visioncloud

JAVA FORMATTING OBJECTS

- XSL-FO API
- RTF to XSL-FO Converter
- Reporting Solution

Manual

Table of content

1. Introduction

1.1 What is JFO

JFO is reporting and mail merge tool for Java applications, based on the XSL-FO (XML) standard and Rich Text Format (RTF) file format. Various output formats are available by embedding XSL-FO renderers (the open source render FOP by Apache is included). JFO can also be used as RTF to XML, PDF, Postscript, HTML, ... converter.

The included RTF to XSL-FO processor supports placeholders (i.e. fields, that are part of the RTF specification) to merge business data, tables, images and other objects into RTF templates. It also supports conditional parts (if statements). This technique allows you and your customers to design and edit report templates with your favourite RTF editor, e.g. Microsoft Word (TM) and to process these templates with JFO in your JAVA application.

Our XSL-FO Java API can be used to modify or create XSL-FO documents. In this way you can post process converted documents or you can create documents with Java code and render them easily to other formats.

Making reports has never been easier before.

1.2 Key Features

- **JAVA API** - Use JFO as a component in your application to create documents and reports based on the XSL-FO standard.
- **Frontend** - JFO can be used as a standalone application to convert RTF documents to various output formats. Use the command line interface in a console when necessary.
- **RTF to FO converter** - Transforms RTF to XSL-FO, PDF, Postscript and other formats (some output formats need an XSL-FO renderer, e.g. FOP that comes with JFO).
- **Mailmerge** - Populate RTF templates with business data during conversion. Repeated table-rows, repeated list-items and conditional parts are some supported features.
- **Report-Engine** - Build many reports on fly from a single RTF template and various data sources (XML, database...).
- **Supported XSL-FO renderers** - FOP 0.20.5, FOP 0.9x (FOP 0.95 is included in JFO), XEP, Ibex, Xinc and XSL-Formatter. Other renderers can be integrated very easy.

2. Features

2.1 XSL-FO API

- Simple Java API for creating XSL-FO documents

  Note: API is not based on a W3C DOM implementation (like Xerces) to provide a very handy programming interface, since the W3C DOM isn't very practicable in some situations.
- Pretty formatted XML output with support for different XML character sets
- Event driven SAX output for faster post-processing and avoiding temporary XSL-FO files
- Includes the open source renderer FOP (Apache) that supports many output formats (e.g. PDF, Postscript, AWT, Printer)
- Supports various commercial XSL-FO renderers: XEP, XSL-Formatter, Ibex, Xinc.

  Other XSL-FO renderers can be plugged in very easy. Contact us if you would like us to add support for another renderer.
- Includes HTML renderer, that uses a stylesheet provided by RenderX
- Simple flow writer that speeds up generating documents with Java

2.2 RTF to XSL-FO processor

- Text formatting properties
  - Font size and name
  - Styles: normal, bold, italic, underline
  - Foreground, background and auto-colors
  - Font-Stretching
  - Sub- and Subscripting
  - Font name mapping and character replacement
- Paragraph formatting properties
  - Idents: left, right and hanging (first line left)
  - Alignments: left, right, center, justify
  - Line-spacing
  - Space before and after
  - Paragraph border and backgrounds
  - Keeps, orphans and widows
  - Limitition: Very limited support for tabulators
- Pagination
  - Page settings: height, width, margins
  - Page borders (Note, that page borders are not part of the XSL-FO spec.)
  - Page breaks
  - Section breaks, incl. continuous section breaks
  - Multiple columns per page
  - Headers and footers, incl. title and facing pages
  - Footnotes

- Tables
  - Border properties: width, color and style (NOTE: not all RTF styles are supported by XSL-FO)
  - Vertical and horizontal alignment
  - Padding
  - Background color and patterns (Background patterns are converted to background images)
  - Row and column spanning
  - Fixed row height
  - Nested tables
  - Positioned tables
- Shapes
  - Textboxes (RTF shape type 202)
  - Pictureframe (RTF shape type 75)
  - Rectangle (RTF shape type 1)
  - Absolute and relative positions
  - Shape groups
  - Shape borders
  - Z-Index
- Images
  - Inline images
  - Image-frames
  - Suporting all RTF image formats: JPG, PNG, EMF and WMF. If an image of another format is inserted in a RTF file, the RTF editor converts it to an allowed format(PNG or WMF).
  - Image treatments: File, BASE64 encoded (instream), nested SVG.

    NOTE: When using BASE64 or SVG images are included directly in the XSL-FO output but not in an external file. Both methods are using the URL scheme "data" as defined in RFC 2397 for encoding.
  - Image processing to convert EMF/WMF into renderer friendly format (there isn't a XSL-FO renderer with support for EMF or WMF)
  - Watermark / background image for document
- Fields and Data-Merging
  - Handled fields: if, section, page, pageref, includepicture, docvariable, docproperty, data, hyperlink, title, author, comment and other
  - Format pattern for dates and numbers
  - Merge business data with Java Code (fillin, mergefield)
  - Repeating table rows
  - Conditioned data merging with if fields
- Bullets and Numberings
  - Bullets with any character
  - Auto numberings (arabic, roman and letters)
  - Limitition: Older RTF control words of Word 6 and Word 95 for bullets and numberings are not handled
- Compatibility XSL-FO output
  - Font replacement

        - Character replacement
        - Line height optimization
        - XSL-FO compatibility modes for
            - W3C (XSL-FO standard)
            - FOP 0.9x
            - FOP 0.20.5
            - XEP 4.0 and 4.4
            - XSL-Formatter 3
            - IBEX 3
- Compatibility RTF input
    - Fix for Microsofts Word padding bug in table cells
      See http://www.theimagingsourceforums.com/showthread.php?threadid=317071
    - Strict handling of headers and footers can be disabled
- Other
    - Language and hyphenation support
    - Hidden text
    - Bookmarks
    - Internal and external links
    - Processing of track changes

2.3 Report engine

- Create many reports on fly from a single RTF template
- Supports various data sources: CSV, XML, JDBC or Java-Bean

3. Installation

- Windows
- Unix and Linux
- List of JARs
- Requirements

3.1 Windows

After downloading JFO from our web-side www.vision-cloud.de, simply deflate the ZIP file into a directory of your choice. JFO needs a Java Runtime Environment (JRE) tu run, it can be downloaded for example at http://java.sun.com.

Please ensure that either

- **JAVA_HOME** environment variable is set; or
- The directory of your **java.exe** is listet in your **PATH** environment variable; or
- You have customized batch file **go.bat** of JFO to use a specific JRE

3.2 Linux / Unix

After downloading JFO from our web-side www.vision-cloud.de, simply deflate the ZIP file into a directory of your choice. JFO needs a Java Runtime Environment (JRE) tu run, it can be downloaded for example at http://java.sun.com.

Please ensure that either

- The **JAVA_HOME** environment variable is set; or
- You have customized file **go.sh** of JFO to use a specific JRE

3.3 List of JARs

- **lib/jfo.jar** - The core of JFO
- **lib/jfo-extension.jar** - Extensions of JFO that are distributation under GNU General Public License
- **lib/fex.jar** - Extension for FOP used to create PDF forms
- **lib/commons-logging.jar** - Jakarta Commons Logging (JCL) Framework used by JFO. (http://jakarta.apache.org/commons/logging/
- **lib/xercesImpl.jar** - Xerces 2.8.0, W3C XML-DOM Implementation (not required as of JDK 1.4+)
- **lib/xml-apis.jar** - JAXP (not required as of JDK 1.4+)
- **lib/thirdparty/fop.jar** - FOP XSL-FO processor for rendering XSL-FO files, http://xml.apache.org/fop/
- **lib/thridparty/avalon-framework-4.2.0.jar** - The Avalon Framework, used by FOP (http://avalon.apache.org)
- **lib/thirdparty/JimiProClasses.jar** - Image library, used by FOP (http://java.sun.com/products/jimi/)
- **lib/thirdparty/batik.jar** - Java SVG library, used by FOP (http://xml.apache.org/batik/)
- **lib/thirdparty/itext.jar** - iText is a free PDF library used by JFO to post-process PDFs (e.g. to concatenate PDFs) (http://www.lowagie.com/iText/)
- **lib/gui/kunststoff.jar** - Look and Feel used by JFOs graphical user interface

The lib directory of JFO contains all needed JARs of JFO, optional packages are located in subfolders **thirdparty** and **gui**.

3.4 Requirements

To run JFO you need at least

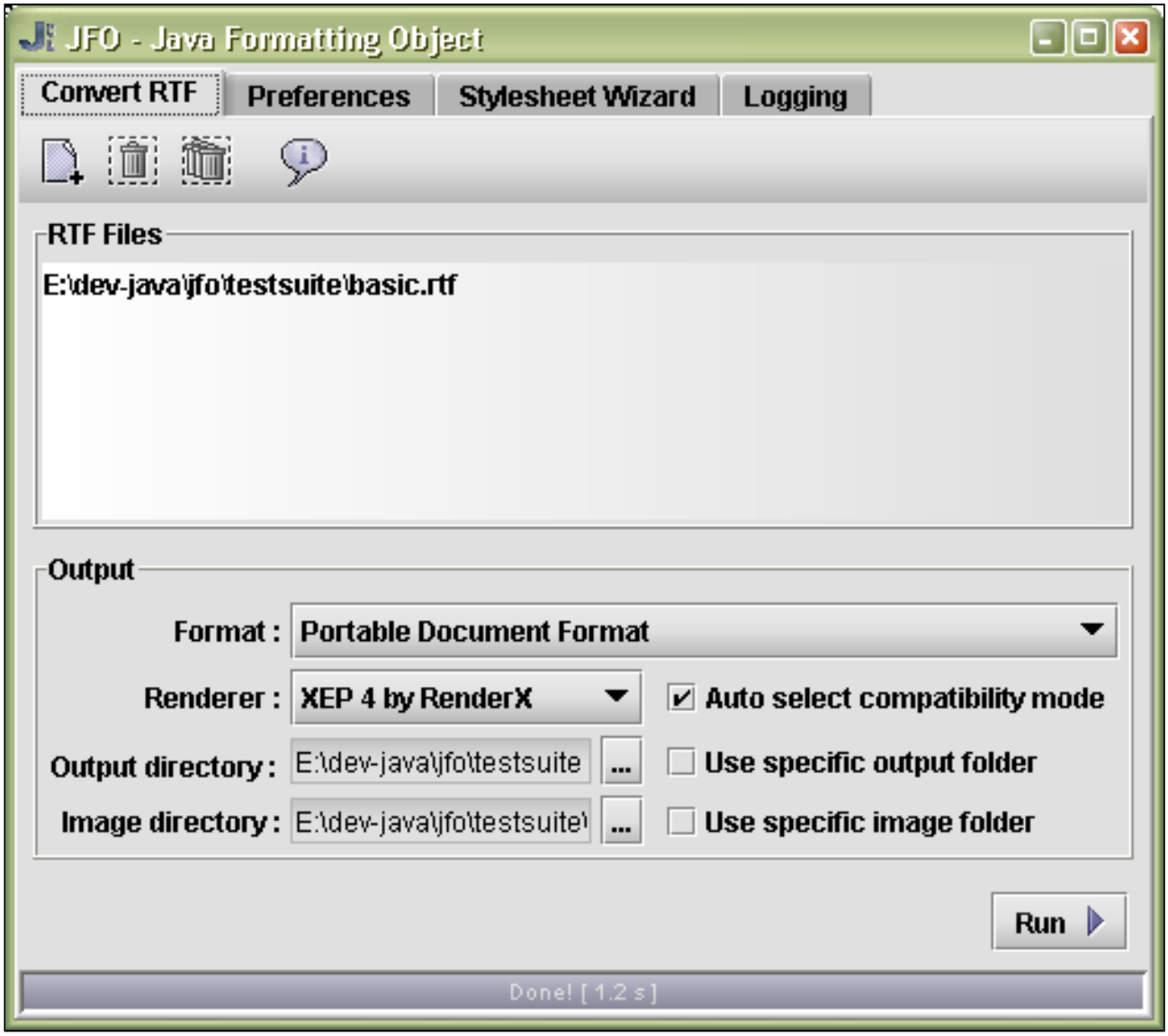
- JRE/JDK version 1.5 or later.

4. Converting RTF to XML, PDF...

- Converting with the GUI
- Converting with the Java API
- Supported fields
- Limititions

4.1 Converting with the GUI

After you've installed JFO simply call go.bat (Windows) or go.sh (Unix) to start. You will get some warning messages like "...Renderer unavailable..." that you can ignore. The following window should appear:



Use the  button to add one or more RTF files for conversion. To remove selected RTF files press  or use  to remove all RTF files. Generated output files are stored in the same folder of input files and images are put into a subfolder called "images". If you wish JFO putting the output into specific directories then

enable "Use specific output folder" and/or "Use specific image folder". Once you have choosen the desired output format (and a renderer) press "Run" to start the conversion.

JFO has multiple compatibility modes for a certain set of renderers. Since some renderers (especially PDF renderers) know their preferred compatibility mode, JFO can automatically select the correspondig mode if "Auto select compatibility mode" is enabled. This overrides compatibility mode settings of the preferences (See chapter Configuration). If a renderer without a specific compatibility mode is selected (e.g. XSL-FO), the checkbox is hidden and the compatibility mode of the preferences is used.

> Note: The number of supported output formats depends on the set of available renderers. If you have downloaded the small distributation package of JFO some output formats like PDF, Postscript are not available. Furthermore you can only select a renderer if an output format is supported by more than one renderer. Since a renderer has a priority, the one with the highest priority is preselected.

4.2 Converting with the Java API

- Example: examples/rtf2fo/RTF2FO.java

To convert an RTF document into an XSL-FO document create a new RTF importer and call the `importDocument` method:

```
de.vc.jfo.rtf.RTFImporter importer = new RTFImporter();
de.vc.jfo.Document doc = importer.importDocument( "file.rtf" );
```

Once you've created an XSL-FO document you can write out an XML file by calling: doc.writeTo(new File("file.fo")); or render it (with a renderer) directly into another output format: doc.render(OutputFormat.PDF, new File("file.pdf"));

Besides JFOs configuration, import parameters can be used to customize an import. Therefore create an instance of class ImportParameters, set the desired properties (e.g. image directory) and finally assign them to the importer before starting the import: de.vc.jfo.import.ImportParameter params = new ImportParameter params.setImagePath("myImagePath"); importer.setParameter(params);

> Note: Possible output formats depends on the list of available renderers.
>
> Note: Please read also the Java Docs for more information and a list of all possible parameters.

4.3 Supported fields

Below is a list of supported fields. Other fields may be handled by customized field listeners.

- **{page}** - Inserts the current page number
- **{numpages}** - Inserts the total number of pages
- **{includepicture "path/picture.jpg"}** - Inserts a picture
- **{includetext "other-file.rtf"}** - Inserts another RTF
- **{symbol 123 [\s font-size] [\f font]}** - Inserts a symbol with ASCII code 123 and a specific font and font-size.
- **{date \@ format}** - Inserts the current date
  - **format** - Format of date (e.g. **dd.MM.yyyy**)
- **{hyperlink "link" [\l "local-destination"]}** - Inserts in internal or external hyperlink
  - **\l "local-destination** - Local destination of document referenced by **link**
- **{docproperty "property"}** - Inserts a property of the document
- **{title}** - Inserts the title of the document
- **{author}** - Inserts the auther of the document

- **{comments}** - Inserts all comments of the document
- **{toc}** - Inserts a table of content
- **{pageref "mark"}** - Inserts the page number of text mark **mark**
- **{if "exp1" cmp "exp2" "trueText" ["falseText"]}** - Processes an IF statement
  - **exp1** - first expression
  - **cmp** - Compare operator; one of: = | > | < | <> | >= | <=
  - **exp2** - second expression
  - **trueText** - Text that is inserted if evaluation becomes true.
  - **falseTest** - Text that is inserted if evaluation becomes false.

  Note: The trueText and falseText may contain tables, nested fields, paragraphs and so on, but you must always put the whole result in quotation marks.

  Attention: There must be at least 1 space character between the expressions and the compare operator.

Optional parameters are surrounded by [ ]

4.4 Limititions

- limited tabulator support
- background patterns not supported
- track changes not supported
- drawing objects not supported

5. Creating Reports from RTF

Listening and processing RTF-fields The data-merge field listener Simple fields Repeated table rows Repeated lists Implementing own field listeners Listening for other import events Mail merge with the report engine The data source The report storage

5.1 Listening and processing RTF-fields

A RTF document can contain several fields. A field is a placeholder for dynamic text and data. For example the field **{page}** is used to insert the current page number and field **{numpages}** inserts the total number of pages. If you are not aware of fields, we recommend to play with Microsoft Word (open a field form by selecting Word's menu item **Insert->Field**).

To insert a field in MS Word fastly press Ctrl+F9. To toggle evaluation of fields on or off press Alt+F9.

Our RTF to XSL-FO importer has a list of field-listeners (de.vc.jfo.rtf.FieldListener). A field listener is notified when a field is encountered during the RTF parsing process. They are notified in that order they were registered. If a field is not handeled by any registered listeners (no listeners marks the field as processed), the internal field listener asked for processing. If the internal listener can't handle the field too, a default field result (as is stored in the RTF) is inserted. The list of all fields handeled by the internal field listener can be found here.

5.1.1 Merging data with the data merge field listener

- Example: rtf2fo/RTF2FOFilling.java
- Example: rtf2fo/RTF2FOwithIF.java
- Example: rtf2fo/DynamicImages.java
- Example: rtf2fo/FillinTables.java

JFO comes with an implementation of a field listener that is sufficient in most cases, it's the `de.vc.jfo.rtf.DataMergeFieldListener`. This listener holds a map of keywords and their values. If a field contains a registered keyword-token the assigned value is inserted for that field and the field is marked as processed.

The creation of the `DataMergeFieldListener` and registering values is very simple:

```
RTFImporer i = new RTFImporter();
DataMergeFieldListener l = new DataMergeFieldListener();
i.addFieldListener(l);
l.setValue("name", "Your name here");
```

A keyword can also be a phrase, a full sentence, a question and so on. If your field is **`{FILLIN "hello world"}`** your keyword is **"hello world"** and NOT **"hello"** or **"world"**. We always recommend to use **`{FILLIN}`** or **`{MERGEFIELD}`** as placeholders for data.

5.1.1.1 Simple values

Let's asume there is following RTF snippet:

Dear **{IF {FILLIN gender} = "male" "Mr." "Mrs."} {FILLIN name}**

... and we would like to get this result:

Dear Mrs. Sunshine

...we must register following values:

```
DataMergeFieldListener l = new DataMergeFieldListener();
l.setValue("gender", "female");
l.setValue("name", "Sunshine");
```

Nearly any type of object can be insert:

- Strings, Numbers and Dates
- Any subclass of `de.vc.jfo.flow.BlockGroupMember`
- Any subclass of `de.vc.jfo.flow.InlineGroupMember`
- `de.vc.jfo.flow.Wrapper`
- `de.vc.xml.DefaultNode` to insert foreign XML nodes
- For all other objects that are not a subclass of `de.vc.xml.NodeElement` the String representation from `toString()` is used.

5.1.1.2 Fill in repeated table rows

It's also possible to fill in tables. Let's have a look at following table:

Name	Phone number
{FILLIN names}	**{FILLIN "phone numbers"}**

... and we want this result:

Name	Phone number
Peter	1232132
Herbert	424342234

... this is the Java code:

```
DataMergeFieldListener l = new DataMergeFieldListener();
l.setTableColumn("names", new String[]{ "Peter" ,"1232132" } );
l.setTableColumn("phone numbers", new String[]{ "Herbert" ,"424342234" } );
```

You may also provide values of class `de.vc.jfo.flow.TableCell`. A table cell object gives you more control over cell-properties, e.g. you may set a background-color (`TableCell.getBackground().setColor(...)`) or span a cell over multiple columns (`TableCell.setColumnsSpanned(...)`). If a table cell a spanned, content and data of following cells will be ignored.

> Note: As of version 2.05 a table cell may contain more then one repeated value and also in combination with IF fields. This means you could put **{FILLIN names}** and **{FILLIN "phone numbers"}** in the same table-cell. In previous versions of JFO this was not possible.
>
> Note: Row spanning in repeated cells is currently not supported.

5.1.1.3 Fill in repeated lists

Enumerations and bullets (list items) are also repeatable:

List of attachments

- **{FILLIN attachments}**

... and we want this result:

List of attachments

- Letter A
- Letter B
- Letter C

... this is the Java code:

```
DataMergeFieldListener l = new DataMergeFieldListener();
l.setListItems("names", new String[]{ "Letter A","Letter B","Letter C" } );
```

It's also possible to set an array of `de.vc.jfo.flow.ListItemBody` or `de.vc.jfo.flow.ListItem`. If a list item array is used the item label is not generated automatically.

> Note: Multiple fields in a repeated list (like in tables) are currently not supported. This will be available in future releases.

5.1.2 Implementing own field listeners

If the `DataMergeFieldListener` doesn't fit your needs you can write your own field listener by implementing the `de.vc.jfo.rtf.FieldListener` interface. This interface has single method `void fieldEncountered(FieldContext context)` that is invoked when a field is encountered by the RTF parser. Use the `context` to collect more information about the current field and to insert elements at the current position or to execute any actions (like wrapping a node). The following example prints out all field instructions:

```
public void fieldEncountered( FieldContext context ) {
  System.out.println( context.getInstructionText() );
}
```

> Note: Look at the Java-Docs to see all available methods of a `FieldContext`.

5.2 Listening for other events during RTF import

Besides field listeners there exists another type of listeners. The `de.vc.jfo.rtf.ImportListener` listens with following methods:

- `public void onBookmarkStart(String bookmark, DocumentContext ctx)` - Invoked when the start of a bookmark is detected
- `public void onBookmarkEnd(String bookmark, DocumentContext ctx)` - Invoked when the end of a bookmark is detected
- `public String onInsertText(String text, DocumentContext ctx)` - Invoked when some text is inserted

The `DocumentContext` provides methodes to insert text and other elements at the current position, this is a superclass of `FieldContext`. Please read the Java Doc for more information and a detailed list of methods.

> Note: We recommend to extend the dummy implementation `ImportListenerAdapter`. Probably we will extend the interface from time to time and so you won't have to adapt any of your Java code.

5.3 Mail merge with the report engine

- Example: report/ReportJavaSource.java
- Example: report/ReportCSVSource.java
- Example: report/ReportXMLSource.java
- Example: report/ReportMixedXMLSource.java
- Example: report/ReportDBSource.java

If you want to create many reports on fly from a RTF form, there is a small reporting frameworking included in JFO. The usage of that framework differs a bit from a normal RTF to XML/PDF/PS... conversion. The RTF is parsed one time in a run, an in memory XML template is created and the XML template is filled for each report with data that comes from a data source. Finally the created report is stored in a report storage. During RTF parsing the report engine looks for **`{FILLIN "keyword"}`** and **`{MERGEFIELD "keyword"}`** fields. If such is a field is encountered a mark is set at fields position in the XML template. When the report engine builds a report, it replaces all marks with values from a report data.

The general usage pattern is as follows:

```
DataSource source = ...;
ReportStorage storage = ...;
FOReportBuilder reportBuilder = new FOReportBuilder( "myRtfTemplate.rtf" );
reportBuilder.setOutputFormat( OutputFormat.PDF );
new ReportEngine(source, reportBuilder, storage).run();
```

Note: The default output format of the report engine is XSL-FO. Other formats needs a renderer.

5.3.1 The report data source

As already mentioned the report data source delivers a set of report data (each data contains some key-value pairs). Class `de.vc.report.DataSource` has an abstract methode `protected void deliverData()`. If you want to implement your own data source you can use following code pattern:

```
protected void deliverData() {
  // create 5 dummy reports
  for (int reportIndex=0; reportIndex<5; reportIndex++) {
    ReportData data = new ReportData();
    data.setId( String.valueOf(reportIndex) );
    data.setValue( "keyword1", "value1");
    data.setValue( "keyword2", "value2");
    super.deliver( data );
  }
}
```

Each time `super.deliver(data)` is called a new report is generated. The report engine comes already with some data sources that are introduced shortly here. Also please have a look at the refered examples.

5.3.1.1 List data source

- Example: report/ReportJavaSource.java

- Example: report/ReportMixedXMLSource.java

The list data source holds the data of all reports in memory. This could cause memory problems if you generate many reports. But if you need only a few reports this data source might be very useful.

5.3.1.2 CSV data source

- Example: report/ReportCSVSource.java

This data source uses a CSV file as report source. Each CSV line represents the data for one report. JFO needs the keyword for each CSV-column, thus a bit Java code is needed:

```
String[] csvColumns = new String[]{"id","cd-artist","cd-title","cd-year","cd-label"};
CSVDataSource source = new CSVDataSource("myCsvFile.csv",csvColumns);
```

Note: It's also possible to use a CSV stream instead of a file.

5.3.1.3 XML data source

- Example: report/ReportXMLSource.java

You can also use an XML data source that matches this Document Type Definition. Here is a small example:

```
<report-data-set>
  <report-data id="CD1">
    <report-value name="cd-artist" value="Kraftwerk"/>
    <report-value name="cd-title" value="Mensche-Maschine"/>
    <report-table>
      <report-table-row>
        <report-value name="track-title" value="Die Roboter"/>
        <report-value name="track-length" value="6:17"/>
      </report-table-row>
      <report-table-row>
        <report-value name="track-title" value="Spacelab"/>
        <report-value name="track-length" value="5:56"/>
      </report-table-row>
    </report-table>
  </report-data>

  <report-data id="CD2">
    <report-value name="cd-artist">Enigma</report-value>
    <report-value name="cd-title">The screen behind the mirror</report-value>
    <report-table>
      <report-table-row>
        <report-value name="track-title" value="The Gate"/>
        <report-value name="track-length" value="2:03"/>
      </report-table-row>
    </report-table>
  </report-data>
</report-data-set>
```

5.3.1.4 JDBC data base as data source

- Example: report/ReportDBSource.java

It's also possible to use a database over JDBC as a data source. To set up the datasource you have to write an XML file that matches this Document Type Definition and contains information about the used data base, SQLs, column aliases and so on. Again a small example:

```
<report-data-set>
  <connection url="jdbc:mysql://localhost/test" user="" password=""/>

  <global-values>
    <selection>
      <sql-query>SELECT count(*) FROM album $WHERE_CLAUSE$</sql-query>
      <column-alias column="count(*)" alias="total-number"/>
    </selection>
  </global-values>

  <selection>
    <sql-query>SELECT * FROM album $WHERE_CLAUSE$</sql-query>
    <column-alias column="title" alias="cd-title"/>
    <column-alias column="artist" alias="cd-artist"/>
    <column-alias column="year" alias="cd-year"/>
    <column-alias column="label" alias="cd-label"/>
  </selection>

  <report-table>
    <selection>
      <sql-query>SELECT * FROM track WHERE album=$id$;</sql-query>
      <column-alias column="title"  alias="track-title"/>
      <column-alias column="length" alias="track-length"/>
    </selection>
  </report-table>
</report-data-set>
```

The `$WHERE_CLAUSE$` is replaced by a value that is set with java code.

5.3.2 The report storage

If a report is created it's forwarded to the report storage. Class `de.vc.report.ReportStorage` has a methode `protected abstract void storeReport(Report report, ReportContext ctx);` that is invoked for a generated report and should store the report available from `reportStream`. The `report` parameter represents the generated report and has methods for writing it into a output-stream or reading from an input-stream.

The `de.vc.report.storages.FileStorage` writes generated reports in a directory of your file system.

The `de.vc.report.storages.PDFMergeStorage` merges generated PDF reports and writes them into one big PDF file. This storage needs the thirdparty library **iText**.

6. Configuration

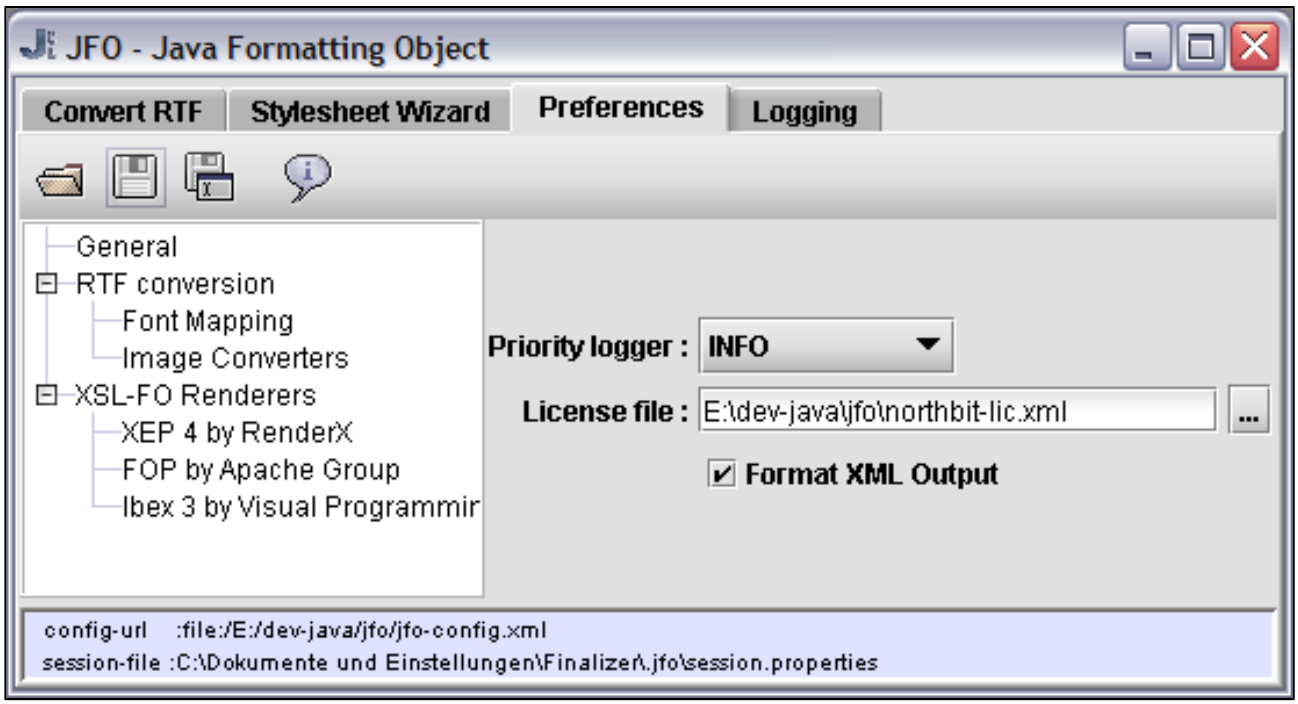
Loading and saving Logging General preferences RTF to XSL-FO Font-Mapping Renderers Image Transcoding

6.1 Configuration - Loading and Saving

JFOs configuration is stored in an XML-file, typicalled named **jfo-config.xml**. To alter the configuration either use the graphical user interface, edit the XML file manually or use the Java API of JFO.

6.1.1 Graphical User Interface

Nearly the whole configuration is editable comfortably with the GUI. To show and edit the preferences of JFO, select the "Preferences" tab. A status bar at the bottom of the preferences panel displays the URL of the currently loaded configuration. Also the location of the session file is shown. The session file holds information for JFOs GUI, e.g. last converted RTF files, loaded configuration, selected output format and other informations. When using JFOs API the session file has **no effect**.



The toolbar contains these actions:

-  - Load a configuration file
-  - Save configuration file (overwrites loaded)
-  - Save configuration file as
-  - Show JFO version information

6.1.2 Embedded

JFO tries to configure itself automatically on start up (when JFOConfig.instance() is invoked the first time) and uses the following discovery to find it's configuration file:

Check system property **de.vc.JFOConfigURL**. The property must be an URL or must point to a file or to a resource in classpath.

Check current working directory for file "jfo-config.xml"

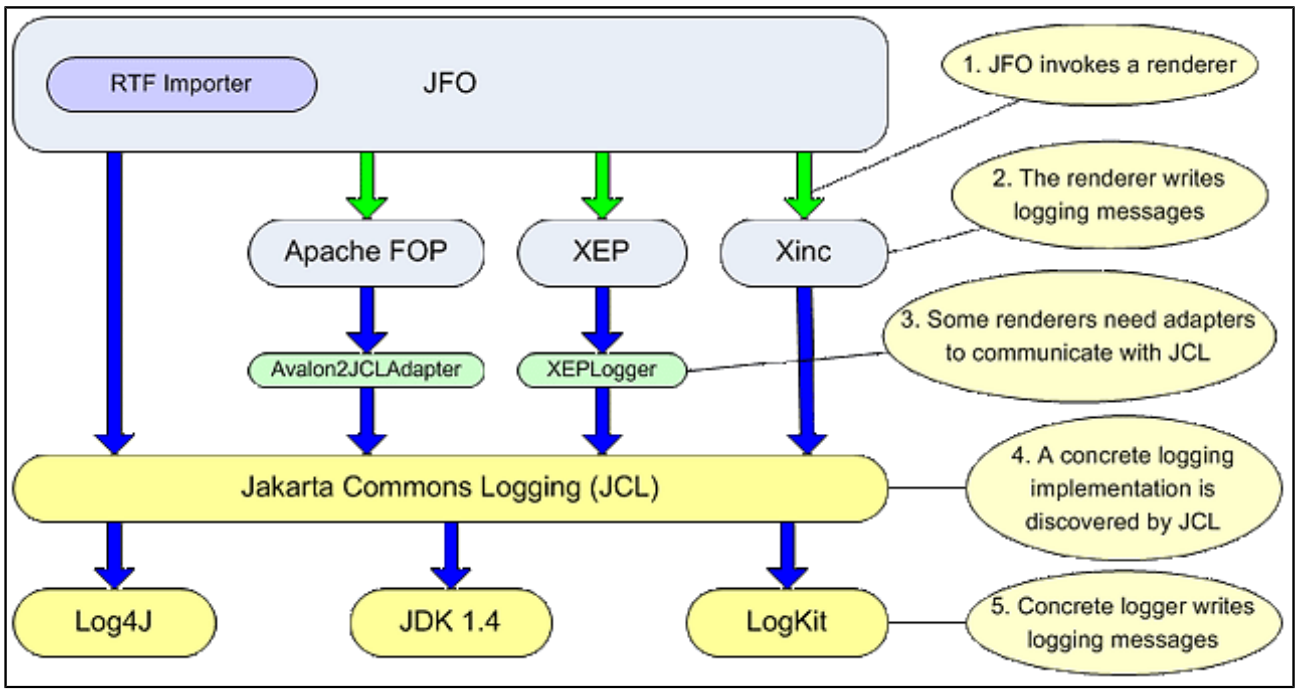
Check resource "jfo-config.xml" in classpath. (recommended, just put your jfo-config.xml in your classpath or JAR)

Also you can load the configuration by calling `JFOConfig.loadFrom(java.net.URL url);`. To save a configuration invoke `JFOConfig.instance().saveTo(File file)`.

6.2 Configuration - Logging

6.2.1 Jakarta Common Logging package

JFO doesn't use a concrete logging framework. Instead it uses the **Jakarta Common Logging** (JCL) package, available at http://jakarta.apache.org/commons/logging/. JCL is an ultra thin bridge between JFO and a logging implementations. JFO uses the JCL Log-Factory, that discovers a conrete log implementation, to obtain a logger. The logger name of JFO is "de.vc.jfo". When running JFO in a production environment, please read the User Guide of JCL to ensure a correct logging. For testing and evaluation there is most likely no need to read the JCL guide, since JCL initialize itself in a reasonable manner.



Logging in JFO

A JFO configuration file can contain a certain log level. This level is only used if running the JFO-GUI or if JCL falls back to the SimpleLog implementation. If JFO is running in an application or on a server, JCL is probably using / wrapping Log4J or JDK 1.4 logging (see JCL User Guide). In this case the configured log level is ignored then, since a level can only be set for a concrete log implementation.

6.2.2 Setup for Java Logging API (JDK 1.4 logger)

IF JCL discovers the JDK 1.4 logging framework, you probably would like to change the format and destination of logging messages. A full documentation of the Java Logging API is available at http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/. Following examples gives you a quick startup:

```
import java.util.logging.*;

...

// Get the JDK 1.4 logger for JFO
Logger logger = Logger.getLogger("de.vc.jfo");

// Set level for logger to INFO
logger.setLevel(Level.INFO);

// Dont inherit parent handlers
logger.setUseParentHandlers(false);

// But use our own handler
StreamHandler h = new java.util.logging.StreamHandler(
  System.err, new Formatter() {
    public String format(LogRecord record) {
      return record.getLoggerName()+" ["+record.getLevel().toString()+"] "+record.getMessage()+"\n";
    }
  }
);

// Handler may display ALL log levels
h.setLevel(Level.ALL);

// And add the handler to our logger
logger.addHandler(h);

// Run JFO now...
```

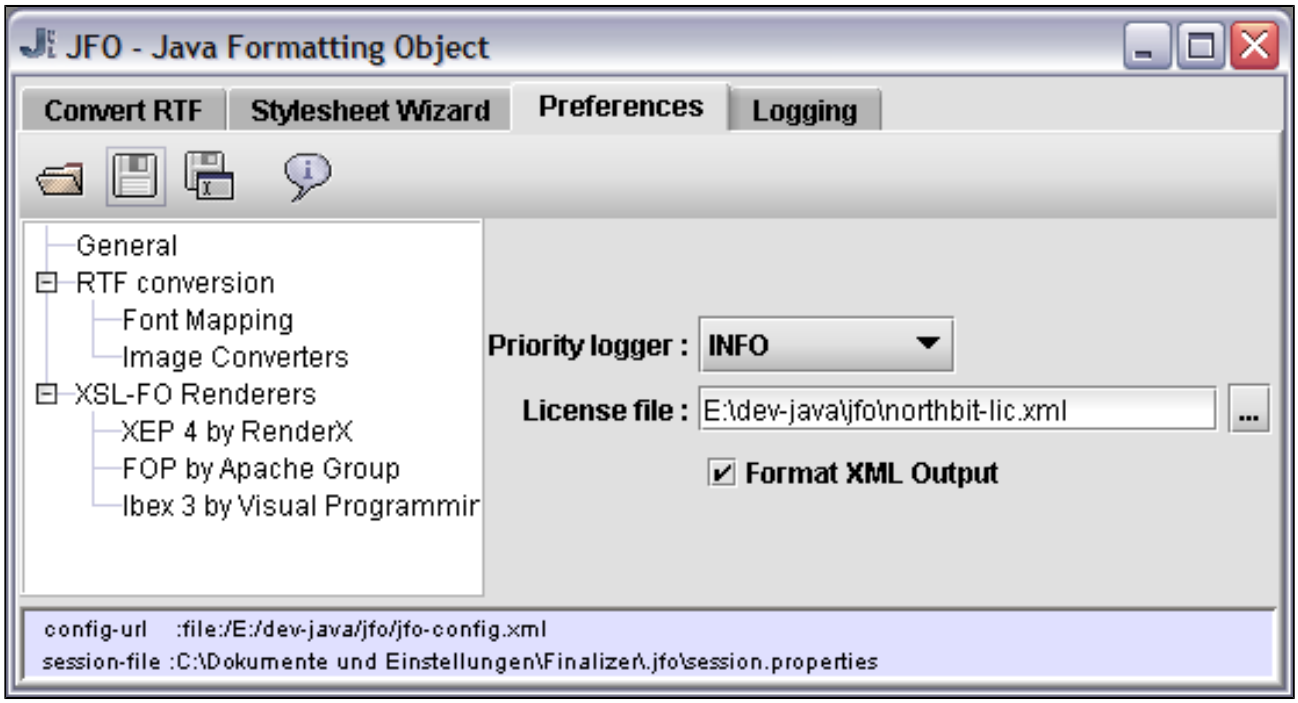
6.2.3 Using a concrete JCL logger

As already mentioned, JFO uses the JCL LogFactory to get a logger. However, you can also set a concrete logger manually using the Java API.

```
org.apache.commons.logging.Log logger =
  new org.apache.commons.logging.impl.Log4JLogger( org.apache.log4j.Logger log );
  // new org.apache.commons.logging.impl.Jdk14Logger( java.util.logging.Logger log );
  // ... ;
JFOConfig.instance().setLogger(logger);
```

Note, we recommend to avoid setting a logger manually if possible.

6.3 Configuration - General Preferences



6.3.1 Logging

JFO writes **logging** information during import and rendering phases. You can choose betwwen 5 log levels: DEBUG, INFO, WARN, ERROR and FATAL-ERROR.

Configuration via jfo-config.xml

```
<configuration>
...<logger priority="INFO"/>...
</configuration>
```

Possible values are `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL_ERROR`.

Configuration via Java API

```
JFOConfig.instance().setLogger(org.apache.avalon.framework.logger.Logger logger);
```

Since the logger of the avalon framework is just a wrapper for other loggers like Log4J, JDK1.4-Logger, LogKit and other, you can integrate JFO into your logging system. Some wrappers for other logging tools are:

- `org.apache.avalon.framework.logger.Jdk14Logger(java.util.logging.Logger jdkLogger)`
- `org.apache.avalon.framework.logger.Log4JLogger(org.apache.log4j.Logger log4jLogger)`

- `org.apache.avalon.framework.logger.LogKitLogger(org.apache.log.Logger logkitLogger)`

The `org.apache.avalon.framework.logger.ConsoleLogger(int logLevel)` dumps out logging information to the console.

6.3.2 Activation JFO (Licensing)

After purchasing a JFO license you've obtained a **license key**. This license key can either be put into a classpath directory or JAR, or you can specify the location of this file.

> Note:As of version 3.0 the license key is named "vc-license.xml", in previous versions "northbit-lic.xml" or "license.dd".

Configuration via jfo-config.xml

```
<configuration>
...<license-key file="vc-license.xml"/>...
</configuration>
```

Configuration via Java API

```
JFOConfig.instance().setLicenseKeyFile(URL url);
```

6.3.3 xmlFormatting

If you want to read with JFO generated XSL-FO files, you may want to enable **pretty XML formatting**. Then line breaks and indententions are inserted that makes the file better human readable with a text editor.

Configuration via jfo-config.xml

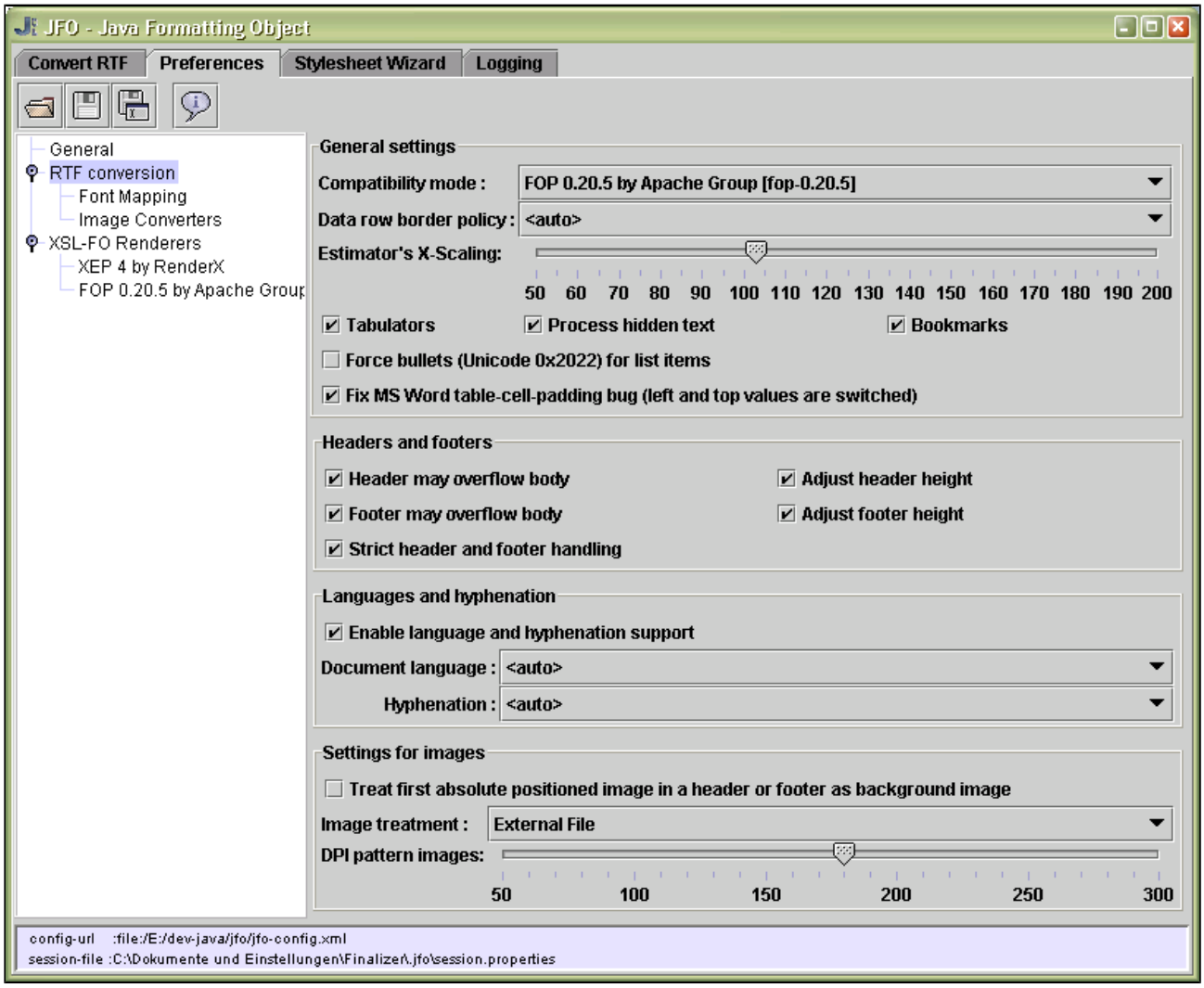
```
<configuration>
...<xml-formatting enable="true"/>...
</configuration>
```

Configuration via Java API

```
JFOConfig.instance().setFormatXMLOutput(boolean enable);
```

6.4 Configuration - RTF to XSL-FO processor

- General settings
- Headers and footers
- Language and hyphenation support
- Images



The configuration of the RTF to XSL-FO converter is stored in node `<rtf2fo-config>` of the configuration file. With Java it's accessible by calling:

```
RTF2FOConfig config = JFOConfig.instance().getRTF2FOConfig();
```

6.4.1 General settings

6.4.1.1 Specific Renderer

Use the **compatibility mode** to optimize the XSL-FO output for a specific renderer. Especially for Apache FOP JFO provides some work arounds for missing features or minor bugs (e.g. bottom-alignment in a footer, left margin of tables, etc.).

- **w3c** - XSL-FO standard by W3C.
- **fop-0.20.5** - FOP 0.20.5 by Apache Group (distributed with JFO)
- **xep-4.0** - XEP 4.0 by RenderX
- **xep-4.4** - XEP 4.4 by RenderX
- **axf-3** - XSL Formatter 3 by Antenna House
- **ibex-3** - IBEX 3 by visual Programming

  Note: The uppler list may be incomplete. A complete list of supported compatibility modes is available in the GUI of JFO (tab Preferences) or as a comment in the distributed configuration file `jfo-config.xml`.

Configuration via jfo-config.xml

```
<rtf2fo-config compatibility-mode="w3c" .../>
```

Some values are **w3c**, **fop-0.20.5**, **xep-4.0**, **xep-4.4**, **axf-3**, **ibex-3**. Also, please refer to the comment in JFOs configuration file `jfo-config.xml`.

Configuration via Java API

```
RTF2FOConfig.setCompatibilityMode(RTF2FOCompatibilityMode mode);
```

Example values are `RTF2FOCompatibilityMode.W3C`, `RTF2FOCompatibilityMode.FOP_0_20_5`. Please have a look at the JavaDoc for a complete list. You may also look up the compatibility mode by name:

```
RTF2FOConfig.setCompatibilityMode( RTF2FOCompatibilityMode.modeForName( "xep-4.4" ) );
```

6.4.1.2 Border

If a table row is filled (with user values) and repeated, a border conflict comes up if the top and bottom border is not equal. Use the **data row border policy** to set JFO what border is used when table rows are repeated:

- **AUTO** - The XSL-FO renderer resolves according to the XSL-FO specification what type of border is used. See border conflict resolution.
- **NO-BORDER** - No border is put between repeated table rows.

- **TOP-BORDER** - Use top border of master row between repeated table rows.
- **BOTTOM-BORDER** - Use the bottom border of master row between repeated table rows.

Configuration via jfo-config.xml

```
<rtf2fo-config data-row-border-policy="auto" .../>
```

Possible values are **auto**, **no-border**, **top-border**, **bottom-border**.

Configuration via Java API

```
RTF2FOConfig.setDataRowBorderPolicy(int policy);
```

Possible values are `RTF2FOConfig.DATA_ROW_BORDER_POLICY_AUTO`, `RTF2FOConfig.DATA_ROW_BORDER_POLICY_NO_BORDER`, `RTF2FOConfig.DATA_ROW_BORDER_POLICY_TOP_BORDER`, `RTF2FOConfig.DATA_ROW_BORDER_POLICY_BOTTOM_BORDER`.

6.4.1.3 Tabulators

The processing of **tabulators** can be enabled or disabled. JFO provides only a minimal support for tabulators since there is no equivalent in XSL-FO.

> Note: JFO needs the AWT to support tabs, i.e. on unix/linux systems you must either run an XWindows server or start your JVM in headless mode.

Configuration via jfo-config.xml

```
<rtf2fo-config enable-tabs="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setEnableTabulators(boolean enable);
```

6.4.1.4 Bookmarks

An RTF document may contain several **bookmarks**. A bookmark is converted to an fo:wrapper with ID attribute. Several bookmarks can start at the same place, so several fo:wrappers are created. Some renderers (e.g. FOP) have problems with following fo:wrappers. In this case you can disable bookmarks.

Configuration via jfo-config.xml

```
<rtf2fo-config enable-bookmarks="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setEnableBookmarks(boolean enable);
```

6.4.1.5 Bullets

Lists in an RTF document can have specific characters as a **bullet** (e.g. a square, triangle...). Must of these symbols are part of a special symbol font (Microsofts Symbol font). Since not all renderers may support a special symbol, you can force JFO to convert any bullet symbol to Unicode • (•).

Configuration via jfo-config.xml

```
<rtf2fo-config force-symbol-bullet="false" .../>
```

Configuration via Java API

```
RTF2FOConfig.setForceSymbolBullet(boolean force);
```

6.4.1.6 Text hidden

RTF documents may contain text that is marked as **hidden text**. If you want JFO to process this text you should set this property to true.

Configuration via jfo-config.xml

```
<rtf2fo-config process-hidden-text="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setProcessHiddenText(boolean enable);
```

6.4.1.7 MS Word Bugfix

Microsoft Word has a bug in interpreting and writing individual table cell padding properties of an RTF file. Left and top padding values are swapped. To fix the **MS Word table-cell-padding bug** you should

enable the corresponding checkbox. See also http://www.theimagingsourceforums.com/showthread.php?threadid=317071

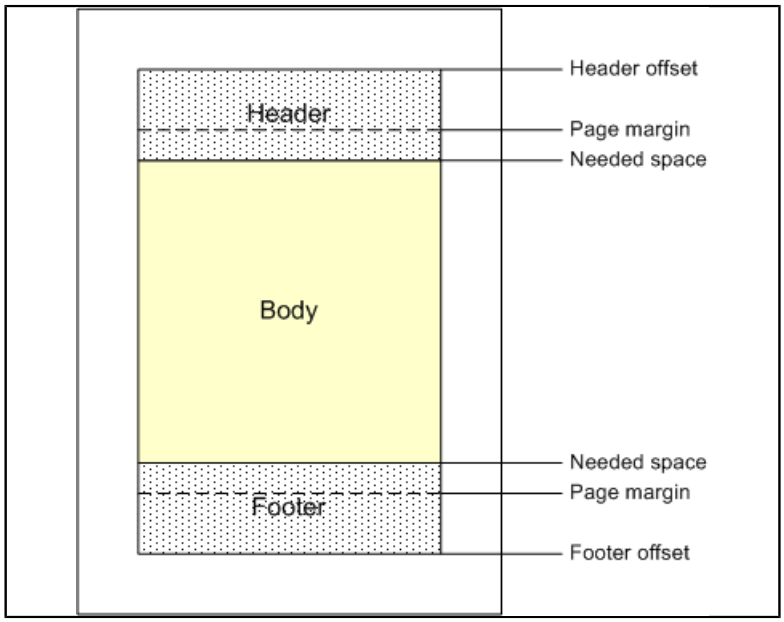
Configuration via jfo-config.xml

```
<rtf2fo-config fix-word-cell-padding-bug="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setFixWordCellPaddingBug(boolean fix);
```

6.4.2 Headers and footers



The right image shows some properties of a page. Header and footer offsets are the gaps between the document edge and the top edge of header / bottom edge of footer. Page margins are the space between the document edge and the top / bottom edge of the body (filled yellow). The area between header/footer offset and page margin is reserved for header and footer text (filled with dots).

However, in some situations the content of a header or footer needs more space than the page properties reserve. Since we have to define a fixed header and footer-height in XSL-FO there could be problems if the needed height is larger than the reserved. JFO offers two properties to control this.

If you enable **adjust header/footer height** the height of headers and footers is estimated by JFO. The maximum value of reserved and estimated height is applied to header/footer height.

If you enable the **header/footer overflow** property a header/footer is not cropped or suppressed by the renderer and it could overlay the body.

We recommend to enable both properties.

> Note: JFO needs the AWT to adjust the height of headers and footers. Thus you need to run your JVM in headless mode (`-Djava.awt.headless=true`) on Unix/Linux systems if no XWindows server is running.

6.4.2.1 Height of a header

Enable **adjust header height** to allow JFO increasing the height of a header if needed.

Configuration via jfo-config.xml

```
<rtf2fo-config adjust-header-height="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setAdjustHeaderHeight(boolean enable);
```

6.4.2.2 Height of a footer

Enable **adjust footer height** to allow JFO increasing the height of a footer if needed.

Configuration via jfo-config.xml

```
<rtf2fo-config adjust-footer-height="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setAdjustFooterHeight(boolean enable);
```

6.4.2.3 Header to overflow

To ensure that the header is never cropped or suppressed if the reserved height is insufficient, you can allow the header to overflow the body.

Configuration via jfo-config.xml

```
<rtf2fo-config header-overflow-visible="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setHeaderOverflowVisible(boolean enable);
```

6.4.2.4 Footer to overflow

To ensure that the footer is never cropped or suppressed if the reserved height is insufficient, you can allow the footer to overflow the body.

Configuration via jfo-config.xml

```
<rtf2fo-config footer-overflow-visible="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setFooterOverflowVisible(boolean enable);
```

6.4.2.5 Header-Footer-Handling

The RTF specification says: "In particular, if \facingp is not set, then only \header and \footer should be used; if \facingp is set, then only \headerl, \headerr, \footerl, and \footerr should be used." However, there are RTF documents with just a single \headerr (right header) and no standard header (\header). MS Word handles those \headerr like a standard header and doesn't respect the specification. When disabling strict footer/header handling, JFO behaves like MS Word for those (really rare) cases.

Configuration via jfo-config.xml

```
<rtf2fo-config strict-header-footer-handling="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setStrictHeaderFooterHandling(boolean strict);
```

6.4.3 Language and hyphenation support

6.4.3.1 Language Support

RTFs may contain information about the language they are written in. JFO can translate the RTF language code (LCID) into an ISO 639/ISO 3166 code and apply the ISO code to generated fo:blocks. The language is overridable by the `document-language` property.

> Note: Since the language code is applied to each paragraph, one may turn off language support to reduce XSL-FO size and improve speed. In some situation, especially when creating reports, JFO uses the language (Locale) to format date and numbers correctly. Nevertheless, if you turn off language support the language is still used to format special values.

Configuration via jfo-config.xml

```
<rtf2fo-config enable-language-support="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setEnableLanguageSupport(boolean enable);
```

6.4.3.2 Hyphenation

Hyphenation is initially turned off in XSL-FO and must be explicitly turn on. You can choose between enabling hyphenation if it's also enabled in the RTF, disabling hyphenation (regardless if enabled in RTF) or forcing hyphenation (regardless if disabled in RTF).

> Note: If you are using FOP (shipped with JFO) to render your documents, you'll probably need to do some configuration. Have a look at FOPs Homepage for details

Configuration via jfo-config.xml

```
<rtf2fo-config hyphenation-policy="auto" .../>
```

Possible values are **auto**, **force** and **disable**.

Configuration via Java API

```
RTF2FOConfig.setHyphenationPolicy(int policy);
```

Possible values are `RTF2FOConfig.HYPHENATION_POLICY_AUTO`, `RTF2FOConfig.HYPHENATION_POLICY_FORCE`, `RTF2FOConfig.HYPHENATION_POLICY_DISABLE`

6.4.3.3 Document Language

The language of an RTF text can be "overridden" during conversion. If language support is enabled, the overridden language is applied to XSL-FO instead of the language in the RTF document. The language is encoded by an ISO 639/ISO 3166 code like "en" (language only) or "en_US" (language + country).

Configuration via jfo-config.xml

```
<rtf2fo-config document-language="de_DE" .../>
```

The special value **auto** disables overriding and the languages of the converted RTF are used.

Configuration via Java API

```
RTF2FOConfig.setDocumentLanguage(java.util.Locale locale);
```

Use a `null` Locale to disable overriding.

6.4.4 Images

6.4.4.1 Image Treatment

The **image-treatment** sets the method, how images are stored in the resulting XSL-FO document.

- **external** - An external file is created in the image directory (see also section converting).
- **instream** - JFO uses the data schema (base64 encoded) of an URL to embed the image. (see RFC 2397)
- **svg** - The image is stored as a nested SVG graphic.

  Attention: The data URL scheme is currently not handled by the JVM itselfs. Thus JFO automatically installs an URL stream handler if possible (see also Java DOC, `JFOConfig.checkUrlDataScheme()`).

Configuration via jfo-config.xml

```
<rtf2fo-config image-treatment="external" .../>
```

Possible values are **external**, **instream**, **svg-instream**.

Configuration via Java API

```
RTF2FOConfig.setImageTreatment(int treatment);
```

Possible values are `RTF2FOConfig.IMAGE_TREATMENT_EXTERNAL`, `RTF2FOConfig.IMAGE_TREATMENT_INSTREAM`, `RTF2FOConfig.IMAGE_TREATMENT_SVG_INSTREAM`.

6.4.4.2 Image Background

An XSL-FO document can have a **background-image** (watermark). If this property is enabled, the first absolute positioned image of a header or footer is used as background image.

Configuration via jfo-config.xml

```
<rtf2fo-config enable-background-image="true" .../>
```

Configuration via Java API

```
RTF2FOConfig.setEnableBackgroundImage(boolean enable);
```

6.4.4.3 Image DPI

Some RTF elements, especially table cells, can have background patterns. For those patterns, JFO generates background-images. Use the pattern-image-dpi property to set the DPI (dots per inch) of generated images.

> Note: Some renderers ignore the DPI value and use a fixed DPI (e.g. FOP 0.20.5 uses 72 DPI always)

Configuration via jfo-config.xml

```
<rtf2fo-config pattern-image-dpi="180" .../>
```

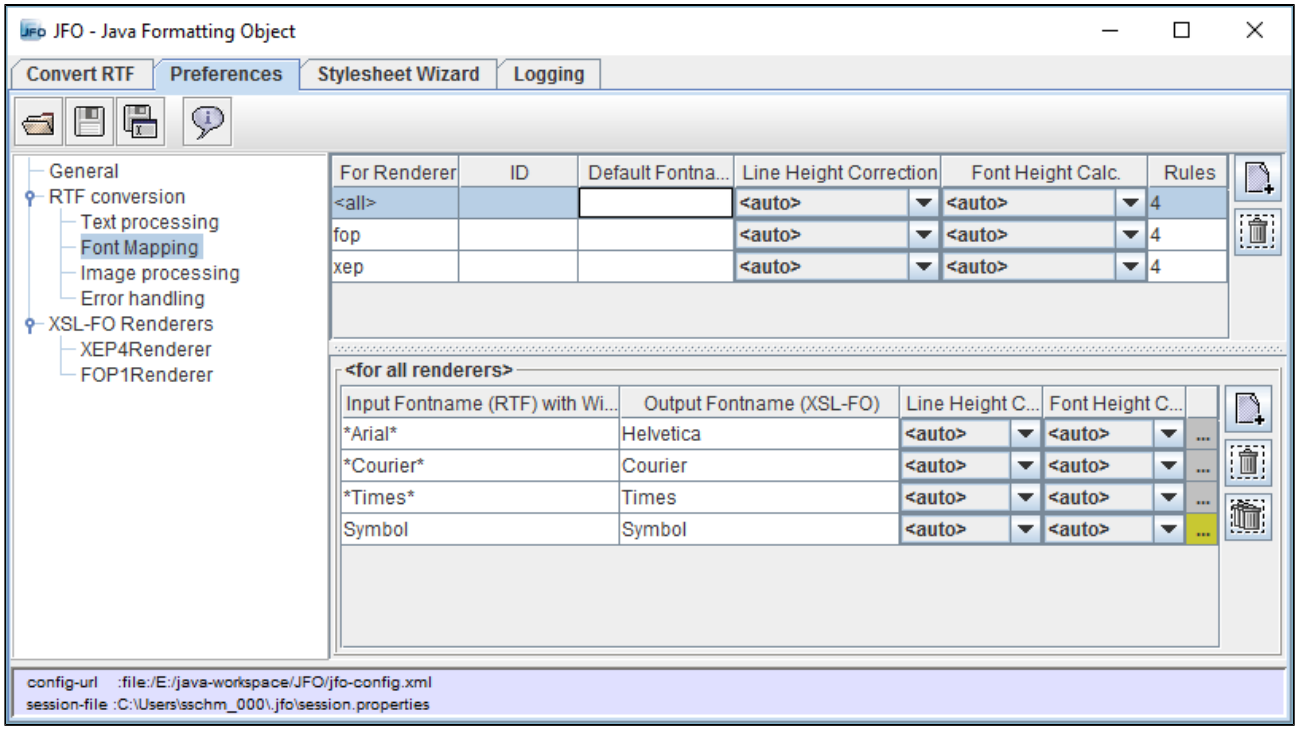
Use a value between 70 and 300 DPI. 180 DPI seems to be a good value.

Configuration via Java API

```
RTF2FOConfig.setPatternImageDPI(int dpi);
```

6.5 Configuration - Font Mapping

6.5.1 Font names and properties



Many renderers, especially PDF renderers, are naturally supporting only few fonts (Adobe Fonts) without doing further configuration. In such situations it could be very useful to **replace unsupported fonts**. Thus JFO is initially configured to replace "Times New Roman" to "Times", "Arial" to "Helvetica" and so on. This job is done by a so called **FontMapper**. In addition one may specify a default font that is used for unreplaced fonts.

When replacing a font there could arise the need to replace single characters too and not just the font name. This is often used for symbol fonts. See below for more details.

Renderers may vary in handling line heights of fonts. To produce high quality results you can set a global and font specific **font-line-height** and **line-height-correction**.

The **font-line-height** is used to determine the line-height for a given font-size. Common practice is, that the line-height is 115%-120% of the font-size, i.e. the distance between two lines with a given font-size of 12pt is between 13.8pt and 14.4pt. According to our experience, Microsoft Word uses a 115% of font-size as line-height. JFO needs to calculate the absolute line-height for paragraphs with a multiple line height (e.g. 2 times height). Let's asume your document has a paragraph with 12pt font-size and double line-height. In that case the calculated line-height is not 2 * 12pt = 24pt, but 2 * 12pt * 115% = 27.6pt. If a absolute/fixed line height is given in the RTF document, it won't be changed by the font-line-height. One of these strings are accepted as font-line-height as well:

- **auto** - Uses the standard value of 115%
- **disabled** - Disables font-line-height calculation. JFO outputs percentages for paragraphs with mutltiple line heights (e.g. 200% for double line height)
- **java-font-metrics** - Use Javas Font Metrics to compute the line-height You can customize by extending class `de.vc.jfo.rtf.FontMetricsHelper` and overriding Method `calculateJavaFontLineHeight()`. Use method `RTFImporter.setFontMetricsHelper()` to register your customized implementation.
- **50-200%** - Multiply font-size with given percentage to calculate the line-height

Once the line-height has been calculated, the **line-height-correction** is applied. This includes absolute line heights as well as to relative/calculated line heights. Let's asume a document that has one paragraph with a fixed line-height of 20pt and a second paragraph with 12pt font-size, double line height and a configured font-line-height=115%. Without a line-height-correction, the XSL-FO documents first paragraph keeps the line-height of 20pt and the second paragraph gets a computed line-height of 2*12*115%=27.6.8pt. If we now apply a line-height-correction with 150%, the first paragraph has a height of 20pt * 150% = 30pt and the second 27.6 x 150% = 41.4%. One of these strings are accepted as line-height-correction as well:

- **auto** - Auto line-height-correction (defaults to **enabled**)
- **enabled** - Enables line-height-correction (only effective if font-line-height has not been disabled)
- **disabled** - Disables line-height-correction
- **50-200%** - Enables line-height-correction with the given percentage

  Note: A line-height-correction in percentage is required very seldom. But you should set "line-height-correction"="auto" to force a defined line-height attribute on each paragraph. This ensures, that even paragraphs without a fixed or multiplied line height (in the RTF) will have a defined line height in the output document.

  Note: The test document line-height.rtf in the testsuite directory may help you finding correct line heights.

As pointed out, font mapping depends from the used XSL-FO renderer. Thus JFO supports **renderer-specific font-mappings**. JFO's default configuration contains a default font mapping and mappings for FOP, XEP and maybe other renderers. So if the compatibility mode is set to FOP or XEP, JFO uses the best matching font mapper. The font mapping is resolved as follows for a certain compatibiity mode:

Check whether there is a font mapping thats `for-renderer` (e.g. 'fop-0.20.5') attribute matches compatibility modes name (e.g. 'fop-0.20.5').

Look for a font mapping thats `for-renderer` attribute (e.g. 'fop') matches the begin of compatibility modes name (e.g. 'fop-0.20.5'). Use the mapping with the longest match. (e.g. use 'xep-4' and not 'xep' for compatibility mode 'xep-4.4')

Fallback to default font mapping

Feel free to add further font mappings for other renderers if necessary. BTW, if you use only one renderer in your project, you can remove all specific font-mappers and use the default-mapping only. This makes the configuration file smaller and better readable.

The following snippet is a typical example of a font-mapping inside a JFO configurion file.

```
<configuration>
...
  <font-mapping default-font="Helvetica">
    <font-entry input-font="*Arial*"   ouput-font="Helvetica"/>
    <font-entry input-font="*Courier*" ouput-font="Courier"/>
    <font-entry input-font="*Times*"   ouput-font="Times"/>
    <font-entry input-font="symbol"    ouput-font="ms_symbol"/>
  </font-mapping>

  <font-mapping for-renderer="xep" line-height-correction="enabled" font-line-height="110%">
    <font-entry input-font="*Arial*"   ouput-font="Helvetica"/>
    <font-entry input-font="*Courier*" ouput-font="Courier"/>
    <font-entry input-font="*Times*"   ouput-font="Times"/>
    <font-entry input-font="symbol"    ouput-font="ms_symbol"/>
  </font-mapping>
...
</configuration>
```

Note: The input-font-name is not case sensitive and may contain wildcards.

If you need to set up font-mapping through the Java API, please refer to the Java Docs of class `de.vc.jfo.tools.FontMapper`. To override the font mapping declared in the currently loaded configuration, call `setFontMapping(FontMapper mapper)` on the ImportParameters object of the RTFImporter.

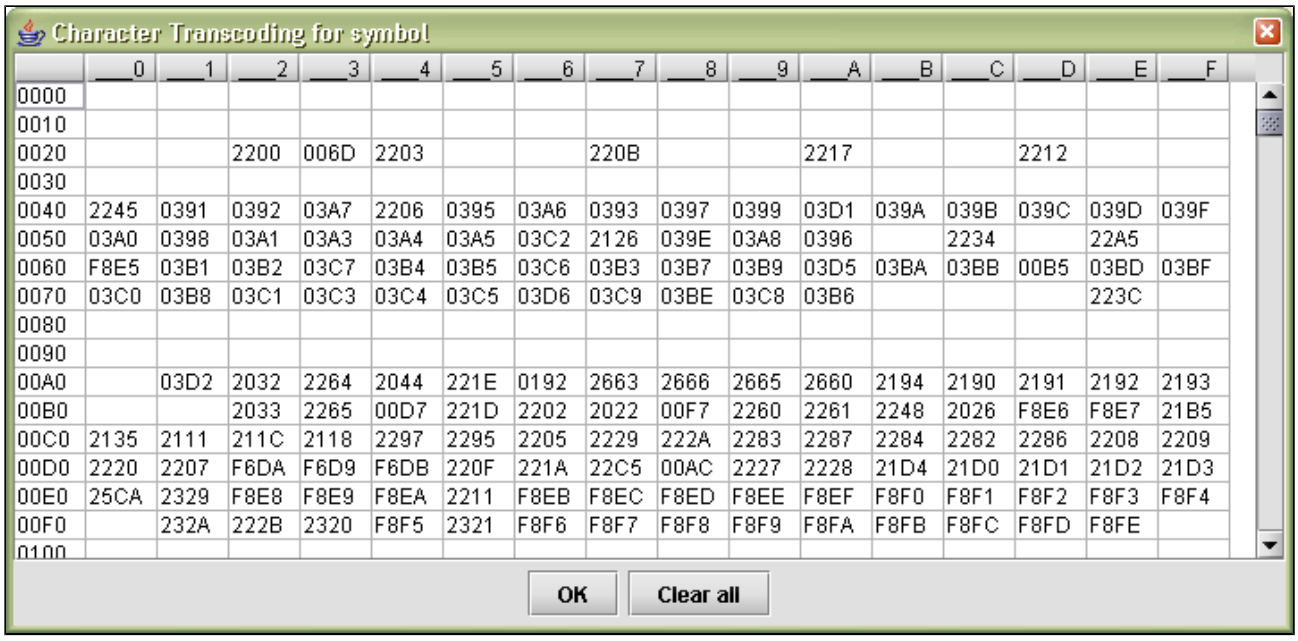
6.5.2 Replacing characters

Character transcoding (replacing) is available as of version 2.06 of JFO. This feature was introduced for replacing characters of Microsofts Symbol Font with those of Adobes Symbol Font. Prior, Microsofts Symbol Font has to be embedded in PDFs to support bullets and other symbols. In many cases embedding MS Symbol is not necessary any longer, since common Symbols of Microsoft's Symbol font can be replaced by a Symbol of Adobe's Symbol font. With JFO's API character replacement is set up as follows:

```
FontMapper mapper = JFOConfig.instance().getDefaultFontMapper();
FontMappingRule r = mapper.addMappingRule( "*Arial*", "Helvetica" );
// FontMappingRule r = mapper.getMappingRule("Arial"); // also possible
f.setTranscoding('A','X');
f.setTranscoding('B','Y');
```

Note: You must ensure a font mapping rule was added (`addMappingRule`) prior adding character transcodings, otherwise `getMappingRule` returns `null` and character transcoding is not setable.

When converting RTF with the example above, each occurrence of 'A' is replaced by 'X' and 'B' by 'Y'. The picture below shows transcoding rules displayed in JFO's GUI. To open the rules, click on the "..." button of a font-mapping table-row, a yellow button indicates existing rules. The hexadecimal value of row-label + column-label is the Unicode of an RTF character and a table cell displays the transcoded char, also as hexadecimal unicode. Click on a cell to edit a value. An empty table cell means that the character is not replaced. Example: Unicode 0x0022 is replaced by 0x2200, Unicode 0x0040 is replaced by 0x2245.



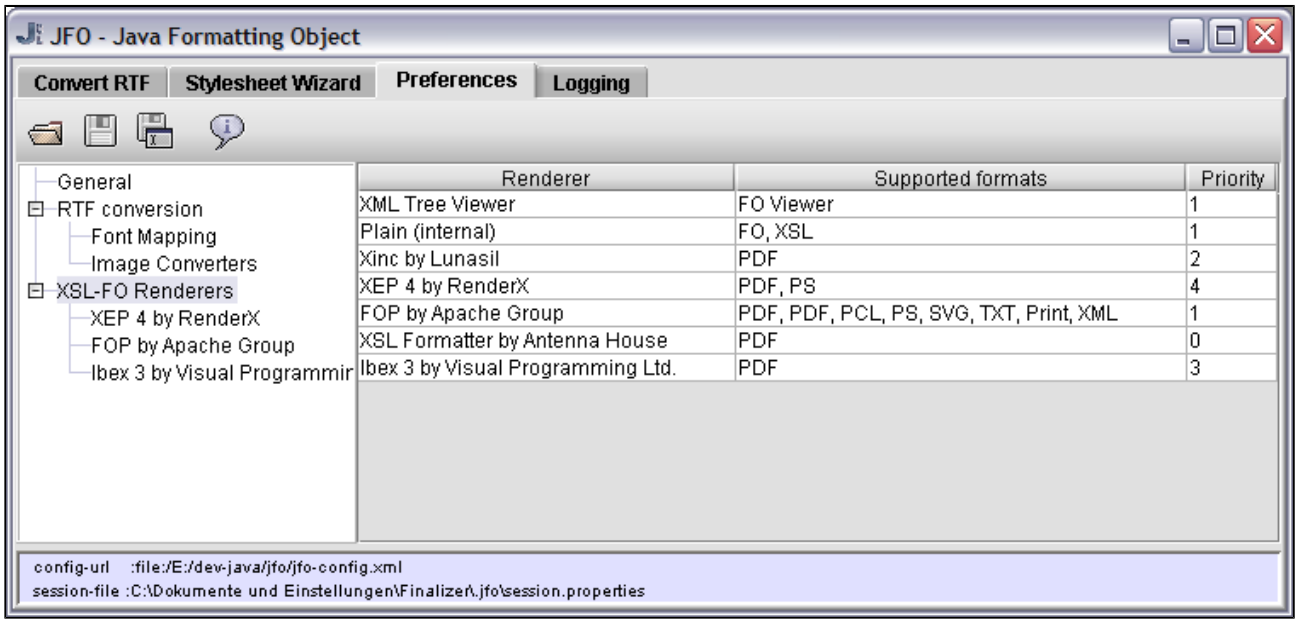
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000																
0010																
0020			2200	006D	2203			220B			2217			2212		
0030																
0040	2245	0391	0392	03A7	2206	0395	03A6	0393	0397	0399	03D1	039A	039B	039C	039D	039F
0050	03A0	0398	03A1	03A3	03A4	03A5	03C2	2126	039E	03A8	0396		2234		22A5	
0060	F8E5	03B1	03B2	03C7	03B4	03B5	03C6	03B3	03B7	03B9	03D5	03BA	03BB	00B5	03BD	03BF
0070	03C0	03B8	03C1	03C3	03C4	03C5	03D6	03C9	03BE	03C8	03B6				223C	
0080																
0090																
00A0		03D2	2032	2264	2044	221E	0192	2663	2666	2665	2660	2194	2190	2191	2192	2193
00B0			2033	2265	00D7	221D	2202	2022	00F7	2260	2261	2248	2026	F8E6	F8E7	21B5
00C0	2135	2111	211C	2118	2297	2295	2205	2229	222A	2283	2287	2284	2282	2286	2208	2209
00D0	2220	2207	F6DA	F6D9	F6DB	220F	221A	22C5	00AC	2227	2228	21D4	21D0	21D1	21D2	21D3
00E0	25CA	2329	F8E8	F8E9	F8EA	2211	F8EB	F8EC	F8ED	F8EE	F8EF	F8F0	F8F1	F8F2	F8F3	F8F4
00F0		232A	222B	2320	F8F5	2321	F8F6	F8F7	F8F8	F8F9	F8FA	F8FB	F8FC	F8FD	F8FE	
0100																

Of course transcoding rules are parts of JFOs configuration file (jfo-config.xml). Let's look at the example below:

```
<configuration>
...
  <font-mapping>
    <font-entry input-font="Symbol" ouput-font="Symbol">
      <transcoding offset="0040">
        , 0058, 0059
      </transcoding>
    </font-entry>
  </font-mapping>
...
</configuration>
```

A font-entry element contains (zero, one or many) transcoding elements. Each transcoding element contains a chunk of comma separated, hexadecimally encoded Unicodes. A character thats Unicode is equal to offset-value (also hexadecimal) is replaced by the first character of the comma separated list. Offset+1 is replaced by the second character in list and so on... The above example replaces 0x0041 'A' with 0x0058 'X' and 0x0042 'B' with 0x0059 'Y'. 0x0040 is passed through, since the first token (before the first comma) is empty. JFO's configuration file contains initially a character transcoding for Microsoft Symbol font to Adobes Symbol font, perhaps you would like to have a look at it. ;-)

6.6 Configuration - Renderers



Every XSL-FO renderer can be plugged in JFO, only one a simple abstract class must be extended (`de.vc.jfo.renderer.Renderer`). JFO comes already with support for famous renderers like FOP, XEP, XSL-Formatter, Ibex 3 and Xinc. Available renderers can be configured through the configuration file:

```
<configuration>
  ...
    ...
    <renderer class="de.vc.jfo.renderer.FOPRenderer" priority="1"
        config-file="fop-userconfig.xml" reset-image-cache="true"/>
    <renderer class="de.vc.jfo.renderer.XEP4Renderer" priority="4" config-file="xep.xml"/>
    ...
  ...
</configuration>
```

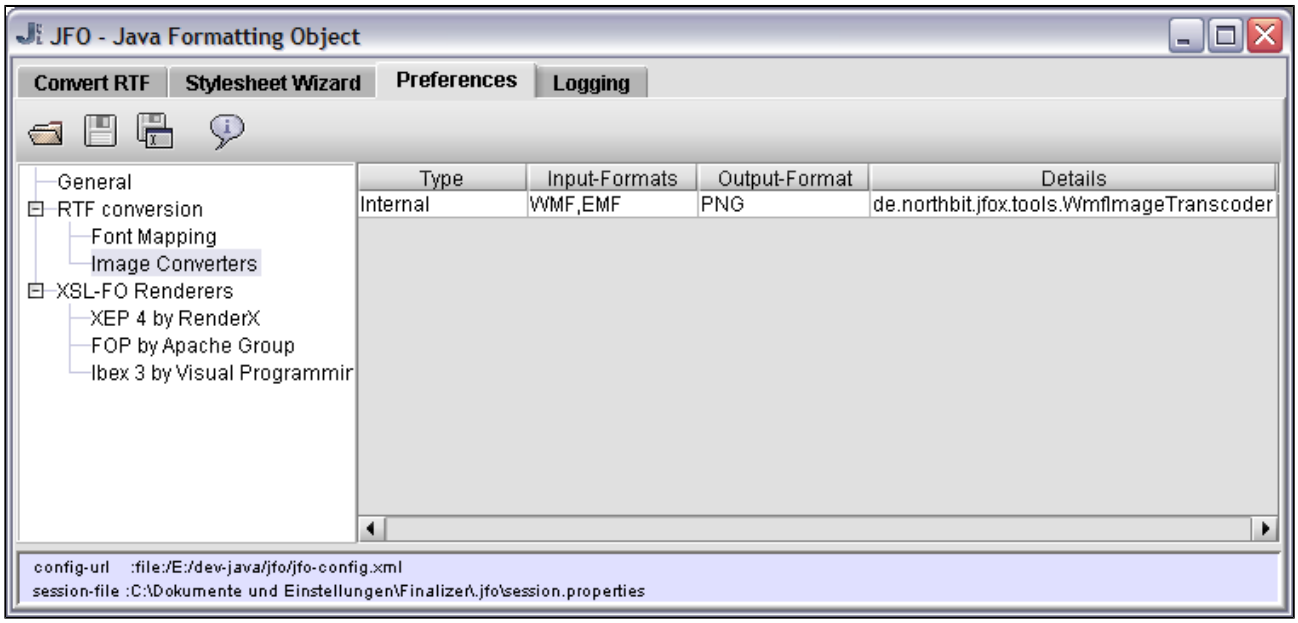
The priority attribute gives JFO a hint, what renderer is to use if multiple renderers support a given output format. A higher priority wins (e.g. 4 is heigher than 1). You may also use a concrete renderer in your Java Code:

```
OutputStream out = ...;
Document doc = ...;
Renderer renderer = ...;
doc.render(renderer, OutputFormat.PDF, out);
```

But note, if you want to change a renderer you would have to adapt and recompile your Java code and must recompile. We always recommend to configure the prefered renderer and call `doc.renderer(OutputFormat out, OutputStream out)` which selects the concrete renderer according the highest priority (and of course output format).

Note: Please contact us (www.vision-cloud.de) if you need to plug-in another renderer.

6.7 Configuration - Image Transcoders



An **image converter** converts an image of type A into an image of type B. RTF documents may contain images in WMF, EMF (or any other) format that are currently not supported by any (?) renderer. To solve this problem you can configure image converts that transform RTF images into a renderer friendly format. The configuration of image converters can only be done through the configuration file or the Java API, configuring using the GUI is not possible.

External image converts are command line tools that are invoked to convert by a native system call. For example IrfanView is such a tool and is available at http://www.irfanview.com/. Class `NativeCmdImageTranscode` wrappes the call to the external tool and invokes native command. To add an external converter to JFOs configuration simply add following line, **`?i`** is a placeholder for the image input file (e.g. c:/images/pic123.wmf) and **`?o`** for the output file (e.g. c:/images/pic123.png).:

```
<configuration>
  ...
  <image-converter type="native-cmd" input-format="wmf" output-format="png"
      command="i_view32.exe ?i /convert=?o"/>
  ...
</configuration>
```

And with the Java API:

```
ImageTranscode ic = new NativeCmdImageTranscode("wmf","png","i_view32.exe ?i /convert=?o");
JFOConfig.instance().addImageTranscode(ic);
```

Internal image converters transcode images through Java Code and must extend class `de.vc.jfo.tools.InternalImageTranscoder`. JFO is shipped with a simple implementations of a transcoder that converts from EMF and WMF into PNG using an open source WMF decoder. Source code can be found in directory `extensions/src/de/vc/jfox/tools/`. Following snippets show how an internal image converted is added.

```
<configuration>
  ...
  <image-converter type="internal" class="de.vc.jfox.tools.WmfImageTranscoder"/>
  ...
</configuration>
```

or

```
ImageTranscode ic = new MyWmfImageTranscoder();
JFOConfig.instance().addImageTranscode(ic);
```

7. FAQ, Tipps & Tricks

General Questions Questions related to RTF conversion Rendering documents Server integration

7.1 General Questions

What are these warnings `[WARNING] ...Renderer unavailable due to java.lang.NoClassDefFoundError?`

When JFO loads its configuration it checks the availability of configured renderers. If a concrete renderer is not in the classpath, a warning is logged out. Simply remove (or comment out) the corresponding renderer-tags in your `jfo-config.xml` to avoid these warnings.

Why are there so many lines with `... de.vc.util.log.Avalon2JCLAdapter` on the console?

This indicates that JFO (actually FOP) writes it's logging messages to the Java Logging API, introduced with Java 1.4. Since JFO uses the Jakarta Commons Logging (JCL), it doesn't use the Java Logging API directly. Instead FOP sends its logging messages through an adapter to JCL and JCL sends them to the Java Logging API. See the logging section in the manual for details.

Is the JFO configuration object a singleton?

Yes, the JFO configuration object (`JFOConfig.instance()`) is a singleton. This means, only one instance exists at classloader-level (class instance). The URL of the (last) loaded configuration is stored in the system property `de.vc.JFOConfigURL`. If the singleton is destroyed after a class garbage collection and `JFOConfig.instance()` is called later, the old configuration is loaded.

Is there a way to send SAX events from an imported document?

Yes, you may send SAX events from an XSL-FO document by calling `document.fireSaxEvents(ContentHandler ch)`

Is the API threadsafe?

No, the XSL-FO API and the RTF-Importer are not thread safe.

7.2 RTF to XSL-FO conversion

Can an RTF importer be re-used?

Yes.

The footer of my RTF documents is placed at the top of the document.

If you are running JFO together with FOP, you should ensure that you have set the compatibility-mode to FOP.

7.3 Rendering documents

Is a renderer threadsafe?

Yes, a renderer object can be used by different threads, but calls to the renderer method are synchronized. Thus real multithreading is only possible with different renderer objects.

7.4 Server Integration

Where do I have to put my jfo-config.xml for my web-app?

Copy your jfo-config.xml to WEB-INF/classes/jfo-config.xml.