/security/
18 com.levigo.jadice.server.msoffice.MSOutlookNode
19 Find under Extras → Options → More
20 Find under Extras → Options → More → Extended Options
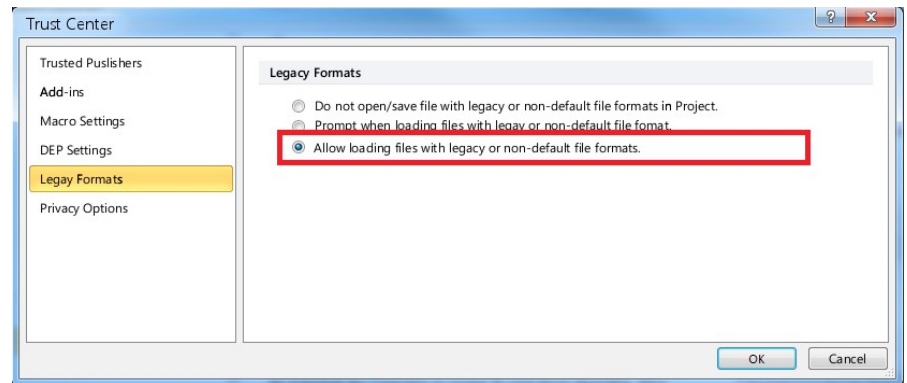
Chart 4: Permission of legacy formats in MS Project 2010

4.1.10. Configuration logging

For the administration of log entries the framework log4j[21] is accessed. It is configured in the file **/server-config/logging/log4j-appenders.xml** respectively **log4j-appenders-mvm.xml** in multi-VM-mode (see chapter 4.1.12). Log reports are written on the console and saved in the file **/log/jadice-server.log** by default. Further possibilities are listed as comments in the configuration file and have only to be activated.

4.1.11. Configuration Ghostscript

In order to use the GhostscriptNode[22], first the following foundations have to be laid:

- Installation of GPL Ghostscript[23], version 8.64 or newer
- Adaptation of the file **/server-config/ghostscript/ghostscript.xml**. Under **<bean id="ghostscript" (...)>** an element with the following contents has to be inserted:
  <property name="executableName" value="*<location of Ghostscript-application file>*" />
  Templates already exist for the locations in which Ghostscript is installed under Windows or Linux by default. For this just the appropriate XML-comment has to be removed.

4.1.12. Configuration Multi-VM-Mode

Due to the embedding of libraries it is possible that they crash the JAVA Virtual Machine in a case of error and thus the complete jadice server.

To continue work on further jobs in this case it is possible to start jadice server on one computer in multiple instances. At this a central instance of jadice server takes over the supervision of all other instances which perform the actual work. If one of these worker instances should not react any more or should have crashed, the central instance terminates the related process and starts automatically a new instance of the jadice server.

In order to start jadice server in this mode, in the file **/server-config/application/server.xml** the part which has been commented out must be changed as follows:

```
<!--  <import resource="single-instance.xml"/> -->
      <import resource="multi-vm-manager.xml"/>
```
Code example

In the file **/server-config/application/multi-vm-manager.xml** the worker instances may be configured in the paragraph <bean id="server" …>:

The number of worker-instances to be started can be adjusted in this way:

---

21 See http://logging.apache.org/log4j/
22 com.levigo.jadice.server.ghostscript.GhostscriptNode
23 See http://www.ghostscript.com/

- Fixed number of worker instances:
  ```
  <property name="fixedVMCount" value="<n>" />
  ```
- *n* worker instances per processor core:
  ```
  <property name="perProcessorVMCount" value="<n>" />
  ```

Beyond this it is possible to provide under <property name="instanceJVMOptions"… > start parameters like e.g. the available memory size of the JAVA VM to be started.

4.1.13. Configuration web service interface

In order to activate the interface by which queries in XML-format may be transmitted to the jadice server, the comment's paragraph **<import resource="webservices.xml"/>** in the file **server-config/application/server.xml** has to be removed.

In this file end points[24] of the web service are published in the paragraph <bean id="web-service-provider" …>. The given entry

```
<entry key=
  "http://${jadice.server.hostname}:9000/jadiceServer">
```
Code example

provides an end point under the computer's name on port 9000.

How to use the web service interface see chapter 6.

4.1.14. Configuration security interface

By using the security interface it can be prevented that clients without authentification acces jadice server and without any limitation its full functional range. In order to activate this interface you have to delete in file **server-config/application/server.xml**[25] the XML comment sign surrounding the element **<import resource="security.xml" />.**

The file **security.xml** contains an exemplary configuration which first has to be adapted to the desired security level. This and how the security-interface may be used then is described in chapter 7

4.2. Client

For the client-sided use the jar files from the directory
     /client-lib (for Java 1.6 and above)
have to be embedded into the class path of the application / developing environment.

4.3. Installation in the developing environment Eclipse

4.3.1. Server

– Create a new JAVA project.
– Add the following jar-files to class path:
  All jar-files from directory /server-lib
  File activemq-all-5.x.x.jar from directory /apache-activemq-5.x.x
  Files spring-###-2.5.5.jar from directory /apache-activemq-5.x.x/lib/optional
– Copy all configuration files from directory /server-conf into source-directory (src).
– Start class JadiceServerControl from jar-file server-core at last.
– The server is now ready for use.

---

24 See javax.xml.ws.Endpoint
25 If jadice server uses the Multi-VM-Mode: server-config/application/multi-vm-instance.xml

4.3.2. Client

– Create a new JAVA project.
– Add all jar-files from directory /client-lib to class path.

5. Application / Functionality

Worker are server-sided implementations which deal with certain tasks being computer intensive and resource consuming. This includes e.g. the generating of large
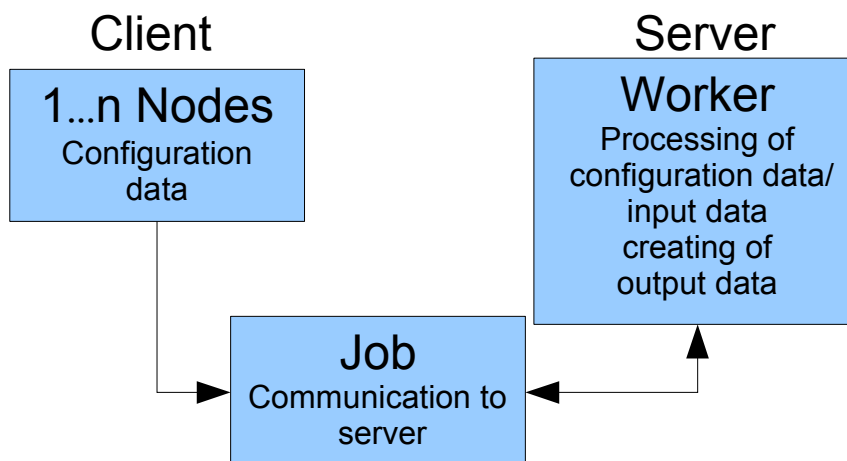
documents, the creating of data streams for displaying / printing / page preview / page selection. Workers are contacted by a client via corresponding nodes and provided with data.

The super class for a node implementation is the class **com.levigo.jadice.server.Node**, for a worker implementation **com.levigo.jadice.server.core.worker.NodeWorker**.

A detailed description of the nodes predefined in jadice server may be taken from the attached Javadoc documentation. How to implement own nodes and workers to expand jadice server's functionality is described in chapter 5.4 using an example.

5.1. Job definition client-sided

On client side a job is created to which one or more node implementations with the required data (configuration / data streams) are given. Then this job contacts the server and sets the nodes in a queue. Due to the asynchronous communication interface the client is not blocked by the server during the operation.

The nodes build a directed, acyclic graph and thus define the work flow (e.g. node 1 loads data and node 2 processes them, node 3 sends the data finally back to the client). How this is to be implemented, is illustrated in chapter 5.3 by means of some examples.

5.2. Job definition server-sided

Jadice server creates a work flow basing on the nodes transmitted by the job. In doing so the nodes which were linked together are processed one after the other by starting the corresponding workers. The created data are passed on to the next worker by StreamBundle[26] objects.

5.3. Application scenarios including code examples

The application scenarios and configuration possibilities for jadice server are multiple and manifold.

That is why only the most common scenarios which may serve as a starting point for own implementations are treated here.

Mostly they are built along this pattern:

---

26 com.levigo.jadice.server.shared.types.StreamBundle

- File received from client
- Server-sided processing
- Submit result to client

But this is just a simplified displaying of the structure. In real applications results are normally not sent back right after their processing, but they are processed in cascaded steps. Even the client is not stringently the source and the destination of data streams, but this may be e.g. a central file, mail or archive server, too.

5.3.1. Create a server job

The server job is created on client side; 1...n nodes may be attached to the job.

Possible configurations of the server job:

- Timeout and similar limits[27], see chapter 5.3.3
- JobListener implementation, see chapter 5.3.2
- Work flow (= acyclic graph defined by linked nodes)

Create job (in example with ActiveMQ as message broker):

```java
public Job createServerJob() {
  // Create job factory with parameters „Server-
  // Url" and „Queue-Name" (compare 4.1.4)
  JMSJobFactory jobFactory = new JMSJobFactory(
    new ActiveMQConnectionFactory("tcp://<Broker_IP>
      :<Broker-Port>"), "<Queue-Name>");
  // create server job
  Job job = jobFactory.createJob();
  return job;
}
```
Code example

Configure and perform job:

```java
// Create job
Job job = createServerJob();
// Set timeout(60 seconds)
job.apply(new TimeLimit(60, TimeUnit.SECONDS));
// Register JobListener (compare 5.3.2)
job.addJobListener(<JobListener-Implementation>);
// Define workflow (see below)
job.attach(<Node-Workflow>);
// Send job to server
job.submit();
```
Code example

5.3.2. Create a JobListener

With the JobListener[28] server job states and server-sided error reports may be processed.

Example of a JobListener implementation MyJobListener:

```java
public class MyJobListener implements JobListener {
 public void stateChanged(Job job, State old,
   State new) {
   dump("stateChanged", job, old, new, null, null);
```

---

27 com.levigo.jadice.server.Limit
28 com.levigo.jadice.server.JobListener

```java
}

public void executionFailed(Job job, Node node,
  String messageId, String reason, Throwable cause) {
  dump("executionFailed", job, node, messageId,
    reason, cause);
}

public void errorOccurred(Job job, Node node,
 String messageId, String message, Throwable cause) {
 dump("errorOccurred", job, node, messageId, message,
   cause);
}

public void warningOccurred(Job job, Node node,
  String messageId, String message, Throwable cause) {
  dump("warningOccurred", job, node, messageId,
    message, cause);
}
public void subPipelineCreated(Job job, Node parent,
  Set<? extends Node> createdNodes) {
  dump("subPipelineCreated", job, parent,
    null, null, null)
}
private void dump(String ctx, Job job, Object
  arg1, Object arg2, Object arg3, Object arg4) {
  System.err.println("Context:    " + ctx);
  System.err.println("Job:        " + job.toString());
  System.err.println("            " + arg1);
  System.err.println("            " + arg2);
  System.err.println("            " + arg3);
  System.err.println("            " + arg4);
}
}
```

Code example

Jadice server is equipped with two implementations of this interface which may be used in the integration:

Class name	Description
TraceListener[29]	Transmits error report via Apache Commons Logging to the Client-Log.
JobListenerAdapter[30]	Empty default implementation of JobListener-Interface. Classes derived from that must only overrride the desired methods.

5.3.3. Configuration of Limits

In order to limit the resource consumption of a job or its nodes it is sensible to use limits. The following limits are possible for this:

Limit type	Description	Job	Node
		Context	
TimeLimit	Maximum processing time	☑	☑
StreamCountLimit	Maximum number of streams provided by a node	☐	☑
StreamSizeLimit	Maximum size of streams provided by a node	☐	☑

---

29 com.levigo.jadice.server.util.TraceListener
30 com.levigo.jadice.server.JobListenerAdapter

PageCountLimit	Maximum size of pages of a generated document	☐	☑[31]
NodeCountLimit	Maximum number of nodes which a job may have. Respects also nodes which may be created dynamically by the server.	☑	☒

Legend:
☑ is directly respected at this location
☒ is not respected at this location
☐ is not respected, but is passed to the nodes (see below)

When defining a limit it may be chosen what is to happen if it is exceeded:

```
TimeLimit tl = new TimeLimit(60, TimeUnit.SECONDS);
tl.setExceedAction(WhenExceedAction.ABORT); // default

NodeCountLimit ncl = new NodeCountLimit(20);
ncl.setExceedAction(WhenExceedAction.WARN);
```

At the action ABORT the complete job is cancelled, at the action WARN a warning is reported to the client.

Since at the clientsided workflow definition possibly not all nodes are known or it is not sensible to allocate each single node with limits, these may also be used on the job. They are passed to the single nodes. At this the following precedence rules apply:

1. Limits with action WARN are passed on in any case.

2. Limits with action ABORT are not passed on to nodes on which already a limit of the same class with the action ABORT has been applied even if this is less restrictive.

3. If limits of the same class are set with Action ABORT both clientsided as well as by the security interface, the more restrictive have priority. Compare chapter 7.1.2

5.3.4. Identification of unknown input data

jadice server offers powerful modules for the recognition of unknown file formats. These are used in the modules to convert unknown files respectively e-mails automatically (see chapter 5.3.10 and 5.3.12).

Beyond this it is also possible to address these modules by the StreamAnalysisNode[32] and to use it for own purposes.

```
// Create server job (see 5.3.1)
Job job = createServerJob();
// Create nodes:
// 1. Data input node
StreamInputNode siNode = new StreamInputNode();
// 2. Analyse node
StreamAnalysisNode saNode = new StreamAnalysisNode();
// 3. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create workflow
job.attach(siNode.
  appendSuccessor(saNode).
  appendSuccessor(soNode));
// Perform job
job.submit();
// Send document data stream
```

---

31 For nodes generating documents which know a page term. Compare javadoc
32 com.levigo.jadice.server.jadice4x.StreamAnalysisNode

```
siNode.addStream(<InputStream>);
// Finish data input
siNode.complete();
// Await server's response³³
for (Stream stream : soNode.getStreamBundle()) {
      // Read meta data.
      StreamDescriptor descr = stream.getDescriptor();
      String mimeType = descr.getMimeType();
}
```
Code example

5.3.5. Extraction of document information

The JadiceDocumentInfoNode[34] implementation sends a document to the server.
The server loads the document and provides document specific information[35] to the client. The format to be analyzed must be supported by the jadice document platform.

The DocumentInfoListener implementation:

```
public class DocumentInfoListener implements
 IDocumentInfoResultListener {
 // Documentinfo created by server-sided worker.
 private IDocumentInfo documentInfo = null;
 private boolean documentInfoReceived = false;

 public void documentInfoRecieved(IDocumentInfo info) {
  // When the worker has finished, the documentinfo
  // is passed here.
  documentInfo = info;
  documentInfoReceived = true;
 }

 public void waitForDocumentInfo() {
  // Blocks until the worker is ready.
  while (!documentInfoReceived) {
   try {
    Thread.sleep(250);
   } catch (Exception e) {
   }
  }
 }

 public IDocumentInfo getDocumentInfo() {
  return documentInfo;
 }
}
```
Code example

Create and perform a job by using the JadiceDocumentInfoNode implementation:

```
// Create server job (see 5.3.1)
Job job = createServerJob();
// Create listener
DocumentInfoListener documentInfoListener =
  new DocumentInfoListener();
```

---

33 The method getStreamBundle() blocks till the server has finished processing. An asynchronous processing is to be realised by using a JobListener implementation (compare chapter 5.3.2).
34 com.levigo.jadice.server.jadice4x.JadiceDocumentInfoNode
35 com.levigo.jadice.server.jadice4x.IDocumentInfo

```
// Create InfoNode, add listener
JadiceDocumentInfoNode infoNode =
  new JadiceDocumentInfoNode();
infoNode.addInfoResultListener(documentInfoListener);
// Create InputNode is the 1st node in the work flow
StreamInputNode siNode = new StreamInputNode();
// Add Infonode as successor.
// On server-side first the InputNode for the
// document's loading is processed, then the InfoNode
// which analyses the loaded document is performed.
siNode.appendSuccessor(infoNode);
// 1. Pass node to job...
job.attach(siNode);
// ...and start job
job.submit();
// Only after the job's starting the data stream may be
// passed to the Inputnode.
siNode.addStream(new FileInputstream("<Filename>"));
// Finish data input
siNode.complete();
// Wait for processing by server (see above)
documentInfoListener.waitForDocumentInfo();
// Get documentInfo and release data
IDocumentInfo documentInfo =
  documentInfoListener.getDocumentInfo();
System.out.println("Format          : " +
  documentInfo.getFormat(0));
System.out.println("Number of pages  : " +
  documentInfo.getPageCount());
System.out.println("Size (Pixel) : " +
  documentInfo.getSize(0).width + "x" +
  documentInfo.getSize(0).height);
System.out.println("Resolution (dpi): " +
  documentInfo.getResolution(0));
```

Code example

5.3.6. Merging of multiple PDF documents

With the PDFMergeNode[36] it is possible to merge multiple PDF documents to a single one.

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
// 1. Data input node
StreamInputNode siNode = new StreamInputNode();
// 2. Merging of input data (1…n to 1)
PDFMergeNode pmNode = new PDFMergeNode();
// 3. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create workflow
job.attach(siNode.
  appendSuccessor(pmNode)
 .appendSuccessor(soNode));
//Perform job
job.submit();
// Send PDF-document-data-stream
siNode.addStream(<InputStream_PDF_1>);
siNode.addStream(<InputStream_PDF_2>);
(…) // further data streams
```

---

36 com.levigo.jadice.server.pdfmerge.PDFMergeNode

```
// Finish data input
siNode.complete();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data.
  InputStream is = stream.getInputStream();
}
```
Code example

5.3.7. Converting to TIFF

Most of the converting processes (e.g. LibreOffice, Shaper) create PDF. However, it is possible to keep on converting the result to TIFF by inserting the JadiceShaper-Node[37].

In the following example the work flow from chapter 5.3.6 is changed; instead of the PDFMergeNode a conversion to TIFF with an added aggregation is attached:

```
(…)
JadiceShaperNode shaperNode = new JadiceShaperNode();
// desired target format
shaperNode.setTargetMimeType („image/tiff");
// merge all incoming streams
shaperNode.setOutputMode(OutputMode.JOINED);
// create workflow, insert Tiff-Converter-Node
job.attach(siNode.
  appendSuccessor(shaperNode).
  appendSuccessor(soNode));
(…)
```
Code example

5.3.8. Permanent anchoring of annotations

In order to display documents and their annotations with standard programmes they have to be anchored as "normal" objects in the source format.

This may also be realised by using the JadiceShaperNode. The necessary association between the document data stream and the annotation data streamings is shown in the following example:

```
// example interface for bundling of document and
annotations
interface DocumentAndAnnotations {
  InputStream getContent();
  List<InputStream> getAnnotations();
}

public void convert(doc DocumentAndAnnotations) {
  Job job = createServerJob(); // (see 5.3.1)

  // Create nodes
  StreamInputNode inputNode = new StreamInputNode();
  JadiceShaperNode shaperNode = new JadiceShaperNode();
  StreamOutputNode outputNode = new StreamOutputNode();

  // Configuration of resulting type (e.g. PDF)
  shaperNode.setTargetMimeType("application/pdf");

  // Associate annotation streams with content
  shaperNode.setOutputMode(
    OutputMode.ASSOCIATED_STREAM);
```

---

37 com.levigo.jadice.server.jadice4x.JadiceShaperNode

```
// Configure the workflow
job.attach(inputNode
  .appendSuccessor(shaperNode)
  .appendSuccessor(outputNode));
job.submit();

// Sending document content
// (here with an explicit MIME type)
final StreamDescriptor contentSD
  = new StreamDescriptor("application/pdf");
inputNode.addStream(doc.getContent(), contentSD);

// Processing of annotations:
for (InputStream annoStream : doc.getAnnotations()) {
  StreamDescriptor annoSD = new StreamDescriptor();
  // Association between document and annotation:
  annoSD.setParent(contentSD);
  // Setting of annotation type (e.g. Filenet P8):
  annoSD.setMimeType(JadiceShaperNode
    .AnnotationMimeTypes.FILENET_P8);
  // Sending of annotation data stream
  inputNode.addStream(annoStream, annoSD);
}
inputNode.complete();
// Processing of result (see 5.3.6)
}
```
Code example

In this configuration there are two settings of particular relevance:

Document and annotation data streamings have to be linked with each other by the StreamDescriptor hierarchy. For this the document's StreamDescriptor must be set as the parent of the annotations' StreamDescriptors.

For the provided MIME types of annotations there are predefined constants in the class JadiceShaperNode which must be set stringently. Further information about annotation formats and their properties you may get from the annotation manual of the jadice document platform[38].

Please note that the document's content is not being analysed. Contents covered by annotations may depending on the data format still be available in the target data stream.

5.3.9. Unpacking of archive files

To reduce the network load files are often compressed. Before the processing they may be unpacked by jadice server. This happens depending on the file format in different node classes:

File format	Node class	Comment
ZIP	com.levigo.jadice.server.archive.UnZIPNode	
RAR	com.levigo.jadice.server.archive.UnRARNode	
GZIP	com.levigo.jadice.server.archive.UnGZIPNode	.tar.gz-files have to pass the GZIP-Node first, then the TAR-Node.
TAR	com.levigo.jadice.server.archive.UnTARNode	

How this looks like for the UnZIPNode, is shown in the following code example:

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
```

---

38 http://support.levigo.de/products/jadice/documentplatform/4.3/documentation/annotation
   -manual.html

```
// 1. Data input node
StreamInputNode siNode = new StreamInputNode();
// 2. Unpacking of ZIP-archives
UnZIPNode unzipNode = new UnZIPNode();
// 3. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create work flow
job.attach(siNode.
  appendSuccessor(unzipNode).
  appendSuccessor(soNode));
// Perform job
job.submit();
// Send document data stream
siNode.addStream(<zipped_Inputstream>);
// Finish data input
siNode.complete();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data(1 stream per file in archive)
  InputStream is = stream.getInputStream();
}
```

Code example

5.3.10. Converting unknown input data in a unified format (PDF)

A unification of documents is particularly in the domain of long time archiving of use. The access on the data source, the automatic data analysis, a target-oriented, dynamic processing and a final archiving in the archive bring the following advantages:

The calling application does not need any knowledge about source files and formats. There is no danger by malign data or documents. Consequently the network transfer is minimised. Due to its structure jadice server makes it possible to control at any time the converting result in a flexible way.

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//  1. Data input node
StreamInputNode siNode = new StreamInputNode();
//  2. Analysis node
DynamicPipelineNode dpNode = new DynamicPipelineNode();
dpNode.setRulesetName("default");
//  3. Merging of input data (1…n to 1)
PDFMergeNode pmNode = new PDFMergeNode();
//  4. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create work flow by nodes' linking
job.attach(siNode.
  appendSuccessor(dpNode).
  appendSuccessor(pmNode).
  appendSuccessor(soNode));
// Perform job
job.submit();
// Send document-data stream
siNode.addStream(<InputStream>);
// Finish data input
siNode.complete();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data
  InputStream is = stream.getInputStream();
}
```

Code example

You can finde the ruleset used in the folder **/server-config/dynamic-pipeline-rules**. In this folder you can also find an XML Schema which helps to costumize these XML base ruleset for own need.

5.3.11. Converting Office-documents to PDF

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//  1. Data input node
StreamInputNode siNode = new StreamInputNode();
//  2. LibreOffice-conversion node[39]
LibreOfficeConversionNode loNode =
  new LibreOfficeConversionNode();
// 3. Merging of input data(1…n to 1)
PDFMergeNode pmNode = new PDFMergeNode();
// 4. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create work flow
job.attach(siNode.
  appendSuccessor(loNode).
  appendSuccessor(pmNode).
  appendSuccessor(soNode));
// Perform job
job.submit();
// Send document data stream
siNode.addStream(is);
// Finish data input
siNode.complete();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data
  InputStream is = stream.getInputStream());
}
```
Code example

Note: Documents in Word2007 format (file extension .docx) must pass the StreamAnalysisNode before being converted by LibreOffice (compare chapter 5.3.4).

5.3.12. Converting e-mails to PDF

When converting e-mails the e-mail is taken directly from the mail server. For this the corresponding access data have to be indicated.

This process is similar to the dynamic conversion (see chapter 5.3.10). The e-mail is analysed, possible attachments like e.g. Office documents, pictures etc. are all converted, merged in a review and attached to the e-mail text.

Archive files are unpacked and their content is embedded in the converting process.

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//  1. Input node, here from server-side a mail
//server is addressed.
JavamailInputNode jiNode = new JavamailInputNode();
// Set mail server specific data
jiNode.setStoreProtocol(<Protocol>); // POP3 or IMAP
jiNode.setHostName(<Server>);
jiNode.setUsername(<User>);
jiNode.setPassword(<Password>);
jiNode.setFolderName(<E-Mail folder>);
jiNode.setImapMessageUID(<E-Mail ID>);
```

---

39 The class path has to be set as described in chapter 4.1.6.

```
//  2. Analysis node with script for e-mail-conversion
ScriptNode scNode = new ScriptNode();
scNode.setScript(new URI("resource:
  email-conversion/EmailConversion.groovy")));
// 3. Merging of input data(1...n to 1) PDFMergeNode
pmNode = new PDFMergeNode();
// 4. Output node
StreamOutputNode soNode = new StreamOutputNode();
// Create work flow
job.attach(jiNode.
  appendSuccessor(scNode).
  appendSuccessor(pmNode).
  appendSuccessor(soNode));
// Perform job
job.submit();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data
  InputStream is = stream.getInputStream();
}
```
Code example

If e-mails are not to be requested by the JavamailInputNode via a IMAP or POP3 ac-
count, but e.g. read as eml-file, additionally a MessageRFC822Node[40] which separ-
ates the e-mail header and the body has to be interposed:

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//  1. Input node, here as file from client.
StreamInputNode siNode = new StreamInputNode();
//  2. Separation of e-mail-header and -body
MessageRFC822Node msgNode = new MessageRFC822Node();
//  3. Analysis node with script for e-mail-conversion
ScriptNode scNode = new ScriptNode();
scNode.setScript(new URI("resource:
  email-conversion/EmailConversion.groovy")));
// Rest see above
```
Code example

E-mails which are available in MS Outlook format (msg-files) may be converted by
TNEFNode[41] without starting MS Outlook into a format supported by jadice server
and by using this configuration:

```
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//  1. Input node, here as file from client.
StreamInputNode siNode = new StreamInputNode();
//  2. Pre-conversion of MSG files
TNEFNode tnefNode = new TNEFNode();
tnefNode.setInputFormat(InputFormat.MSG)
//  3. Analysis node with script for e-mail-conversion
ScriptNode scNode = new ScriptNode();
scNode.setScript(new URI("resource:
  email-conversion/EmailConversion.groovy")));
// Rest see above
```
Code example

Please note that the mail body in msg-files is available as rich text (rtf) and thus in
the standard configuration it is converted to PDF by LibreOffice.

---

40 com.levigo.jadice.server.javamail.MessageRFC822Node
41 com.levigo.jadice.server.javamail.TNEFNode

In the configuration shown above a separator which contains meta data of the corresponding attachment is generated by default for each file attachment. If no separators are desired, they may be deactivated for all file attachments with the following configuration of the ScriptNode:

```
(…)
scNode.getParameters().put
   ("showAttachmentSeparators", false));
(…)
```
Code example

Another configuration possibility regards formatted e-mails. If these were sent both in HTML and plain-text format, the HTML part is converted by default. If the plain-text part is to be converted instead, the following configuration of the ScriptNode has to be done:

```
(…)
scNode.getParameters().put
   ("preferPlainTextBody", true));
(…)
```
Code example

Besides this it is possible to attach this part which is normally not converted as an additional attachment to the e-mail. Thus the formatted e-mail may be displayed both in the HTML and in the plain-text format. The required configuration is:

```
(…)
scNode.getParameters().put
   ("showAllAlternativeBody", true));
(…)
```
Code example

In order to avoid that jadice server downloads images and other files referenced in e-mails from unknown sources, this may be done by the following setting:

```
(…)
scNode.getParameters().put
   ("allowExternalHTTPResolution", false));
(…)
```
Code example

How to handle attachments with a format that has not been recognised or that is not configured to be converted by the jadice server may be controlled by the parameter **unhandledAttachmentAction**:

```
(…)
scNode.getParameters().put
   ("unhandledAttachmentAction", "failure"));
(…)
```
Code example

The following values are accepted herewith:

Value	Significance
warning	A warning is written into the log.
error	An error is written into the log (default value).
failure	The corresponding job breaks off with a failure.

Image files which are referenced in an e-mail but have not been converted are replaced by the following place holders for identification:

Value	Meaning
http://example.com/restricted.jpeg	The image has not been loaded due to the setting „allowExternalHTTPResolution" (see above).
cid://failed.jpeg	The image file could not be loaded.

5.3.13. Controlling of external programmes

The controlling of external programmes is very easily possible by using the ExternalProcessCallNode[42]. Jadice server takes automatically care that incoming and outgoing data streams are automatically converted to temporary files and that these files are deleted after having been prcessedby the external programme.

The only condition is that the programme on the server may be accessed by com-mand line:

```java
Job job = createServerJob(); // (see 5.3.1)
// Create nodes:
//   1. Data input node
StreamInputNode siNode = new StreamInputNode();
//   2. External process
ExternalProcessCallNode epcNode =
  new ExternalProcessCallNode();
// Configuration:
//     Programme name (backslashes must be escaped)
epcNode.setProgramName(
  "C:\\Programme\\MyConverter\\MyConverter.exe");
//     Command line parameters
//     ${infile} and ${outfile} substituted by jadice
server
epcNode.setArguments(
  "-s -a ${infile} /convert=${outfile}");
//     file extensions, if required by programme
epcNode.setInfileExtension(".foo");
epcNode.setOutfileExtension(".pdf");

// 3. Output node
StreamOutputNode soNode = new StreamOutputNode();
// create work flow
job.attach(siNode.
  appendSuccessor(epcNode).
  appendSuccessor(soNode));
job.submit();
// Await server's response
for (Stream stream : soNode.getStreamBundle()) {
  // Read data
  InputStream is = stream.getInputStream();
}
```
Code example

5.4. Implementation of own nodes / workers

In this chapter it will be shown by using a simple example how jadice server may be extended with own nodes and workers in order to realise new processing steps.

For this mainly two steps are necessary: First a node class which exists both on cli-ent and server side has to be implemented (see chapter 5.4.1). Then the corres-ponding worker class has to be implemented (see chapter 5.4.2). This worker class needs to be available only on server side.

5.4.1. Node class

The node class to be newly created must inherit of the abstract super class Node[43]. It must own a parameterless constructor („default constructor").

---

42 com.levigo.jadice.server.external.ExternalProcessCallNode
43 com.levigo.jadice.server.Node

Apart from this the method getWorkerClassName() may be overwritten. As the default return value this method provides as return value the fully qualified class name of the node, in which "node" is replaced by "worker" as well as "worker" is inserted as an additional name space layer (example: com.acme.jadiceserver.ExampleNode.-getWorkerClassName() provides „com.acme.jadiceserver.worker.ExampleWorker").

If you have chosen a different package structure, this method may be overwritten in order to provide the fully qualified class name of the corresponding worker class:

```java
package com.mycompany.jadice.client;
import com.levigo.jadice.server.Node;

public class DemoNode extends Node {
  public String getWorkerClassName() {
    // Class name of worker class from example below
    // default return value would be
    // "com.mycompany.jadice.client.worker.DemoWorker"
    return "com.mycompany.jadice.worker.DemoWorker";
  }
}
```

Code example: Implementation of a node

If it should be possible to transmit parameters to the worker during the runtime, this may be done by further methods in the node implementation. At this you have to bear in mind that all object and static attributes have to implement the interface Serializable[44] since they are serialized and transported by JMS (see chapter 4.1.4)

```java
public String getMyParameter() {
  // Should be set by e.g. setter-method
  return "a Parameter";
}
```

Code example: Node from example above extended with a parameter

The node implemented by oneself has to be embedded in the class path both on client and server side and may be exactly like the nodes shown in chapter 5.3 embedded in own work flows.

5.4.2. Worker class

The worker class in which the converting is performed inherits of the abstract generic super class NodeWorker<N>[45] whereat the type parameter <N> stands for the corresponding node class.

Here the abstract method work() is to be implemented in which the server-sided conversion is performed.

```java
package com.mycompany.jadice.server;

// Here are the most important imports
import com.levigo.jadice.server.core.NodeWorker;
import com.mycompany.jadice.client.DemoNode;

public class DemoWorker extends NodeWorker<DemoNode> {

  protected void work() throws Throwable {
    // Parameter defined in example above
    String myParam = getNode().getMyParameter();

    // Retrieve input data
    for (Stream stream : getInputBundle()) {
      InputStream unprocessedIS =
        stream.getInputStream();
        // Meta data of received data stream
```

---

44 java.io.Serializable
45 com.levigo.jadice.server.core.NodeWorker<N>

```
      StreamDescriptor unprocessedSD =
        stream.getDescriptor();

      // Method which processes data stream
      // (not shown in listing)
      InputStream processedIS =
        process(unprocessedIS, myParam);

      // Meta data of processing data stream
      // unprocessedSD is set as „Parent"
      StreamDescriptor processedSD =
        new StreamDescriptor(unprocessedSD);
      processedSD.setDescription("<Description>");
      processedSD.setMimeType("<MIME Type>");
      processedSD.setFileName("<File name>");

      // Combining of result and meta data
      Stream result =
        new BundledStream(processedIS, processedSD);
      // Pass results
      getOutputBundle().addStream(result);
    }
}
```

Code example: Implementation of a worker

The worker implemented this way must be embedded only in the class path of jadice server and is automatically recalled from the previous chapter when using the corresponding nodes.

6. Web service Interface

In order to offer jadice server's full functionality to clients independently of their implementation language a web service interface has been introduced in version 4.2.0.0.
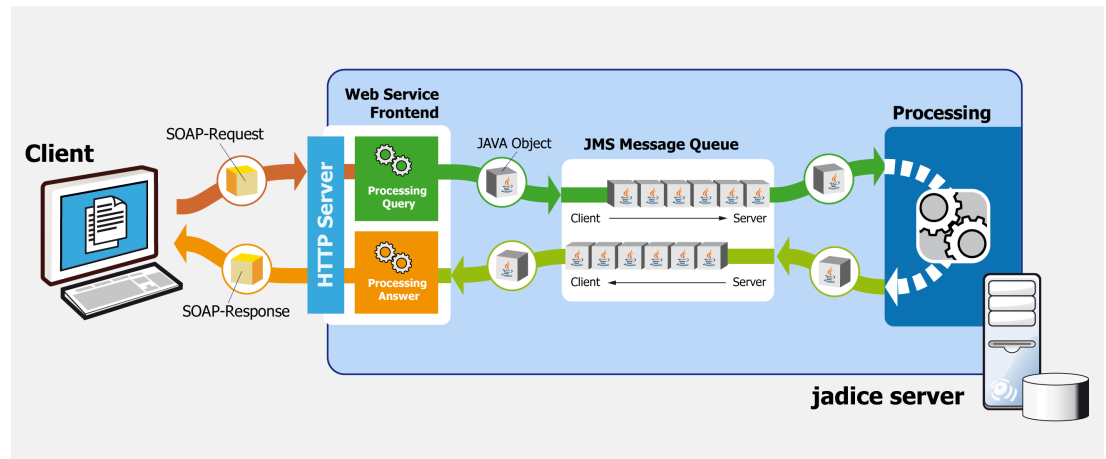
Chart 5: Schematic structure of the web service interface

The communication with jadice server – respectively with the web service front end – takes place by HTTP protocol via SOAP report in the XML-format. The transmission of files which are to be converted or sent back to the client as a result is realised as a MTOM-Attachment[46].

For developing and debugging of SOAP queries we recommend soapUI[47].

6.1. Structure of a SOAP-message

After the web service interface has been activated as described in chapter 4.1.13, the interface's formal description may be downloaded in the WSDL format[48] under http://<url>?wsdl (e. g. with a configuration like in the chapter above: http://localhost:9000/jadiceServer?wsdl). Thus code may be generated in many web service frameworks in order to address jadice server's web service. Within a SOAP request jadice server may be addressed in two different ways

- The workflow is preconfigured by means of a template which has been saved on server side before.
- The workflow is defined during runtime within the SOAP request.

The two possibilities are explained in the two following chapters.

6.1.1. Request by means of a template

It is possible to store a XML-coded job description on server side in such a way that a client may refer to it during a SOAP request and the job does not have to be configured during the runtime.

This request's structure is to be explained by using the following example:

```
<soapenv:Envelope xmlns:soapenv=
      "http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ws="http://ws.server.jadice.levigo.com/">
 <soapenv:Header/>
 <soapenv:Body>
  <ws:run>
```

---

46 See http://www.w3.org/TR/soap12-mtom/
47 See http://www.soapui.org/
48 See http://www.w3.org/TR/wsdl20-primer/

```xml
  <job templateLocation=
    "resource:/jobtemplates/x2pdf.xml">
   <property name="dp.rulesetName">default</property>
   <property name="dp.targetMimeType">
     application/pdf</property>
   <property name="dp.timeout">8000</property>
   <stream mimeType="unknown/*" uuid="123456789"
     nodeId="node0">
    <documentData>BASE64_encoded_data</documentData>
   </stream>
  </job>
 </ws:run>
 </soapenv:Body>
</soapenv:Envelope>
```
Code example of a SOAP-request with job template

Beside the elements for header and body determined by the SOAP standard there is the specific element <run> which addresses the method run offered by the web service.

Therein a job (see chapter 3) is defined which is predefined by a template (see chapter 6.3). In the attribute **templateLocation** the location is given where the respective template is found on server side. If variables are defined in the template, they may be configured by **property** elements respectively their default value may be overwritten. The attribute **messageID** is optional. It may be freely allocated by the client and eventually it is assumed in the server's response.

Data streams to be processed are referred to in the SOAP request by **stream** elements. Informations about a unique ID (**uuid**) and the MIME type are optional. If the MIME type is not known, but should be indicated, you have to indicate there „unknown/*".

If multiple StreamInputNodes[49] are defined in the template file, it has to be distinctly allocated which data stream is sent to which StreamInputNode. This is performed by the attribute **nodeId**. It refers to the ID given to the StreamInputNode within the template (attribute **id**).

The actual data follow either directly in the tag **documentData** as base64-encoded String or in a multipart/related container which owns the CID indicated here (content ID) if the MTOM feature is enabled.

6.1.2. Job definition within the SOAP request

If the client is not to use a job configuration predefined on server-side, it is possible to embed it in the SOAP request. The format is the same like within a separate job-template (compare chapter 6.3). Instead of the root element **job** the definition is embedded in the SOAP request as **configuration** element.

This is explained in the following example:

```xml
<soapenv:Envelope xmlns:soapenv=
      "http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ws="http://ws.server.jadice.levigo.com/">
 <soapenv:Header/>
 <soapenv:Body>
  <ws:run>
   <job messageID="4711">
    <configuration>
     <nodes>
      <node class=
      "com.levigo.jadice.server.nodes.StreamInputNode"
      id="input1" />
      <node class=
      "com.levigo.jadice.server.nodes.StreamInputNode"
      id="input2" />
```

---

49 com.levigo.jadice.server.nodes.StreamInputNode

```xml
      <node class=
    "com.levigo.jadice.server.nodes.DemultiplexerNode"
     id="demux" />
     <node class=
     "com.levigo.jadice.server.nodes.StreamOutputNode"
     id="out" />
    </nodes>
    <connections>
     <connect from="input1" to="demux" />
     <connect from="input2" to="demux" />
     <connect from="demux" to="out" />
    </connections>
   </configuration>
   <stream nodeId="input1">
    <documentData>BASE64_encoded_data</documentData>
   </stream>
   <stream nodeId="input2">
    <documentData>BASE64_encoded_data</documentData>
   </stream>
  </job>
 </ws:run>
</soapenv:Body>
</soapenv:Envelope>
```
Code example of a SOAP request with embedded job definition

In this example two StreamInputNodes are connected by a DemultiplexerNode[50] and the input data are returned unchanged to the client.

The nodes' definition and which work flow graph they build is described within the **configuration** block.

Beyond this you can see here how it is possible to bind certain input streams to a StreamInputNode: The first document is bound to the first StreamInputNode (nodeId input1), the second document is bound to the second StreamInputNode (nodeId input2).

6.2. Structure of a SOAP response

The structure of a response which is sent to a client in answering a request is also specified in the WSDL mentioned above.

A possible response may look like this:

```xml
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <ns2:runResponse xmlns:ns2=
     "http://ws.server.jadice.levigo.com/">
   <return>
    <stream>
     <documentData>BASE64_encoded_data</documentData>
    </stream>
    <status>COMPLETED</status>
   </return>
  </ns2:runResponse>
 </soap:Body>
</soap:Envelope>
```
Code example of a SOAP response

Beside a (eventually empty) set of result streams which are each referenced by a unique ID in a multipart/related container there is a status report. The following values are possible:

---

50 com.levigo.jadice.server.nodes.DemultiplexerNode

Value	Significance
COMPLETED	Job has been performed.
FAILED	Job could not be performed.

In both cases the return element may contain a set of log-entry-elements which contain indications about the process failure or log messages which have occurred during the processing (compare chapter 5.3.2 „Create a JobListener"). The following example shows the error report displayed when a non-existing job-template file is referenced:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <ns2:runResponse xmlns:ns2=
      "http://ws.server.jadice.levigo.com/">
    <return messageID="12345">
     <log-entry timeStamp="31.12.2009 22:33:44">
      <level>ERROR</level>
      <id>JS.WEBSERVICE-EXCEPTION</id>
      <message>java.io.FileNotFoundException: Could not
        locate resource: does_not_exist.xml</message>
     </log-entry>
     <status>FAILED</status>
    </return>
   </ns2:runResponse>
 </soap:Body>
</soap:Envelope>
```
Code example of an error report

6.3. Definition of job-templates

Due to the definition of job-templates it has been made possible that clients do not have to know any longer jadice server's internal steps which are necessary for a conversion. These are provided for the web service interface on a central location. Thus the client has to know only the web service method „**run**" and the location of the template to be performed (which are normally in the sub-folder server-config/).

The XSD definition for these templates is provided in the folder server-config/job-templates.

An example how such a template may look like is provided in the distributed template x2pdf.xml which similar to the example from chapter 5.3.10 identifies unknown input data and converts them into PDF format:

```
<job
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="jobtemplate.xsd">
 <properties>
  <property name="PROPERTY_KEY_COMMUNICATION_TIMEOUT">
   ${communication_timeout:22000}
  </property>
 </properties>
 <nodes>
  <node class="com.levigo.jadice.server.
    nodes.StreamInputNode" id="node0">
   <property name="timeout">${timeout:6000}</property>
  </node>
  <node class="com.levigo.jadice.server.
    nodes.DynamicPipelineNode" id="node1">
   <property name="rulesetName">
    ${rulesetName:default}</property>
   <property name="targetMimeType">
```

```
    ${targetMimeType:application/pdf}</property>
  <property name="timeout">${timeout:6000}</property>
 </node>
 <node class="com.levigo.jadice.server.
   nodes.StreamOutputNode" id="node2">
  <property name="timeout">${timeout:6000}</property>
 </node>
</nodes>
<connections>
 <connect from="node0" to="node1" />
 <connect from="node1" to="node2" />
</connections>
</job>
```
Code example of a job-template

As you can see templates are structured in three blocks:

- Properties concerning the whole job (timeout, etc.)
- Definition of single nodes (element <node>) and their properties
- Linking of nodes to a work flow

Since the single nodes correspond to the conventions for Java Beans the respective properties may be simply set over their names. Apart from this it is possible to make them variable as shown in the example above. This happens according to the following pattern:

${<identifier>}** or **${<identifier>:<default value>}

whereas the identifier has to start obligatorily with a letter and as further glyphs it may have letters, numbers, "_" (underscore), "-" (hyphen) and "." (dot).

Due to this identifier these values may be set respectively overwritten as property (**Element <property name="identifier">value</...>**) when calling SOAP. If variables without a default value are not set, this results in the following failure when calling:

„**com.thoughtworks.xstream.converters.ConversionException: Pattern refers to undefined variable <identifier> for which there is no default**"

The single node elements have to stick to an ID which is unique for the respective template. This ID links the node elements in the <connections> block to a work flow.

If data are to be transmitted from the client to the server by the SOAP call belonging to this template, it is necessary to define at least one StreamInputNode. If multiple StreamInputNodes are defined, so the single stream elements in the SOAP call have to reference the corresponding node by the attribute **nodeId**. This does not apply if there is exactly one StreamInputNode.

Data streams resulting from StreamOutputNodes as returned as MTOM attachments in the SOAP response to the client. Here it is also possible to define multiple StreamOutputNodes. At this the order in which the StreamOutputNodes are requested to attach their data streams to the SOAP response is arbitrary.

To embed job-templates in a SOAP-request (see chapter 6.1.2) the root-element job must be removed; the content is instead attached to the element **configuration** in the SOAP request.

6.4. Generation of web service clients

Since the web service interface is clearly defined by the WSDL, web service libraries which are freely available may process this definition and generate proxy-classes that encapsulate the required SOAP requests and thus enable an efficient development of client applications. In this chapter this is shown with the help of Sun's reference implementation of JAX-WS and the library Apache Axis2.

6.4.1. JAX-WS reference implementation

In the distribution of the Java Development Kit (JDK) version 1.6 there is the command line tool **wsimport** which may be used for generating proxy-classes. If jadice server's web service has been activated as described in chapter 4.1.13, the required client classes are created with the following call:

```
<jdk1.6.0>\bin\wsimport
  –keep
  http://localhost:9000/jadiceServer?wsdl
```

The switch "-keep" causes that not only that the classes but also their source texts are saved. For a further development it is recommended to proceed with them. Chart 6 shows the generated classes which may be embedded in the developing environment.

```
com.levigo.jadice.server.ws
    JadiceServerJobInvoker.java
    JobConfiguration.java
    JobInvocationService.java
    JobResult.java
    Level.java
    Log.java
    ObjectFactory.java
    package-info.java
    Property.java
    ResultStatus.java
    Run.java
    RunResponse.java
    Stream.java
```
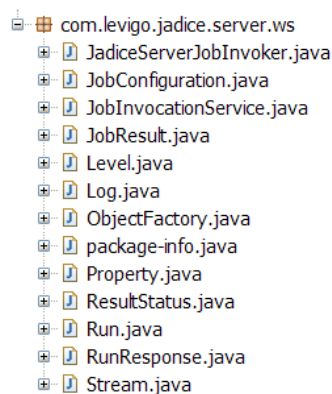
Chart 6: Classes generated by JAX-WS

Starting points for a client application are the classes **JadiceServerJobInvoker** and **JobInvocationService** by which the SOAP interface is accessed and the class **JobConfiguration** by which the call is configured. A minimal implementation may look like this:

```java
// Abstraction of SOAP interface:
JadiceServerJobInvoker invoker =
  new JadiceServerJobInvoker();
JobInvocationService service =
    invoker.getJobInvocationServiceImplPort();

// Job configuration
JobConfiguration job = new JobConfiguration();
job.setTemplateLocation(
  "resource:/jobtemplates/x2pdf.xml");

// Optional: Set a property (e.g. timeout)
Property timeout = new Property();
timeout.setName("timeout");
timeout.setValue("20000");
job.getProperty().add(timeout);

// Attach input data
// (only if template possesses a StreamInputNode)
Stream inputStream = new Stream();
inputStream.setDocumentData(…);   // Byte-Array
job.getStream().add(inputStream);

// Offset SOAP request (method blocks)
JobResult result = service.run(job);
```

```
// request result-status
ResultStatus status = result.getStatus();

for (Log log : result.getLogEntry()) {
  // Evaluate log input …
}

for (Stream stream : result.getStream()) {
  // Result as Byte-Array
  byte[] data = stream.getDocumentData();
}
```
Code example: Implementation of SOAP-client with JAX-WS reference implementation

6.4.2. Apache Axis2

Apache Axis2 is available under **http://ws.apache.org/axis2/** and is under Apache Licence. If jadice server's web service has been activated as described in chapter 4.1.13, the required client classes are created with the following call:

```
<AXIS2_HOME>\bin\wsdl2java
  -o generatedCode
  -p com.levigo.jadice.server.ws.client.axis2.stub
  -d jaxbri
  -uri http://localhost:9000/jadiceServer?wsdl
```

Using the switch "-o" for the output directory and "-p" for the package name to be used are optional. Switch "-d" determines which data binding is to be used for the conversion to / from XML. The Apache Axis Data Binding (ADB) is used by default. However, in the current version it has got problems with the deserialisation of SOAP/MTOM attachments, so that the JAX-B reference implementation (jaxbri) should be used instead.

Starting points for a client application are the classes **JadiceServerJobInvoker-Stub** and **Run** by which the SOAP interface is accessed and the class **JobConfig-uration** by which the call is configured. A minimal implementation may look like this:

```
// Abstraction of SOAP interface
JadiceServerJobInvokerStub invoker =
 new JadiceServerJobInvokerStub();
Run run = new Run();

// Configuration of request
run.setJob(…); // Typ JobConfiguration (see above)

// Offset SOAP request (method blocks)
RunResponse response = invoker.run(run);

// Fetch result object
JobResult jobResult = response.getReturn();

// Processing of result compare above
```
Code example: Implementation of a SOAP client with Apache Axis2

7. Security-Interface

Using the security-interface it is possible to allocate a static configuration in jadice server by which the client access may be restricted according to its role. The following restrictions are possible depending on the permitted role:

- Forbid the complete access on nodes
- Forbid access on nodes depending on the set values
- Force job limits
- Force node limits (optionally in dependence to set values)

The adaptations necessary for this are described in the following subchapters.

7.1. Configuration

The configuration takes place in file **server-config/application/security.xml** which first has to be activated in file **server.xml**.

7.1.1. Authentication

For authentication and permitting of roles it is relied on the framework Spring Security[51]. The following chart is showing how jadice server and Spring Security interact:
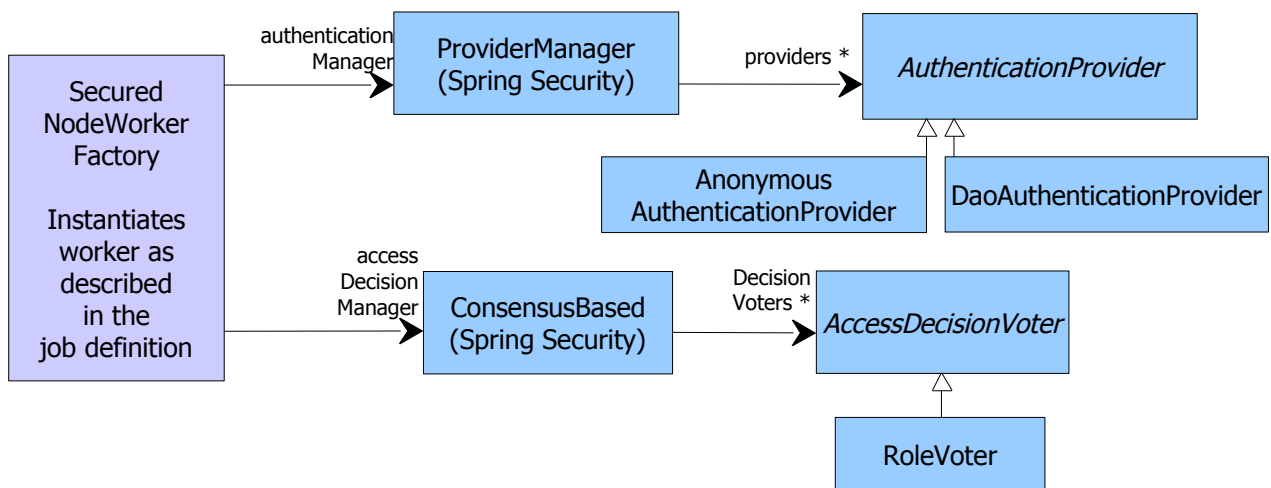
Chart 7: Relation between jadice server and Spring Security

In order to verify the authentication jadice server accesses on the ProviderManager[52] when creating workers. This ProviderManager requests the indicated AuthentificationProviders[53], if they accept the credentials provided by the client. In the exemplary configuration these credentials are on the one hand the DaoAuthenticationProvider which contains hard coded accounts in the XML-configuration. Apart from this an AnonymousAuthenticationProvider is preconfigured which allows clients accessing jadice server without credentials the role „ROLE_ANONYMOUS".

At this point further AuthentificationProvider may be configured which e.g. undertake an authentication against a LDAP-database. You can find a detailed description in Spring Security's documentation. There you can also find further jar-files for download which may be necessary for this. These files can be copied in the folder <jadice-server>/server-lib.

---

51 http://static.springsource.org/spring-security/site/index.html
52 org.springframework.security.authentication.ProviderManager
53 org.springframework.security.authentication.AuthenticationProvider

If an authentication could be performed successfully, subsequently (if nodes under-lying a restriction are to be called) the AccessDecisionManager[54] is requested, if this authentication meets the restriction. This is done by the RoleVoter by checking the allowed role(s).

7.1.2. Restrictions

The access on jadice server can be limited in different directions. This happens as a rule either by AccessRules which limit the access on nodes or by LimitRules which force limits for jobs or nodes.
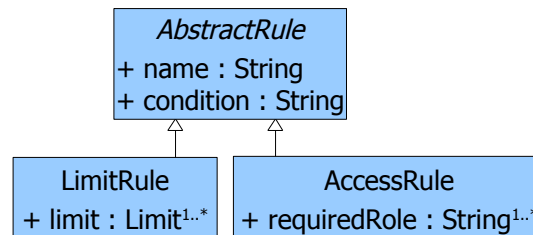

Chart 8: Class hierarchy of security rules
(xml namespace http://www.levigo.com/jadice-server/schema/security)

The condition if a rule applies is expressed in the Spring Expression Language (SPEL[55]). Apart from the in SPEL predefined logic operators **and**, **or**, **!** (not) or the relational operators >,<,<=,>=, == and != there are various special operators for the evaluation of nodes and the values (properties) set there:

Operator	Description
`access('class name node')`	Evaluates to true, if a node of the given type is provided.
`call('class name node')`	
`property('class name node', 'property name')`	Evaluates to the value which a node of the given type has got as indicated property. This may be also applied in a nested way on properties of properties (see example below).
`value('class name node', 'property name')`	

The following ways are possible to combine these operators:

- rules using access and call are allowed to match on different node types:
  z. B. `access('com.levigo.jadice.server.nodes.StreamInputNode') or access( 'com.levigo.jadice.server.nodes.StreamOutputNode')`
  A rule with this condition applies for all nodes of type StreamInputNode or StreamOutputNode
- Rules using property or value may only match on one node type:
  z. B. `value('com.levigo.jadice.server.nodes.URLInputNode', 'URLs[0].host') != 'localhost' or value('com.levigo.-jadice.server.nodes.URLInputNode', 'URLs[0].protocol') != 'http'`
  A rule with this condition applies only for URLInputNodes whose first indic-ated URL does not start with „http://localhost".

For rules which are to take effect depending on the allowed roles there are the op-erators **isAnonymous()** and **hasRole('*role name*').** They may also be mixed with other operators.

e.g. `access('com.levigo.jadice.server.nodes.URLOutputNode') and !hasRole('ROLE_SUPER')`

A rule with this condition takes effect, if a client not holding the role „ROLE_SUPER" tries to access the URLOutputNode.

---

54 org.springframework.security.access.vote.AbstractAccessDecisionManager

55 http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/expres-sions.html

Using these conditions rules of the type AccessRule and LimitRule can be built. The XML pattern definition which these rules have to follow in the configuration file can be found within the jar-file <jadice-server>/server-lib/server-core-<Version>.jar under com/levigo/jadice/server/core/security/jadice-server-security-4.3.xsd.

Rules of the type **AccessRule** contain one or more elements with the name "requiredRole" defining which role is required if the indicated condition applies. These rules are included in the XML configuration with the entry <property name="nodeAccessRules">.

Rules of the type **LimitRule** contain one or more references on limits which are used, if the indicated condition applies. These limits are declared in the XML-file as normal Spring-Beans which have to possess a class derived from Limit[56] and an ID. If limits of the same class are set with action ABORT both client-sided and by the security interface, the more restrictive limits have priority.

Such rules whose limits are to be applied only on particular nodes can be inserted in the paragraph <property name="nodeLimitRules">. Such rules whose limits are to be used on the complete job can be inserted in the paragraph <property name="jobLimitRules">; but it should be noted that the operators access, call, property and value cannot be used in the condition.

7.2. Client-sided use

The client-sided use is limited to the setting[57] of access data (credentials[58]) in the job:

```
Job job = ...;
job.setServerCredentials(
  new Credentials("myUsername", "myPassword"));
// Alternatively:
// Simultaneous authentication towards the MOM:
job.setCredentials(
  new Credentials("myUsername", "myPassword"));
```

Code example: Client-sided authentication

If the authentication cannot be verified by the server, the job fails with this error report (see JobListener#executionFailed):

messageID	JS.SECURITY-BAD_CREDENTIALS
reason	You could not be logged in with the user name '*unknownUsername*'.
cause	org.springframework.security.authentication.BadCredentialsException

If the client does not hold the required role in order to be allowed to access a node or to allocate it with particular values, the job fails with this error report:

messageID	JS.SECURITY-INSUFFICIENT_PRIVILEGES
reason	You do not possess the required authorisation [*required role(s)*] for this action: *name of responsible rule*
cause	org.springframework.security.access.AccessDeniedException

---

56 com.levigo.jadice.server.Limit
57 Please note that the credentials – depending on the used MOM and its configuration – are possibly transmitted in plain text.
58 com.levigo.jadice.server.Credentials

8. Monitoring

Supported by the Java Management Extensions (JMX) it is possible to observe jadice server during its operation and to change some settings during the runtime. Important core components are interpreted here as Managed Beans (MBeans) so that these may be strictly observed.

In order to activate the JMX interface the following entries have to be added in the file **<jadice-server>/wrapper/wrapper.log**:

```
wrapper.java.additional.1⁵⁹=-Dcom.sun.management.jmxremote.port=61619
wrapper.java.additional.2=-Dcom.sun.management.jmxremote.authenticate=false
wrapper.java.additional.3=-Dcom.sun.management.jmxremote.ssl=false
```

Please note that in this example no authentication is prescribed so that even unauthorised users might access jadice server by this interface and thus influence the working process. In the Java SE Monitoring and Management Guide[60] it is described how the authentication may be activated.

Connect yourself with jadice server by using the tools **JConsole** or **Java VisualVM** provided by Sun's Java Runtime Environment. Important components in branch „com.levigo.jadice.server" are (see chart 9):
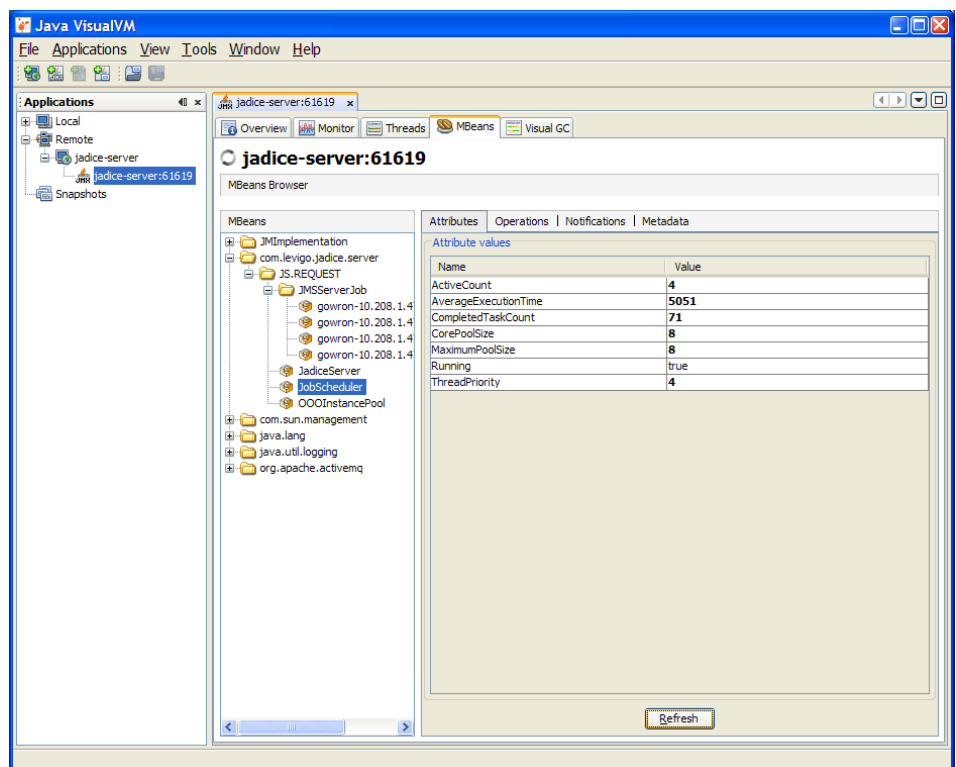
Chart 9: Mbeans-view of a running jadice server instance

- JobScheduler:
  Statistic data for running and already performed jobs may be displayed here.
- Pools for MS Office and LibreOffice:
  Apart from the displaying of presently active Office-instances the pools may be emptied and thus the office-instances may be shut down. If in the

---

59 The numbering must be consecutive and must not clash already existing entries.
60 See http://java.sun.com/javase/6/docs/technotes/guides/management/toc.html

following jobs needing Office-instances are started, new Office-instances are started and the respective pools are refilled. Besides the number of maximally started instances may be changed[61].

– Under JMSServerJob each presently running job is shown. These jobs may be cancelled by the operation "abort". Besides it is possible to query statistic data about the nodes in use, the corresponding threads and for how long the single jobs have already been running.

---

61 This specification is not persisted. When restarting jadice server the value specified in the configuration files is loaded.

9. Troubleshooting

In this chapter some typical failures and errors are shown which may occur during the operation of jadice server.

- Error when submitting a job
  - Error report in client log:

```
com.levigo.jadice.server.JobException:
  Failure during server-side initialization on JADICE-SERVER-JS.REQUEST
   at
com.levigo.jadice.server.client.jms.JMSJobController.performHandshake(JMSJobController.java:579)
   at com.levigo.jadice.server.client.jms.JMSJobController.submit(JMSJobController.java:365)
   at com.levigo.jadice.server.client.DefaultJobInternalImpl.submit(DefaultJobInternalImpl.java:74)
```

  - Error report in server-log:

```
WARN  [; core.ThreadPoolJobScheduler; JadiceServer job scheduler master thread]:
  Not executing job due to expired request message: (…)
```

  - Typical cause:
    The times of client and server differ.
  - Solution:
    Check the times of client and server and re-synchronize them if necessary.
- Error when converting via MS Office
  - Error report in server-log:

```
Exception in thread "main" java.Lang.UnsatisfiedLinkError: (…)\msoffice-lib\jacob-1.14M7-
x86.dll: This application could not been started, because the application configuration is
not correct. To solve the problem please reinstall the application.
```

  - Cause
    Missing C++-Runtime
  - Solution
    Installation of „Microsoft Visual C++ 2005 SP1 Redistributable Package (x86)", see chapter 4.1.7
- Error when converting via MS Office
  - Error report in server-log:

```
Processing failed: MSWordNode/MSWordWorker:
JS.SERVER-WORKER_FAILED: Processing for Node MSWordNode failed because of
com.jacob.com.ComFailException: Invoke of: exportAsFixedFormat
Source: Microsoft Word
Description: Error when exporting because this feature is not installed.
   at com.jacob.com.Dispatch.invokev(Native Method)
   at com.jacob.com.Dispatch.invokev(Dispatch.java:858)
   at com.jacob.com.Dispatch.callN(Dispatch.java:455)
   at com.levigo.jadice.server.msoffice.MSWordConverter.convert(MSWordConverter.java:103)
   at com.levigo.jadice.server.msoffice.CommandReceiver.run(CommandReceiver.java:110)
   at com.levigo.jadice.server.msoffice.MSWordConverter.main(MSWordConverter.java:31)
```

  - Cause
    No native PDF-export installed in MS Office 2007
  - Solution
    Installation of „2007 Microsoft Office Add-in: Microsoft Save as PDF", see chapter4.1.7
- Error when converting via MS Office
  - environment
    MS Office 2007 or newer, Windows Server 2008R2, installation of jadice server as a service
  - Error report in server-log:
    - for MSWordNode:

```
java.lang.Exception: com.jacob.com.ComFailException: VariantChangeType failed
   at com.jacob.com.Variant.toVariantDispatch(Native Method)
   at com.jacob.com.Variant.toDispatch(Variant.java:1976)
   at com.levigo.jadice.server.msoffice.MSWordConverter.convert(MSWordConverter.java:91)
   at com.levigo.jadice.server.msoffice.CommandReceiver.run(CommandReceiver.java:115)
   at com.levigo.jadice.server.msoffice.MSWordConverter.main(MSWordConverter.java:35)
```

```
(…)
```

<center>for MSExcelNode:</center>

```
Incoming Exception from Converter
java.lang.Exception: com.jacob.com.ComFailException: Invoke of: Open
Source: Microsoft Office Excel
Description: Microsoft Office Excel cannot access the file '(...).xls'. There are several
possible reasons:

• The file name or path does not exist.
• The file is being used by another program.
• The workbook you are trying to save has the same name as a currently opened workbook.

   at com.jacob.com.Dispatch.invokev(Native Method)
   at com.jacob.com.Dispatch.invokev(Dispatch.java:858)
   at com.jacob.com.Dispatch.callN(Dispatch.java:455)
   at com.levigo.jadice.server.msoffice.MSExcelConverter.convert(MSExcelConverter.java:79)
(…)
```

- ◦ Cause
  MS Office tries to access a not existing folder in the system directory.
- ◦ Solution
  Creating of an empty folder:
  C:\Windows\SysWOW64\config\systemprofile\Desktop (for x64)
  C:\Windows\System32\config\systemprofile\Desktop (for x86)
- ◦ see also
  MS Office Developer Center[62]

- • Error when converting via MS Office

  - ◦ Error report in server-log:
    ```
    Word cannot start converter mswrd632.
    ```
  - ◦ Cause
    Error in MS Word configuration

- • Solution:
  Cancel registration of converter mswrd632 as described in the Microsoft Knowledge Base:
  http://support.microsoft.com/kb/973904

- • Dialogue when converting via MS Outlook

  - ◦ displayed dialogue

  - ◦ Cause
    MS Outlook's security regulations forbid access by jadice server
  - ◦ Solution
    Installation of „Advanced Security for Outlook", see chapter 4.1.8

- • Error when converting via MS Project

  - ◦ Error report in server-log

```
WARN  [ MSProjectConverter] Incoming exception from converter
java.lang.Exception: com.jacob.com.ComFailException: Invoke of: FileOpenEx
Source: Microsoft Project
Description: Unexpected method error.
   at com.jacob.com.Dispatch.invokev(Native Method)
   at com.jacob.com.Dispatch.invokev(Dispatch.java:858)
   at com.jacob.com.Dispatch.callN(Dispatch.java:455)
```

---

62 http://social.msdn.microsoft.com/Forums/en-US/innovateonoffice/thread/b81a3c4e-62db-488b-af06-44421818ef91?prof=required

```
    at com.levigo.jadice.server.msoffice.MSProjectConverter.
        convert(MSProjectConverter.java:73)
(…)
```

- Cause
  MS Project not configured correctly

  Solution
  Allow processing of legacy format, see chapter 4.1.9

- Error when converting via LibreOffice

  - Error report in server-log

```
INFO  [LibreOfficeInstancePool] Creating an instance
ERROR [ManagedLibreOfficeInstance] Could not bootstrap
com.sun.star.comp.helper.BootstrapException: no office executable found!
    at com.levigo.jadice.server.libreoffice.server.ManagedLibreOfficeInstance.
        launchOOOProcess(ManagedLibreOfficeInstance.java:102)
    at com.levigo.jadice.server.libreoffice.server.ManagedLibreOfficeInstance.
        <init>(ManagedLibreOfficeInstance.java:86)
(…)
```

- Cause
  LibreOffice not configured correctly

- Solution
  Adapt file jadice-server-local.options, see chapter 4.1.6

- Dialogue during PDF-export via LibreOffice
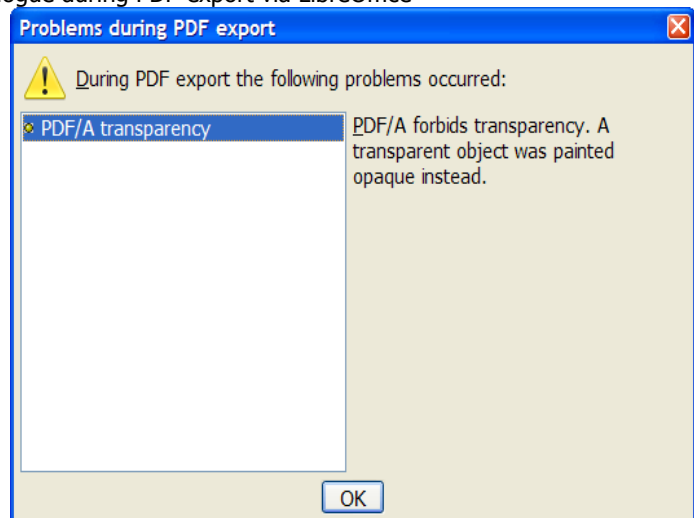
- displayed dialogue

- Cause
  This dialogue is normal when jadice server runs on the same com-
  puter while the client development takes place in parallel and another
  instance of LibreOffice is open in windows.

- Solution
  Close all instances of LibreOffice (soffice.exe / bin in task manager)
  and of LibreOffice quickstarter. At a new conversion LibreOffice is star-
  ted headlessly; the dialogue does not occur any longer.

- Converting with nodes which are not contained in the product by default.

  - Error report in server-log

```
javax.jms.JMSException: Failed to build body from bytes.
    Reason: java.io.IOException: <Node-Klassenname>
    at org.apache.activemq.util.JMSExceptionSupport.create(JMSExceptionSupport.java:35)
    at org.apache.activemq.command.ActiveMQObjectMessage.
        getObject(ActiveMQObjectMessage.java:183)
    at com.levigo.jadice.server.core.JMSServerJob.<init>(JMSServerJob.java:267)
    at com.levigo.jadice.server.core.ThreadPoolJobScheduler$SchedulerThread.
        handleMessage(ThreadPoolJobScheduler.java:203)
    at com.levigo.jadice.server.core.ThreadPoolJobScheduler$SchedulerThread.
        run(ThreadPoolJobScheduler.java:122)
```

```
Caused by: java.io.IOException:
(…)
```

- ○ Cause
  The job created by the client references a node class which is not provided in the class path of the server.
- ○ Solution
  Check the server's class path and add the missing library.

- • JMS Exception when submitting a job
  - ○ possible error reports in client-log

```
javax.jms.JMSException: Failed to build body from bytes.
   Reason: java.io.IOException:
   com.levigo.jadice.server.shared.jms.ServerMessage$InitializationResponse

javax.jms.JMSException: Failed to build body from bytes.
   Reason: java.io.InvalidClassException:
   com.levigo.util.internal.log.qualified.ResolvedQualifiedLogEvent;
   class invalid for deserialization
```
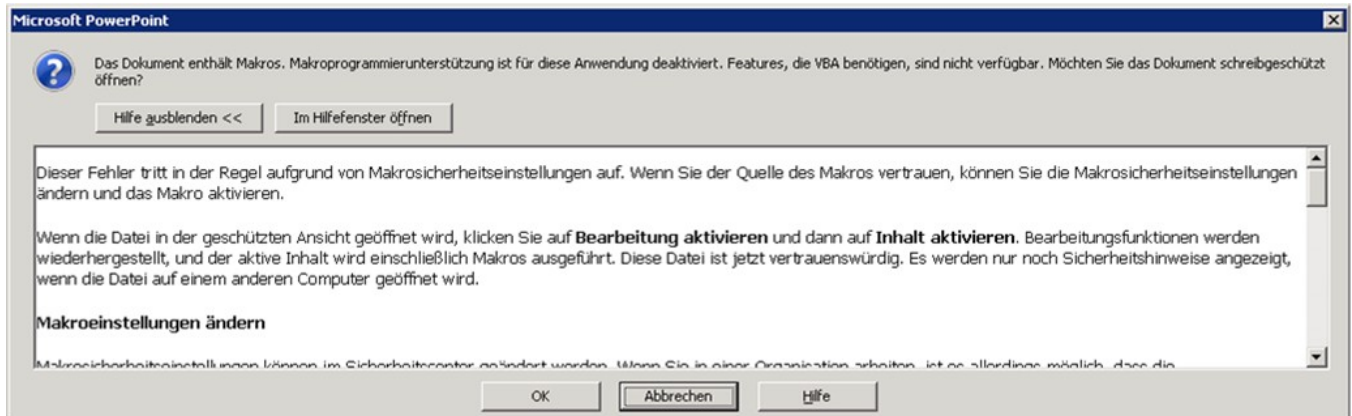
- ○ possible cause
  There is more than one JMS provider in the client's classpath.
- ○ Solution
  As far as possible use only one JMS provider in your application.
  If this is not applicable change the order of the classpath so that the JMS provider that is used for jadice server takes precedence over the other ones.
- • Exception when converting HTML documents / HTML mails
  - ○ error reports in server log

```
java.lang.StackOverflowError
   (…)
   at org.lobobrowser.html.parser.HtmlParser.parseToken(HtmlParser.java:642)
   at org.lobobrowser.html.parser.HtmlParser.parseToken(HtmlParser.java:679)
   at org.lobobrowser.html.parser.HtmlParser.parseToken(HtmlParser.java:679)
   (…)
   at org.lobobrowser.html.parser.HtmlParser.parse(HtmlParser.java:507)
   at org.lobobrowser.html.domimpl.HTMLDocumentImpl.load(HTMLDocumentImpl.java:386)
   at org.lobobrowser.html.domimpl.HTMLDocumentImpl.load(HTMLDocumentImpl.java:366)
   at org.lobobrowser.html.parser.DocumentBuilderImpl.parse(DocumentBuilderImpl.java:98)
   (…)
```

- ○ Possible cause
  Parsing of bad structured HTML documents exceeds the maximum stack deepth of the JVM.
- ○ Solution
  Increase the maximum stack depth of the JVM,
  e. g. set the paramenter -Xss2048k in the configuration file **wrapper/wrapper.conf** (if using an Oracle JVM).
- • Exception when converting MS PowerPoint Files
  - ○ error in server log

```
java.lang.Exception: com.jacob.com.ComFailException: Invoke of: Open Source:
Description:
   at com.jacob.com.Dispatch.invokev(Native Method)
   at com.jacob.com.Dispatch.invokev(Dispatch.java:858)
   at com.jacob.com.Dispatch.callN(Dispatch.java:455)
   at com.levigo.jadice.server.msoffice.MSPowerpointConverter.
   convert(MSPowerpointConverter.java:75)
```

- ○ error in MS PowerPoint
- ○ Possible cause:

Microsoft PowerPoint

Das Dokument enthält Makros. Makroprogrammierunterstützung ist für diese Anwendung deaktiviert. Features, die VBA benötigen, sind nicht verfügbar. Möchten Sie das Dokument schreibgeschützt öffnen?

Hilfe ausblenden <<   Im Hilfefenster öffnen

Dieser Fehler tritt in der Regel aufgrund von Makrosicherheitseinstellungen auf. Wenn Sie der Quelle des Makros vertrauen, können Sie die Makrosicherheitseinstellungen ändern und das Makro aktivieren.

Wenn die Datei in der geschützten Ansicht geöffnet wird, klicken Sie auf **Bearbeitung aktivieren** und dann auf **Inhalt aktivieren**. Bearbeitungsfunktionen werden wiederhergestellt, und der aktive Inhalt wird einschließlich Makros ausgeführt. Diese Datei ist jetzt vertrauenswürdig. Es werden nur noch Sicherheitshinweise angezeigt, wenn die Datei auf einem anderen Computer geöffnet wird.

Makroeinstellungen ändern

OK   Abbrechen   Hilfe

      MS PowerPoint needs an additional feature for files containing macros
- Solution:
  Install the feature "Visual Basic for applications" from the Office Installation
- Exception when converting with LibreofficePortable under Windows 2008:
  - error in server log

```
java.lang.NoClassDefFoundError: Could not initialize class
com.sun.star.lib.connections.pipe.PipeConnection
        at com.sun.star.lib.connections.pipe.pipeConnector.connect(pipeConnector.java:110)
        at com.sun.star.comp.connections.Connector.connect(Connector.java:114)
        at
com.levigo.jadice.server.core.util.ManagedObjectPool.borrowObject(ManagedObjectPool.java:52
)       at
…
```

- Solution:
  Add the following path to the systems "path" variable:
  „<LibreOffice-Verzeichnis> / App/libreoffice/program/"

10. Technical data

Supported formats (excerpt):

– IBM AFP and MO:DCA family
  – PTOCA
  – PTOCA
  – IOCA
  – GOCA
– HTML and e-mails
– TIFF, JPEG, PDF
– Archive formats
  – ZIP
  – GZIP
  – RAR
  – TAR
– Plaintext, XML, XSL:FO

Required qualifications:

Client-sided:

– jadice client extensions for server communication
– JRE 1.6.0 update 26, or JRE 1.7.0 update 3 or newer
– in use as applet, application or embedded

Server-sided:

– JRE 1.6.0 update 26, or JRE 1.7.0 update 3 or newer
– central memory: from 2 GB, 8 GB recommended
– processor: Pentium4 or higher, 3 GHz or faster recommended, dual processor
– hard disc storage: 80 MB software, 2 GB cache

Communication between server and client components

– Java Messaging Services (JMS)
– Multi-server process and load balancing are supported

Provided converting tools

– Office-Plugins for TIFF-export
– Office-Plugins for PDF-export
– jadice shaper

11. Document history

Version	Date	Author	Changes (syntax: * changed, + new, – omitted)
4.1.x	16.03.09	B. Geißelmeier	Completely re-edited
4.2.0.0	02.07.09	B. Geißelmeier	+ Migration from 4.1.x to 4.2.0.0 * Examples in chapter 5.3 adapted to changed API + Description web service interface
4.2.1.0	30.07.09	B. Geißelmeier	* Description web service interface + Chapter „Generation of web service clients" + Chapter „Configuration MS Office" + Chapter „Configuration Ghostscript"
4.2.1.1	31.08.09	B. Geißelmeier	* Description web service interface + Chapter„Job definition within the SOAP request" * Correction: Description of ExternalProcessCallNode
4.2.1.3	27.11.09	B. Geißelmeier	* Configuration MS Office + Configuration MS Outlook * Chapter „Converting e-mails to PDF" + Chapter„Troubleshooting" + Migration to version 4.2.1.3, chapter "Migration"
4.2.1.5	22.01.10	B. Geißelmeier	+ Chapter „Monitoring" * Chapter „Configuration of embedded message broker" * Chapter „Configuration MS Office" * Chapter „Configuration wrapper" + Migration to version 4.2.1.5, chapter„Migration"
4.3.1.0	25.06.10	B. Geißelmeier	+ Chapter „Security-Interface" + Chapter „Configuration of Limits" * Chapter „Troubleshooting" – Chapter „Migration", from now on in online-documentation
4.3.1.2	09.11.10	B. Geißelmeier	* Chapter "Web service Interface"
4.3.1.3	20.12.10	B. Geißelmeier	+ Chapter "Permanent anchoring of annotations"
4.3.1.5	18.04.11	B. Geißelmeier	* Chapter "Configuration of the messaging system" * Chapter "Configuration MS Office" * Chapter "Configuration LibreOffice" * Chapter "Converting e-mails to PDF" + Chapter "Configuration MS Project"
4.3.1.6	12.09.11	B. Geißelmeier	* Chapter "Troubleshooting", multiple JMS providers / clock sync. * Replaced German screenshots with English ones
4.3.1.11	01.02.12	B. Geißelmeier	* Chapter "Troubleshooting", Stack Overflow on HTML conversion
4.3.1.14	21.05.12	B. Geißelmeier	* Chapter "Configuration of the messaging system", Corrected property names
4.3.1.16	23.07.12	B. Geißelmeier	* Chapter "Converting e-mails to PDF", Clarified parameter unhandledAttachmentAction
4.3.1.20	28.01.13	F. Intorp	+ Chapter "Troubleshooting", Exception when converting MS PowerPoint Files
4.4.0.0	05.03.13	B. Geißelmeier	* removed content concerning outdated java versions * Chapter "Configuration of the messaging system", Distinguish between QueueConnectionFactory and TopicConnectionFactory
4.4.0.1	23.04.13	B. Geißelmeier	* Corrected example in "Permanent anchoring of annotations"
4.5.0.0	04.06.13	F. Intorp	* Address change
4.5.1.0	02.06.13	F. Intorp	* Chapter "Troubleshooting", Exception when converting with LibreofficePortable under Windows 2008