ARMATIEK

Armatiek BV
Nieuwendammerdijk 40
1025 LN Amsterdam
The Netherlands
☎ +31612998045

http://www.armatiek.com
info@armatiek.com

Infofuze Technical Overview

Maarten Kroon, Arjan Loeffen

Armatiek BV

Version 1.0, March 2013

Table of Contents

1 Introduction to Infofuze....4
1.1 Java API for XML Processing (JAXP)....4
1.2 Source classes....4
1.3 Result classes....5
1.4 Transformer classes....5
1.5 Streaming transformations....6
1.5.1 Merging source streams....6
1.5.2 Splitting result streams....7
1.6 Executing transformations....8
1.6.1 Use the Infofuze Command Line Interface (CLI)....9
1.6.2 Write (and schedule) an Ant Task....9
1.6.3 Write your own Java code....9
1.7 Full and delta transformations....10
1.8 Configuring Infofuze....10
1.8.1 infofuze-config.xml....11
1.8.2 jobs.xml....13
1.8.3 tasks.xml....14
2 Source classes....15
2.1 Filesystem based source classes....15
2.1.1 File types....15
2.1.1.1 XML files....15
2.1.1.2 Binary files....16
2.1.1.3 JSON files....17
2.1.1.4 CSV files....17
2.1.1.5 Compressed files....18
2.1.1.6 Unparseable files....19
2.1.2 Filesystem based sources....19
2.1.2.1 LocalFileSystemSource....19
2.1.2.2 CIFSSource....20
2.1.2.3 WebDAVSource....21
2.1.2.4 WebCrawlSource....22
2.1.2.5 FTPSource....22
2.2 JDBC based sources....23
2.2.1 DirectJDBCSource....23
2.2.2 PooledJDBCSource....23
2.2.3 JNDIJDBCSource....24
2.3 SolrSource....24
2.4 MongoDBSource....25
2.5 HTTPSource....26
2.6 LDAPSource....26
2.7 NullSource....27
3 Result classes....28
3.1 FileResult....28
3.2 SolrjResult....28
3.3 MongoDBResult....29
3.4 JDBC based results....29
3.4.1 DirectJDBCResult....30
3.4.2 PooledJDBCResult....30
3.4.3 JNDIJDBCResult....31
3.5 JDBCResult....32
3.6 HTTPResult....32
3.7 NullResult....32

4 Third party components....33
5 Building Infofuze....34
6 Deploying Infofuze....35
6.1 infofuze-core jar....35
6.2 infofuze-cli jar....35
6.3 infofuze-web-core jar....35
6.4 infofuze-web-backend war....35
6.5 infofuze-web-frontend war....36

1 Introduction to Infofuze

Infofuze is a Java library and server application that can be used to transform and combine XML stream representations of various sources into a specific XML output stream that can be stored or indexed. These transformations can be fully configured and scheduled. Infofuze is based on the XML transformation interface of the Java API for XML Processing (JAXP) which is bundled with standard Java (J2SE). Infofuze is written in 100% pure Java and will run on all systems for which a Java 1.6 Virtual Machine is available.

1.1 Java API for XML Processing (JAXP)

Diagram 1 shows this XML transformation interface in action. A TransformerFactory object is instantiated, and used to create a Transformer. The source object is the input to the transformation process and must implement the interface javax.xml.transform.Source. The source object can be (among others) of the standard classes SAXSource, DOMSource, or StreamSource. Similarly, the result object is the result of the transformation process and must implement the interface javax.xml.transform.Result. That object can be (among others) of the standard classes SAXResult, DOMResult, or StreamResult.

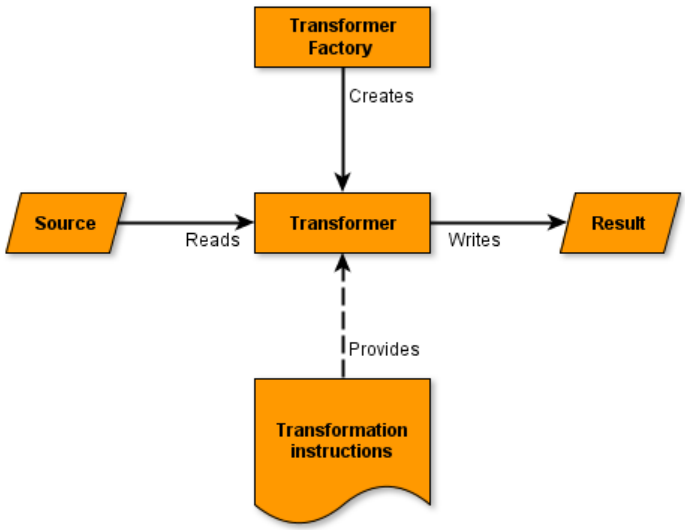

Diagram 1: JAXP XML transformation interface

When the transformer is created, it may be created from a set of transformation instructions, in which case the specified transformations are carried out. These transformation instructions can be in the XSLT or STX language, depending on the implementation of the Transformer object. (STX is a transformation language for **streaming** XML transformations). If it is created without any specific instructions, then the transformer object simply copies the source to the result.

1.2 Source classes

In Infofuze, the Source object can be of any class that implements the interface javax.xml.transform.Source or any of the Source classes that are provided by Infofuze. These Source classes include:

- Filesystem based sources*:
  - LocalFileSystemSource
  - CIFSSource
  - WebDAVSource
  - WebCrawlSource

    - FTPSource
    - HTTPSource
- JDBC based sources:
    - DirectJDBCSource
    - PooledJDBCSource
    - JNDIJDBCSource
- SolrSource
- MongoDBSource
- HTTPSource
- LDAPSource
- NullSource

(*) The filesystem based sources have specific support for structured transformation of XML, JSON, CSV, binary and compressed files (see section 2.1.1)

The task of these Source classes is to provide as much structured data as possible from specific sources in the form of an XML stream (of any form). Therefore these classes all extend from StreamSource. The Infofuze Source classes are further described in chapter 2.

1.3 Result classes

The Result object can be of any class that implements the interface javax.xml.transform.Result (like StreamResult which can be used to write the output to a file) or any of the Result classes that are provided by Infofuze. These Result classes include:

- FileResult
- SolrjResult
- MongoDBResult
- JDBC based results:
    - DirectJDBCResult
    - PooledJDBCResult
    - JNDIJDBCResult
- HTTPResult
- NullResult

The task of these Result classes is to stream XML data (of a specific form, like the Solr update format) to a specific data storage or index (like a Solr instance, a relational database or webservice). Therefore these classes all extend from StreamResult or SAXResult. The Infofuze Result classes are further described chapter 3.

1.4 Transformer classes

The Transformer object can be an XSLT transformer, like the one provided by the XSLT processors Saxon or Xalan, or a STX transformer, like the one provided by Joost.

1.5 Streaming transformations

Although the transformation used by Infofuze can be a XSLT transformation, in most cases this is not the right choice for XML sources that are bigger than, let's say, a couple of tens of megabytes. An XSLT transformation requires that the whole stream is read completely in memory before the transformation can start.
Therefore in most cases it is preferred to do an STX transformation. STX (Streaming Transformations for XML) resembles XSLT on the syntactic level, but where the XSLT template rules match on DOM nodes, STX template rules match on SAX events. You could say that STX combines SAX processing with the XSLT syntax. STX has one big drawback though; where in XSLT you have full random access to all XML nodes in the complete document, in STX you can only access the current node and its ancestors. Luckily, for non-trivial transformations of large XML streams STX and XSLT can be combined so you can use XSLT to do more complex “local” transformations in the context of a STX transformation.

1.5.1 Merging source streams

During the transformation of the main Source, other Sources can be read and merged into the main stream using XPath's document() function (XSLT) or the stx:process-document instruction (STX). The parameter *uri* or the attribute *href* must have the following format:

```
source://<sourcename>[?param_1=value_1&param_n=value_n]
```

where *sourcename* is the name of a source configured in infofuze-config.xml (see section 1.8.1).

XSLT example:

```
<xsl:apply-templates
  select="document(concat('source://my-jdbcsource?id=',$id))/resultset/row"/>
```

STX example:

```
<stx:process-document
  href="concat('source://my-jdbcsource?id=',$id)/resultset/row"/>
```

For instance, during the transformation of the primary source, an Oracle database view, one can query the same database for n x n relations, or query a Microsoft SQL server database, Directory Service or CSV file for additional data. These secondary sources can be read parameterized. These parameters are for instance used in a JDBC Source to fill in parameters of a parameterized query.

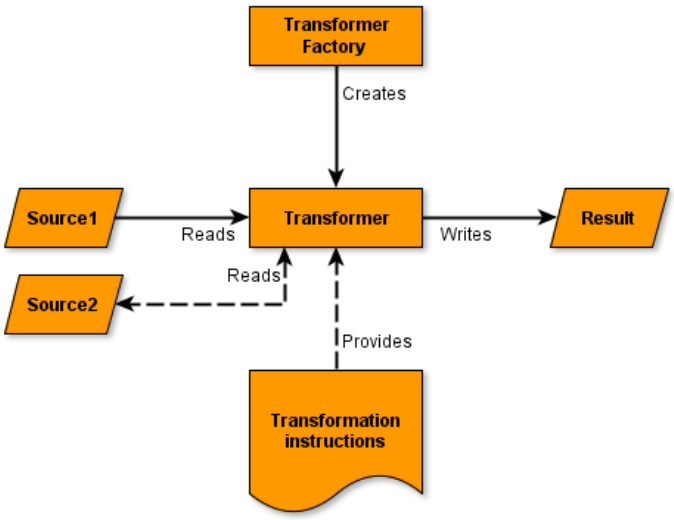

Diagram 2: Merging streams

1.5.2 Splitting result streams

Using the xsl:result-document (XSLT) or stx:result-document (STX) instructions, it is possible to conditionally redirect the output stream to a specific Result class. For instance, during the transformation of a JDBC Source of a table that contains data that is not fully normalized, one could fill one table in the JDBC Result with the primary data of the source, and another table with normalized "lookup data". Also data from one database source can be outputted as multiple smaller XML and/or CSV files.

The attribute *href* must have the following format:

```
result://<resultname>[?param_1=value_1&param_n=value_n]
```

where resultname is the name of a result configured in infofuze-config.xml

XSLT example:

```
<xsl:result-document href="result://my-solrresult"/>
```

STX example:

```
<stx:result-document href="result://my-solrresult"/>
```

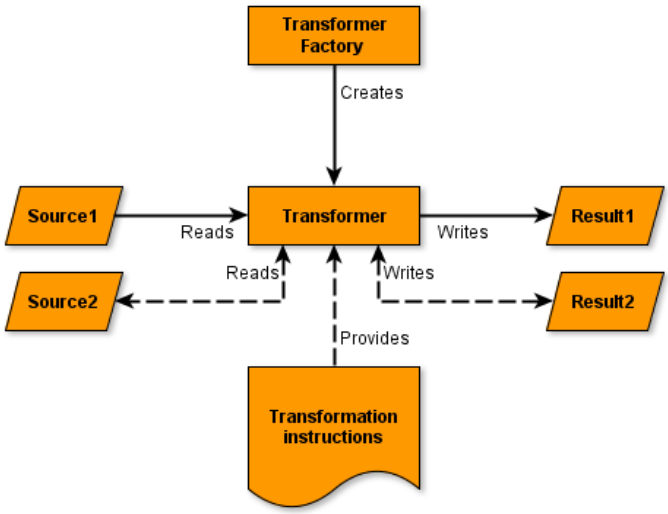

Diagram 3: Merging and splitting streams

1.6 Executing transformations

An Infofuze transformation can be executed in three ways:

1. Use the Infofuze Command Line Interface (CLI).
2. Write (and schedule) an Apache Ant Task.
3. Write your own Java code.

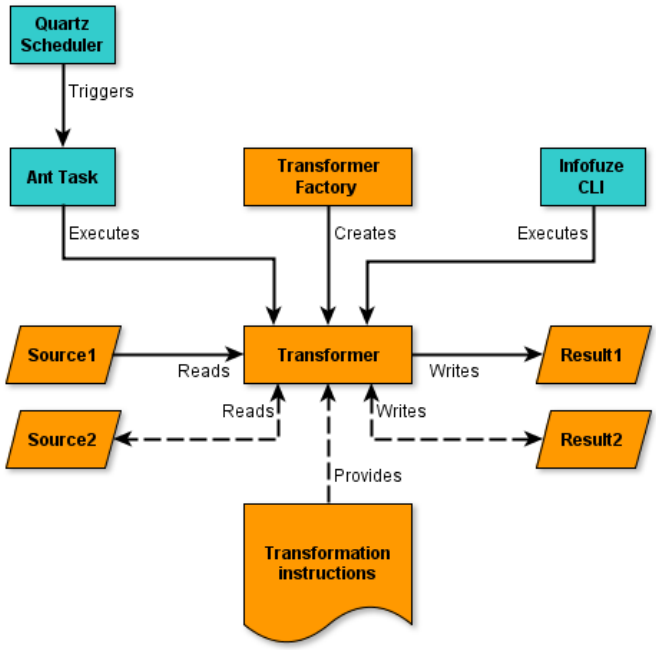

Diagram 4: Executing transformations

1.6.1 Use the Infofuze Command Line Interface (CLI)

The Infofuze command-line interface is implemented in the Java class *com.armatiek.infofuze.cli.Transform*. The interface has the following syntax:

```
Infofuze Transform
==================
usage: transform -s <name> -r <name> [-t <file>] [-m <mode>] [-i <id>]
       [-u <classname>] [-o <classname>] [-x <classname>]

Options:
    -h,--help                              prints this message
    -s,--sourcename <name>                 name of the configured Source
    -r,--resultname <name>                 name of the configured Result
    -t,--transformationfile <file>         path to the transformation file (xsl
                                           or stx) to transform Source to
                                           Result (optional)
    -m,--mode <mode>                       mode of the transformation; "full",
                                           "full_no_delete", "delta" or
                                           "no_index" (optional)
    -i,--transformationid <id>             transformation identifier (optional)
    -u,--uriresolver <classname>           name of URIResolver class (optional)
    -o,--outputuriresolver <classname>     name of OutputURIResolver class
                                           (optional)
    -x,--xsltfactory <classname>           name of TransformerFactory class for
                                           XSLT transformations in context of
                                           STX transformation (optional)
```

See also Deploying Infofuze

1.6.2 Write (and schedule) an Ant Task

Apache Ant is a tool actually designed to automate software build processes. These processes can be defined by writing a build file which contain a set of tasks (instructions) expressed in XML. Besides from invoking the Java compiler, Ant provides a set of other standard tasks that include all file and directory tasks (copying, moving, deleting), archiving tasks (compress and decompress files), logging tasks, mail tasks, remote tasks (sending and receiving files via ftp, scp, execute tasks via ssh), version control tasks (get of commit files from and to CVS or Subversion) and so on. Infofuze provides a custom task *transform* that can be combined with all the standard Ant tasks to define versatile transformation jobs where the actual transformation task can be preceded and/or succeeded by a set of other tasks without any programming. The Ant tasks can be defined in the configuration file tasks.xml.

Using the Terracotta Quartz scheduler, the server process of Infofuze can schedule the execution of Ant tasks and therefore transformation jobs. These executions have their one thread and can be executed in parallel (concurrent). The scheduling of the jobs can be defined in the configuration file jobs.xml. In this configuration file, one can define jobs and triggers. A jobs is the definition of what must be done, a trigger is the definition of when it must be done. In a typical scenario, the job definition will refer to the job class *org.quartz.jobs.AntJob*, which will invoke the execution of an Ant target within tasks.xml. The trigger can be a SimpleTrigger, for a “one-shot” execution of a job on a specific time, or a CronTrigger for job execution based on calendar-like schedules - such as "every Friday, at noon" or "at 10:15 on the 10th day of every month."

1.6.3 Write your own Java code

You can write your own Java code using the Infofuze Java API and JAXP that creates a Transformer, pulls a Source and Result object from the SourcePool en ResultPool and provide the Transformer with an XSLT or STX file. You can also use the class *com.armatiek.infofuze.transformer.Transformer* that does all that work for you (and some more stuff regarding transactioning of the Result classes). You can use that class directly, or use the source as an example. Actually, the command line interface of Infofuze is not more than an thin interface to that class.

1.7 Full and delta transformations

In a lot of scenario's, especially when transforming large amounts of data, it makes sense to not transform data of which is known that is it not changed compared to the data that is already in the data store or index the transformation is writing to. This can avoid a lot of unnecessary network traffic, disk I/O and processing time. Infofuze executes a transformation in one of four modes:

1. **NORMAL**: all data is transformed from the source to the result unconditionally and without any extra operations. This is the default mode.
2. **FULL**: all data is transformed from the source to the result unconditionally. After the transformation the execution date/time is stored of the transformation along with the identifier of the transformation. This transformation identifier can be specified as a parameter on the command line or as a attribute of the Ant task. If no attribute is specified, the name of the target will be used as the identifier.
3. **FULL_WITH_DELETE**: all data is transformed from the source to the result unconditionally and the last transformation date/time of the transformation is stored. Infofuze will try to remove all data (documents, records) from the result (data store or index) that does not exist in the source anymore. Not all Result classes support this mode (but for instance the SolrjResult does).
4. **DELTA**: only data is transformed of which is known that is has changed compared to the data in the data store or index, though no effort will be done to remove data from the data store or index that does not exist in the source anymore. Not all Source classes support this mode.

In some scenario's the mode FULL does perform much better than the mode FULL_WITH_DELETE, and in such situations the scheduling of this mode can be combined with the scheduling of a FULL_WITH_DELETE job at a lower frequency. Furthermore, sometimes the FULL_WITH_DELETE mode is just not feasible because of the sheer amount of data.

The mode can be specified as a parameter on the command line interface or in the attribute "mode" of a transformation task in tasks.xml.

1.8 Configuring Infofuze

Infofuze can be configured by a set of configuration files that are stored under the Infofuze home directory. The most important configuration files are:

- **infofuze-config.xml** : defines the sources and the results
- **jobs.xml** : defines jobs (which execute tasks) and triggers (which trigger jobs)
- **tasks.xml** : defines transformations

The location of the home directory can be specified by JNDI or a Java system property.

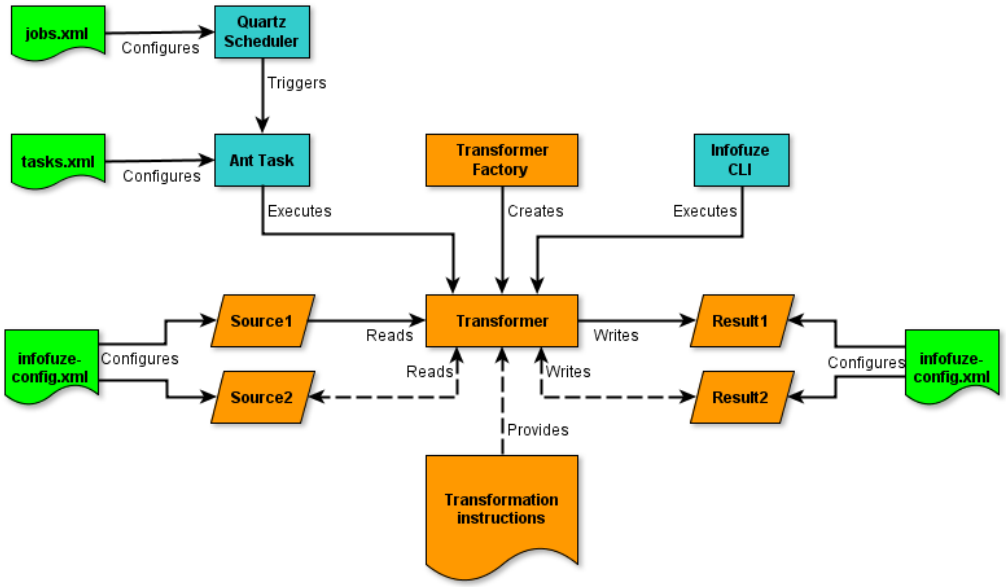

Diagram 5: Configure a transformation

1.8.1 infofuze-config.xml

The configuration file *infofuze-config.xml* contains the definitions of the Source and Result objects. The format of *infofuze-config.xml* is defined by the XML Schema *infofuze-config.xsd*. The configuration file is both used by the server process and when running Infofuze from the command line. A simple configuration file looks like this:

```
<config>
  <sources>
    <localFileSystemSource name="filesystem-network">
      <location>/var/files/myfiles</location>
      <location>/var/files/mydocuments</location>

      <directoryFilter>
        <andFileFilter>
          <hiddenFileFilter>false</hiddenFileFilter>
          <notFileFilter>
            <nameFileFilter>
              <name>.svn</name>
            </nameFileFilter>
          </notFileFilter>
        </andFileFilter>
      </directoryFilter>

      <compressedFileFilters>
        <tarFileFilter>
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.tar</wildcard>
          </wildcardFileFilter>
        </tarFileFilter>
        <zipFileFilter>
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.zip</wildcard>
          </wildcardFileFilter>
        </zipFileFilter>
      </compressedFileFilters>

      <binaryFileFilters>
        <binaryFileFilter
         extractText="true" includeBinary="false" extractMetadata="true" preParse="true">
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.doc</wildcard>
            <wildcard>*.pdf</wildcard>
          <wildcardFileFilter>
        </binaryFileFilter>
      </binaryFileFilters>

      <xmlFileFilters>
        <xmlFileFilter includeBinary="false" preParse="true">
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.xml</wildcard>
            <!-- HTML which will be converted to well-formed XML: -->
            <wildcard>*.htm?</wildcard>
            <wildcard>*.xhtm?</wildcard>
          </wildcardFileFilter>
        </xmlFileFilter>
      </xmlFileFilters>

      <csvFileFilters>
        <csvFileFilter
          includeBinary="false" delimiter="," encapsulator="&quot;"
          ignoreLeadingWhitespace="true" ignoreTrailingWhitespace="true"
          interpretUnicodeEscapes="false" ignoreEmptyLines="true"
          hasHeading="false" defaultCharsetName="UTF-8">
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.csv</wildcard>
          </wildcardFileFilter>
        </csvFileFilter>
      </csvFileFilters>

      <jndiJdbcSource name="database-addresses">
        <jndiDataSource>jdbc/addresses</jndiDataSource>
        <queryFull>select * from addresses</queryFull>
```

```
        <queryDelta>
        select * from addresses where lastmodified > :last_transformed_time
        </queryDelta>
      </jndiJdbcSource>

    </localFileSystemSource>
  </sources>

  <results>
    <solrjResult name="solr-core-filesystem">
      <uri>http://localhost:8080/solr/filesystem</uri>
      <uniqueKeyFieldName>id</uniqueKeyFieldName>
      <sourceFieldName>source</sourceFieldName>
      <defaultFieldName>binary</defaultFieldName>
      <commit>true</commit>
      <optimize>true</optimize>
      <waitFlush>true</waitFlush>
      <waitSearcher>true</waitSearcher>
    </solrjResult>

    <fileResult name="result-debug">
      <path>/var/files/debug/result-debug.xml</path>
      <append>false</append>
    </fileResult>
  </results>

</config>
```

For the filesystem based sources there are file filters available that filter on age, size, hidden status, mimetype, prefix, suffix, wildcard, regular expression and so on. These file filters can be combined within AND and OR filters to construct complex and powerful filters.

1.8.2 jobs.xml

The configuration file jobs.xml contains the definitions of jobs (what task to execute) and triggers (when to execute). The format of jobs.xml is that of the Terracotta Quartz scheduler and defined by the XML Schema *job_scheduling_data_1_8.xsd*. A simple configuration file looks like this:

```
<job-scheduling-data
  xmlns="http://www.quartz-scheduler.org/xml/JobSchedulingData"
  version="1.8">
  <schedule>
    <job>
      <name>job-index-solr</name>
      <job-class>com.armatiek.infofuze.job.AntJob</job-class>
      <job-data-map>
        <entry>
          <key>target</key>
          <value>index-solr</value>
        </entry>
      </job-data-map>
    </job>
    <trigger>
      <cron>
        <name>trigger-index-solr</name>
        <job-name>job-index-solr</job-name>
        <!-- Fire at 10:15am from Monday to Friday: -->
        <cron-expression>0 15 10 ? * MON-FRI</cron-expression>
      </cron>
    </trigger>
  </schedule>
</job-scheduling-data>
```

The configuration file is not used when running Infofuze from the command line, it's only used within the server process

1.8.3 tasks.xml

The configuration file tasks.xml contains the definitions of the transformation tasks. This file is a standard Apache Ant build file in which a custom task *transform* can be used. A simple configuration file looks like this:

```
<project name="InfofuzeTasks" basedir="." default="index-solr">
  <target name="index-solr" description="Creates Solr index of network drive">
    <transform
     source="filesystem-network"
     result="solr-core-filesystem"
     transformation="../stx/filesystem-network.stx"
     mode="delta"/>
  </target>
</project>
```

The source "filesystem-network" and result "solr-core-filesystem" are defined in infofuze-config.xml. These transformation jobs can be executed like any other Ant project directly from the command-line (using Ant itself), but can also be scheduled and executed in the context of the Infofuze server process. The configuration file is not used when running Infofuze from the command line, it's only used within the server process.

2 Source classes

2.1 Filesystem based source classes

The filesystem based source classes include:

- LocalFileSystemSource
- CIFSSource
- WebDAVSource
- WebCrawlSource
- HTTPSource
- FTPSource

The task of these sources is to read (a filtered subset) of files from a filesystem and provide as much structured information as possible to the XML stream. During the traversal of the filesystem, files and directories can be filtered on age, size, hidden status and name (using wildcards, suffix, prefix and regular expressions). Filters can be combined with AND and OR operators to create complex filters. Optionally, files that were not changed since the last traversal can be skipped as well (see section 1.7).

All filesystem based sources can provide for every file (regardless of its type) at least the following XML structure :

```
<file parent="" name="" last-modified="" length="" content-type="" hidden="">
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional -->
</file>
```

- **parent**: the full path of the parent directory
- **name**: the name of the file including the extension
- **last-modified**: the last modification time in ISO 8601 format
- **length**: the file size in bytes
- **content-type**: mime type of the file
- **hidden**: true/false

For specific file types and source classes more data is provided, as described in the following sections.

2.1.1 File types

2.1.1.1 XML files

XML structure

All sources can provide at least the following XML structure to the stream for XML files:

```
<file parent="" name="" last-modified="" length="" content-type="" hidden="">
  <xml>
    <!-- The full XML without prolog in the character encoding of stream -->
  </xml>
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional -->
</file>
```

The XML enclosed by the xml tag does not have to be equal to the binary data within the bin tag. The character encoding of the XML within the xml tag will always be converted to that of the stream and this XML will not contain a prolog (any xml declaration, DOCTYPE statement or other constructs before the start element). For XML that relies on named ENTITY declarations in a Document Type Definition (DTD) a DOCTYPE statement can be configured for the entire stream.

HTML files (both well- and non well formed) can also be defined as XML files. Infofuze will convert any HTML to well formed XML and feed this to the stream. This will enable the following functionality:

- Only specific text can be extracted from the HTML. For instance, only text can be extracted that is part of a particular div element with a specific id of class attribute or occurs between two special comment constructs.
- All hyperlinks and/or image links can be extracted.

Configuration
Below is an example how to filter local XML files that are smaller than 5Mb and stored in the directory /var/files/myfiles. Also HTML files will be included and converted to well formed XML files. The binary file is not written to the stream and the source XML is checked for wellformedness before offering it to the stream.

```
<config>
  <sources>
    <localFileSystemSource name="my-local-source">
      <location>/var/files/myfiles</location>
      <!-- ... -->
      <xmlFileFilters>
        <xmlFileFilter includeBinary="false" preParse="true">
          <andFileFilter>
            <wildcardFileFilter caseSensitive="false">
              <wildcard>*.xml</wildcard>
              <!-- HTML which will be converted to well-formed XML: -->
              <wildcard>*.htm?</wildcard>
              <wildcard>*.xhtm?</wildcard>
            </wildcardFileFilter>
            <!-- must be smaller than 5Mb: -->
            <sizeFileFilter acceptLarger="false">5242880</sizeFileFilter>
          </andFileFilter>
        </xmlFileFilter>
      </xmlFileFilters>
      <!-- ... -->
    </localFileSystemSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.1.2 Binary files

XML structure
Through the use of Apache Tika, all sources can provide at least the following XML structure to the stream for specific binary files:

```
<file parent="" name="" last-modified="" length="" content-type="" hidden="">
  <metadata>
    <meta name="" value=""/>
    [<meta> ... </meta>[<meta> ... </meta>]]
  </metadata> <!-- optional -->
  <text><!-- XHTML representation of the contents --></text> <!-- optional -->
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional -->
</file>
```

The most important binary file types that are supported are (X)HTML, Microsoft Office (Word, Excel and PowerPoint; both OLE2 an OOXML, Visio), OpenDocument/OpenOffice, PDF, E-pub, RTF, Text, Email (mbox) and several audio, video and image formats (see http://tika.apache.org/0.9/formats.html).

Configuration
Below is an example how to filter local MS Word and PDF files whose last modification time is after January 13th eight o' clock PM and stored in the directory /var/files/myfiles. The binary file is not written to the stream, but the metadata is. Also the XHTML representation of the content is checked for wellformedness before offering it to the stream.

```
<config>
  <sources>
    <localFileSystemSource name="my-local-source">
      <location>/var/files/myfiles</location>
      <location>/var/files/mydocuments</location>
      <!-- ... -->
      <binaryFileFilters>
        <binaryFileFilter
         extractText="true" extractMetadata="true" includeBinary="false" preParse="true">
          <andFileFilter>
            <wildcardFileFilter caseSensitive="false">
              <wildcard>*.doc?</wildcard>
              <wildcard>*.pdf</wildcard>
            </wildcardFileFilter>
            <!-- must be newer than specified datetime: -->
            <ageFileFilter acceptOlder="false">2001-01-13T20:00:00</ageFileFilter>
          </andFileFilter>
        </binaryFileFilter>
      </binaryFileFilters>
      <!-- ... -->
    </localFileSystemSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.1.3 JSON files

XML structure

For Javascript Object Notation (JSON) files, Infofuze will convert all data within the JSON file to an XML stream. This XML stream has the a structure that is a literal XML representation of the JSON data. All filesystem based sources can provide at least the following XML structure to the stream for JSON files:

```
<file parent="" name="" last-modified="" length="" content-type="" hidden="">
  <json>
    <!-- XML representation of JSON input stream -->
  </json>
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional -->
</file>
```

Configuration

Below is an example how to filter local JSON files that are stored in the directory /var/files/myfiles.

```
<config>
  <sources>
    <httpSource name="my-http-source">
      <location>http://myserver/myjsonstream</location>
      <jsonFileFilters>
        <jsonFileFilter>
          <trueFileFilter/>
        </jsonFileFilter>
      </jsonFileFilters>
    </httpSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.1.4 CSV files

XML structure

For Comma Separated Values (CSV) files, Infofuze will convert all data within the CSV file to an XML stream. This XML stream has the same structure of that of the JDBC Sources. All filesystem based sources can provide at least the following XML structure to the stream for CSV files:

```
<file parent="" name="" last-modified="" length="" content-type="" hidden="">
  <csv>
    <resultset>
      <row>
        <col name=""></col>  <!-- value of cell of row 1, col 1 -->
        [<col> ... </col>[<col> ... </col>]]
      </row>
      [<row> ... </row>[<row> ... </row>]]
    </resultset>
  </csv>
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional -->
</file>
```

Because CSV is not a well defined standard, several options can be configured for the interpretation of the CSV file, like the delimiter, encapsulator and comment characters, if the first line of the CSV file must be treated as headings and so on.

Configuration
Below is an example how to filter local CSV files that are stored in the directory /var/files/myfiles.

```
<config>
  <sources>
    <localFileSystemSource name="my-local-source">
      <location>/var/files/myfiles</location>
      <!-- ... -->
      <csvFileFilters>
        <csvFileFilter
          includeBinary="false"
          delimiter=","
          encapsulator="&quot;"
          ignoreLeadingWhitespace="true"
          ignoreTrailingWhitespace="true"
          interpretUnicodeEscapes="false"
          ignoreEmptyLines="true"
          hasHeading="false"
          defaultCharsetName="UTF-8">
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.csv</wildcard>
          </wildcardFileFilter>
        </csvFileFilter>
      </csvFileFilters>
      <!-- ... -->
    </localFileSystemSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.1.5 Compressed files

ZIP, TAR, GZIP and BZIP2 files will be treated by Infofuze as directories and will traverse them transparently. This also applies to compressed files that are nested within other compressed files. The filtering of files within compressed files works the same as in normal directories. The XML that is provided to the stream is the same as of uncompressed files.

Configuration
Below is an example how to filter local zip and tar files that are stored in the directory /var/files/myfiles.

```
<config>
  <sources>
    <localFileSystemSource name="my-local-source">
      <location>/var/files/myfiles</location>
      <!-- ... -->
      <compressedFileFilters>
        <tarFileFilter>
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.tar</wildcard>
          </wildcardFileFilter>
        </tarFileFilter>
        <zipFileFilter>
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.zip</wildcard>
          </wildcardFileFilter>
        </zipFileFilter>
      </compressedFileFilters>
    <!-- ... -->
    </localFileSystemSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.1.6 Unparseable files

XML structure

Unparseable files are files of which the content can not or should not be parsed or interpreted. Only the minimal structure that is described in section 2.1 will included in the stream.

Configuration

Below is an example of how to filter unparseable files with the extensions .dat and .bin that are stored in the directory /var/files/myfiles. The binary file content will be included in the XML Stream.

```
<config>
  <sources>
    <localFileSystemSource name="my-local-source">
      <location>/var/files/myfiles</location>
      <!-- ... -->
      <unparseableFileFilter includeBinary="true">
        <wildcardFileFilter caseSensitive="false">
          <wildcard>*.dat</wildcard>
          <wildcard>*.bin</wildcard>
        </wildcardFileFilter>
      </unparseableFileFilter>
      <!-- ... -->
    </localFileSystemSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.2 Filesystem based sources

2.1.2.1 LocalFileSystemSource

The LocalFileSystemSource can be used to stream local files that can be accessed by Java's java.io.File class.

XML structure

The LocalFileSystemSource provides the minimal XML structure that is described in section 2.1.

Configuration

Examples of the configuration of the LocalFileSystemSource are included in the section 2.1.1.

2.1.2.2 CIFSSource

XML structure

The CIFSSource can be used to stream files that can be accessed by the CIFS (Common Internet File System) or SMB (Server Message Block) protocol. This includes files on a Microsoft Windows Network and Unix based systems running Samba. In addition to the minimal XML structure this source provides extra information about the file's creation time, UNC path, security descriptor and share permissions. From the security descriptor the read-permissions (The CIFSSource can provide for every file (regardless of its type) at least the following XML structure to the stream:

```
<file parent="" name="" last-modified="" length="" content-type=""
 hidden="" created="">
  <security>
    <ace is-allow="" is-inherited="" access-mask="" flags="">
      <sid account-name="" domain-name="" rid="" type="" numeric="" display=""/>
    </ace>
    [<ace> ... </ace>[<ace> ... </ace>]]
  </security>
  <share-security>
    <ace is-allow="" is-inherited="" access-mask="" flags="">
      <sid account-name="" domain-name="" rid="" type="" numeric="" display=""/>
    </ace>
    [<ace> ... </ace>[<ace> ... </ace>]]
  </share-security>
  <read-permissions>
    <sid account-name="" domain-name="" rid="" type="" numeric="" display=""/>
    [<sid> ... </sid>[<sid> ... </sid>]]
  </read-permissions>
  <bin><![CDATA[base64 encoded binary file]]></bin> <!-- optional →
  <!-- other data depending on type of file -->
</file>
```

- created: the creation time in ISO 8601 format.
- security: contains the Access Control Entries (ACE) representing the security descriptor associated with this file or directory.
- share-security: contains the Access Control Entries (ACE) representing the share permissions on the share exporting this file or directory.
- sid: the Windows SID (a numeric identifier - Security Identifier - used to represent Windows accounts).
- account-name: the sAMAccountName of this SID unless it could not be resolved in which the element contains the numeric RID.
- domain-name: the domain name of this SID unless it could not be resolved in which case the element contains the numeric representation.
- type: the type of this SID indicating the state or type of account (for instance user (1), domain (3)).
- numeric: the numeric representation of the SID such as S-1-5-21-1496946806-2192648263-3843101252-1029.
- display: the string representing this SID ideal for display to users.

The CIFSSource fully supports XML, CVS, binary and compressed files.

Configuration

Below is an example of the configuration of a CIFSSource. The configuration of CIFSSource supports the following additional properties:

- extractSecurity: extract the security descriptors associated with the files.
- extractShareSecurity: include the share permissions on the share exporting the files.
- ldapSourceRef: a reference to an existing LDAPSource in the configuration that is used to retrieve the group membership of users and groups.

- ldapResultXslPath: a reference to an xsl stylesheet that is used to determine the group membership of users and groups using the LDAPSource defined by *ldapSourceRef.* (see the example *windows2008.xsl*).

```
<config>
  <sources>
    <cifsSource name="my-cifs-source" extractSecurity="true" extractShareSecurity="true"
ldapSourceRef="..." ldapResultXslPath="...">
      <location>smb://user:password@myserver.nl/myshare/files/</location>
      <!-- ... -->
      <binaryFileFilters>
        <binaryFileFilter extractText="true" extractMetadata="true">
          <wildcardFileFilter caseSensitive="false">
            <wildcard>*.doc?</wildcard>
            <wildcard>*.pdf</wildcard>
          </wildcardFileFilter>
        </binaryFileFilter>
      </binaryFileFilters>
      <!-- ... -->
    </cifsSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.2.3 WebDAVSource

XML structure

The WebDAVSource can be used to stream files that are accessible by the WebDAV protocol. This protocol is for instance implemented by the Apache HTTP server module *mod_dav*, Microsoft Internet Information Server 5 (and higher), Microsoft Exchange server and others. The source provides the minimal XML structure that is described in section 2.1, and supports HTTP, HTTPS, proxy access, authentication, non standard ports and fully supports XML, CVS, binary and compressed files. The WebDAVSource uses the WebDAV client library of the Apache Jackrabbit project.

Configuration

Below is an example of a configuration of a WebDAVSource:

```
<config>
  <sources>
    <webDAVSource
      name="my-webdav-source"
      username="john"
      password="secret"
      timeout="5000"
      <location>http://localhost/webdav/</location>
      <directoryFilter>
        <hiddenFileFilter>false</hiddenFileFilter>
      </directoryFilter>
      <xmlFileFilters>
        <xmlFileFilter includeBinary="false">
          <andFileFilter>
            <wildcardFileFilter caseSensitive="false">
              <wildcard>*.xml</wildcard>
            </wildcardFileFilter>
            <!-- must be smaller than 5Mb: -->
            <sizeFileFilter acceptLarger="false">5242880</sizeFileFilter>
          </andFileFilter>
        </xmlFileFilter>
      </xmlFileFilters>
    </webDAVSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.2.4 WebCrawlSource

XML structure

The WebCrawlSource can be used to *crawl* or *spider* through all or a subset of the pages and files given one or more so called seed URLs of websites. The source provides the minimal XML structure that is described in section 2.1 but includes an extra attribute *uri* to the element *file*. It supports HTTP, HTTPS, proxy access, authentication, non standard ports and fully supports XML, CVS, binary and compressed files. The crawler has a number of other options that can be configured:

- Wait time (in ms) between requests (to avoid overloading webservers).
- Time out of the HTTP requests
- Whether or not to follow links to images, scripts and css stylesheets
- Maximum depth to crawl

The crawler supports most of the redirect status codes. It performs extensive URL normalization to avoid multiple transformations of the same page (but with different URLs). It does not yet support the Robots Exclusion Standard (robots.txt) but does support the robots meta tag (specifically the values NOINDEX and NOFOLLOW). The crawler does not follow external links (links to other hosts than the seed URL) and does not follow links that are higher in the URL hierarchy than the seed URL.

Configuration

Below is the configuration of a WebCrawlSource:

```
<config>
  <sources>
    <webCrawlSource
      name="my-webcrawl-source"
      timeout="5000"
      maxDepth="5"
      wait="200"
      userAgent="Mozilla/5.0 (compatible; Infofuze/1.0; Windows NT 5.1)"
      followImages="true"
      followScripts="false"
      followLinks="false">
      <location>http://www.armatiek.nl</location>
      <binaryFileFilters>
        <binaryFileFilter extractText="true" extractMetadata="true" preParse="true">
          <mimeTypeFileFilter>
            <mimeType>application/msword</mimeType>
            <mimeType>application/pdf</mimeType>
          </mimeTypeFileFilter>
        </binaryFileFilter>
      </binaryFileFilters>
    </webCrawlSource>
  </sources>
  <!-- ... -->
</config>
```

2.1.2.5 FTPSource

XML structure

The FTPSource can be used to stream files that are accessible by the FTP protocol. The FTPSource provides the minimal XML structure that is described in section 2.1, and supports FTP, FTPS, proxy access, authentication, non standard ports and fully supports XML, CVS, binary and compressed files.

Configuration

{To be written}

2.2 JDBC based sources

XML structure

The JDBC based sources can be used to stream data from a relational database using a JDBC or ODBC driver (via Java's JDBC-ODBC bridge). Infofuze provides plain JDBC connections, but also supports connection pooling for drivers that implement ConnectionPoolDataSource and PooledConnection, or the connection pooling of the Java Application Server via JNDI (like Tomcat or JBoss). JDBCSources can be read parameterized, in which case the source is read from within a STX or XSLT transformation and a parameterize query is executed to obtain additional data.

The XML that is provided to the stream has the following structure:

```
<resultset>
  <row>
    <col name=""></col>  <!-- value of cell of row 1, col 1 -->
    [<col> ... </col>[<col> ... </col>]]
  </row>
  [<row> ... </row>[<row> ... </row>]]
</resultset>
```

2.2.1 DirectJDBCSource

The DirectJDBCSource can be used to connect to a database using a direct unpooled JDBC connection providing the following configuration properties:

- The JDBC driver class name
- The JDBC connection string
- The username to connect to the database (optional)
- The password to connect to the database (optional)
- The SQL query that is used to query the data to transform
- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration

Below is an example of a DirectJDBCSource:

```
<config>
  <sources>
    <directJdbcSource name="my-directjdbc">
      <driver>com.mysql.jdbc.Driver</driver>
      <url>jdbc:mysql://localhost:3306/database</url>
      <username>user</username>
      <password>password</password>
      <queryFull>select * from addresses</queryFull>
    </directJdbcSource>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed outside the context of a Java application server and the source is the primary source of the transformation and only one connection is used during the entire transformation.

2.2.2 PooledJDBCSource

The PooledJDBCSource can be used to connect to a database using a connectionpool. This source can only be used for JDBC drivers that provide implementations of Java's ConnectionPoolDataSource and PooledConnection (like the ones for Oracle, Microsoft SQL Server and MySql). The following configuration properties must be provided:

- A piece of Javascript code that creates and initializes the datasource
- The SQL query that is used to query the data to transform

- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration
Below is an example of a PooledJDBCSource:

```
<config>
  <sources>
    <pooledJdbcSource name="my-pooledjdbc">
      <connectionpoolDataSource>
        var dataSource = new org.h2.jdbcx.JdbcDataSource();
        dataSource.setURL("jdbc:h2:/database");
        dataSource.setUser("sa");
        dataSource.setPassword("");
      </connectionpoolDataSource>
      <queryFull>select * from addresses</queryFull>
    </pooledJdbcSource>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed outside the context of a Java application server and the source is used as a secondary source from within the transformation and multiple connections are necessary during the transformation.

2.2.3 JNDIJDBCSource

The JNDIJDBCSource can be used to connect to a database using a datasource that is configured in a Java application server with JNDI. The following configuration properties must be provided:

- The name of the JNDI datasource that is configured in the Java application server
- The SQL query that is used to query the data to transform
- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration
Below is an example of a JNDIJDBCSource:

```
<config>
  <sources>
    <jndiJdbcSource name="my-jndijdbc">
      <jndiDataSource>jdbc/mydatabase</jndiDataSource>
      <queryFull>select * from addresses</queryFull>
      <queryDelta>
      select * from addresses where lastmodified > :last_transformed_time
      </queryDelta>
    </jndiJdbcSource>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed within the context of a Java application server and the source is used as a secondary source from within the transformation and multiple connections are necessary during the transformation.

2.3 SolrSource

The SolrSource can be used to connect to a Apache Solr core and get the XML stream that is the result of executing a Solr query. The following configuration properties can be provided:

- The uri of the Solr core (required).
- The name of the unique field as defined in the Solr schema (default “id”).
- The name of the field that contains the complete source XML (default “source”).

- The name of the default field as defined in the Solr schema (default “binary”).
- The name of the Solr request (default “select”).
- The user name (optional, in case the requests must be authenticated)
- The password (optional, in case the requests must be authenticated).
- The proxy host (optional).
- The proxy port (optional).
- A timeout of the request to Solr (default -1, meaning no timeout).

Configuration
Below is an example of a SolrSource:

```
<config>
  <sources>
    <solrSource name="my-solr-source">
      <uri>http://localhost:8080/solr/mycore</uri>
      <uniqueKeyFieldName>id</uniqueKeyFieldName>
      <sourceFieldName>xml</sourceFieldName>
      <defaultFieldName>text</defaultFieldName>
    </solrSource>
  </sources>
  <!-- ... -->
</config>
```

2.4 MongoDBSource

The MongoDBSource can be used to connect to a MongDB database and get a XML stream representation of the JSON that is the result a query on the MongoDB database. The following configuration properties can be provided:

- The uri of the MongoDB database (required).
- The database name (required).
- The collection name (required).
- The user name (optional, in case authentication is configured).
- The password (optional, in case authentication is configured).
- The limit (maximum number of results) (optional, default is all).
- The sort method (optional, default is no sorting).
- The query to execute (optional, but left out, it must specified when using the MongoDB source in XSL or XSLT).

Configuration
Below is an example of a MongoDBSource:

```
<config>
  <sources>
    <mongoDBSource name="my-mongodb-source">
      <uri>mongodb://localhostt</uri>
      <uniqueKeyFieldName>myid</uniqueKeyFieldName>
      <database>mydatabase</database>
      <collection>mycollection</collection>
      <queryFull>{ }</queryFull>
      <limit>1</limit>
      <sort>{ creationdatetime : 1 }</sort>
    </mongoDBSource>
  </sources>
  <!-- ... -->
</config>
```

2.5 HTTPSource

The HTTPSource can be used to stream the result of a HTTP POST or GET request and is particularly suited to stream the result of a SOAP request to a webservice. In this scenario the source is not used as the primary source of a transformation, but read during a transformation using XPath's document() function (XSLT) or the stx:process-document instruction (STX).

Configuration
{To be written}

2.6 LDAPSource

XML structure
The LDAPSource can be used to stream the result of a search operation on a directory service using the Lightweight Directory Access Protocol (LDAP). Examples of directory services that can be searched using LDAP are the Microsoft Active Directory, Novell eDirectory, Apache Directory Service (ApacheDS), Oracle Internet Directory (OID), and slapd, part of OpenLDAP.

The result of a search operation is written to the stream in the Directory Services Markup Language (DSML) format. Below is an example of such a result:

```
<batchResponse xmlns="urn:oasis:names:tc:DSML:2:0:core">
  <searchResponse>
    <searchResultEntry dn="CN=jdoe,CN=Users,DC=corp,DC=armatiek,DC=com">
      <attr name="memberOf">
        <value>Administrators</value>
        <value>Developers</value>
      </attr>
      <attr name="sAMAccountName">
        <value>jdoe</value>
      </attr>
    </searchResultEntry>
    <searchResultDone>
      <resultCode code="0"/>
    </searchResultDone>
  </searchResponse>
</batchResponse>
```

Configuration
Below is an example of the configuration of a LDAPSource:

```
<ldapSource name="source-ldap-group-membership-jdoe">
  <host>localhost</host>
  <bindDN>CN=Administrator,CN=Users,DC=corp,DC=armatiek,DC=com</bindDN>
  <bindPassword>mypassword</bindPassword>
  <baseDN>DC=corp,DC=armatiek,DC=com</baseDN>
  <scope>sub</scope>
  <filter>(&amp;(objectCategory=user)(sAMAccountName=jdoe))</filter>
  <attributes>
    <attribute>memberOf</attribute>
    <attribute>sAMAccountName</attribute>
  </attributes>
</ldapSource>
```

When an LDAPSource is read using the using XPath's document() function (XSLT) or the stx:process-document instruction (STX), the properties *baseDN*, *scope*, *filter* and *attributes* can be specified as parameters in the query string and have precedence over any of these properties that are defined in the configuration.

2.7 NullSource

The NullSource provides an empty XML stream. This source is particularly useful in a scenario where all data is read using XPath's document() function (XSLT) or the stx:process-document instruction (STX) and the primary source of the transformation has no purpose.

Configuration

Below is en example of the configuration of a NullSource:

```
<config>
  <sources>
    <nullSource name="my-null-source"/>
  </sources>
  <!-- ... -->
</config>
```

3 Result classes

Result classes are classes that implement the interface javax.xml.transform.Result. Their task is to write the resulting stream of the transformation to a particular datastore or index. The following result classes are provided by Infofuze:

3.1 FileResult

The FileResult can be used to write the result of a transformation to a local file. Below is an example of the configuration of a FileResult in infofuze-config.xml:

```
<config>
  <!-- ... -->
  <results>
    <fileResult name="my-file-result">
      <path>/tmp/debug-result.xml</path>
      <append>false</append>
    </fileResult>
  </results>
</config>
```

3.2 SolrjResult

The SolrJResult can be used to write XML data that complies with the Apache Solr update format to a Solr instance using the Solrj client library. The Solr update format has the following form:

```
<add>
  <doc>
    <field name="employeeId">05991</field>
    <field name="office">Bridgewater</field>
    <field name="skills">Perl</field>
    <field name="skills">Java</field>
  </doc>
  [<doc> ... </doc>[<doc> ... </doc>]]
</add>
```

Below is an example of the configuration a SolrjResult in infofuze-config.xml:

```
<config>
  <!-- ... -->
  <results>
    <solrjResult name="my-solrj-result">
      <uri>http://localhost:8080/solr/demo</uri>
      <uniqueKeyFieldName>id</uniqueKeyFieldName>
      <sourceFieldName>source</sourceFieldName>
      <defaultFieldName>binary</defaultFieldName>
      <commit>true</commit>
      <optimize>true</optimize>
      <waitFlush>true</waitFlush>
      <waitSearcher>true</waitSearcher>
    </solrjResult>
  </results>
</config>
```

- uri: the uri of the Solr Server (including any specific core)
- uniqueKeyFieldName: the name of the field in the index which should be unique for all documents.
- sourceFieldName: the name of the field in which the name of the source of the transformation is stored.
- defaultFieldName: currently not used.
- commit: whether to commit the changes after the transformation is finished

- optimize: whether to optimize the index (merge all segments into one) after the transformation is finished.
- waitFlush: whether to wait for the indexed data to flush to disk after the transformation is finished.
- waitSearcher: whether to wait for a new Solr searcher to be ready to respond to changes after the transformation is finished.

3.3 MongoDBResult

The MongDBResult can be used to write XML data that complies to a specific format to a MongDB database using the MongoDB Java driver. This XML data format must have the following form:

```
<add>
  <doc>
    <field name="employeeid" type="xsd:string">05991</field>
    <field name="name" type="xsd:string">John Doe</field>
    <field name="characteristics">
      <subdoc>
        <field name="age" type="xsd:integer">25</field>
        <field name="skills" type="xsd:string">
          <array>
            <value>Perl</value>
            <value>Java</value>
          </array>
        </field>
      </subdoc>
    </field>
  </doc>
  [<doc> ... </doc>[<doc> ... </doc>]]
</add>
```

Below is an example of the configuration a MongoDBResult in infofuze-config.xml:

```
<config>
  <!-- ... -->
  <results>
    <mongoDBResult name="my-mongodb-result">
      <uri>http://localhost</uri>
      <uniqueKeyFieldName>employeeid</uniqueKeyFieldName>
      <database>mydatabase</database>
      <collection>mycollection</collection>
    </mongoDBResult>
  </results>
</config>
```

- uri: the uri of the MongoDB database
- uniqueKeyFieldName: the name of the field in the index which should be unique for all documents (optional). If this field is specified, Infofuze will always perform an *upsert* instead of a *insert*.
- database: the name of the MongoDB database (required).
- collection: the name of the collection within the database (required).
- username: the username to use when authentication is required (optional).
- password: the password to use when authentication is required (optional).

3.4 JDBC based results

XML structure

The JDBC based results can be used to insert or update records in a relational database using a JDBC or ODBC driver (via Java's JDBC-ODBC bridge). Infofuze provides plain JDBC connections, but also supports connection pooling for drivers that implement ConnectionPoolDataSource and PooledConnection, or the connection pooling of the Java Application Server via JNDI (like Tomcat or JBoss).

The result of the transformation must have the following structure:

```
<resultset>
  <row>
    <col name=""></col>  <!-- value of cell of row 1, col 1 -->
    [<col> ... </col>[<col> ... </col>]]
  </row>
  [<row> ... </row>[<row> ... </row>]]
</resultset>
```

3.4.1 DirectJDBCResult

The DirectJDBCResult can be used to connect to a database using a direct unpooled JDBC connection providing the following configuration properties:

- The JDBC driver class name
- The JDBC connection string
- The username to connect to the database (optional)
- The password to connect to the database (optional)
- The SQL query that is used to query the data to transform
- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration
Below is an example of a DirectJDBCResult:

```
<config>
  <sources>
    <directJdbcResult name="my-directjdbc">
      <driver>com.mysql.jdbc.Driver</driver>
      <url>jdbc:mysql://localhost:3306/database</url>
      <username>user</username>
      <password>password</password>
      <updateQuery>
        UPDATE phone_book SET number = :number WHERE name = :name
      </updateQuery>
      <insertQuery>
        INSERT INTO phone_book (name, number) VALUES (:name, :number)
      </insertQuery>
      <parameters>
        <parameter name="name" type="string"/>
        <parameter name="number" type="string"/>
      </parameters>
    </directJdbcResult>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed outside the context of a Java application server.

3.4.2 PooledJDBCResult

The PooledJDBCResult can be used to connect to a database using a connectionpool. This source can only be used for JDBC drivers that provide implementations of Java's ConnectionPoolDataSource and PooledConnection (like the ones for Oracle, Microsoft SQL Server and MySql). The following configuration properties must be provided:

- A piece of Javascript code that creates and initializes the datasource
- The SQL query that is used to query the data to transform
- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration
Below is an example of a PooledJDBCResult:

```
<config>
  <sources>
    <pooledJdbcResult name="my-pooledjdbc">
      <connectionpoolDataSource>
        var dataSource = new org.h2.jdbcx.JdbcDataSource();
        dataSource.setURL("jdbc:h2:/database");
        dataSource.setUser("sa");
        dataSource.setPassword("");
      </connectionpoolDataSource>
      <updateQuery>
        UPDATE phone_book SET number = :number WHERE name = :name
      </updateQuery>
      <insertQuery>
        INSERT INTO phone_book (name, number) VALUES (:name, :number)
      </insertQuery>
      <parameters>
        <parameter name="name" type="string"/>
        <parameter name="number" type="string"/>
      </parameters>
    </pooledJdbcResult>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed outside the context of a Java application server.

3.4.3 JNDIJDBCResult

The JNDIJDBCResult can be used to connect to a database using a datasource that is configured in a Java application server with JNDI. The following configuration properties must be provided:

- The name of the JNDI datasource that is configured in the Java application server
- The SQL query that is used to query the data to transform
- The SQL query that is used to query the data that has changed since a specific date/time (optional)

Configuration
Below is an example of a JNDIJDBCSource:

```
<config>
  <sources>
    <jndiJdbcResult name="my-jndijdbc">
      <jndiDataSource>jdbc/mydatabase</jndiDataSource>
      <updateQuery>
        UPDATE phone_book SET number = :number WHERE name = :name
      </updateQuery>
      <insertQuery>
        INSERT INTO phone_book (name, number) VALUES (:name, :number)
      </insertQuery>
      <parameters>
        <parameter name="name" type="string"/>
        <parameter name="number" type="string"/>
      </parameters>
    </jndiJdbcResult>
  </sources>
  <!-- ... -->
</config>
```

This source class is particularly useful in scenario's where the transformation is executed within the context of a Java application server.

3.5 JDBCResult

The JDBCResult classes () can be used to write XML data to a table in a relational database using a JDBC or ODBC driver (using Java's JDBC-ODBC bridge). The same connection pooling mechanisms can be used as with the JDBC source classes. The transformed XML data must have the following form:

```
<resultset>
  <metadata>
    <col name="" type=""/>
    [<col name="" type=""/>[<col name="" type=""/>]]
  </metadata>
  <rowset>
    <row>
      <col name=""></col>  <!-- value of cell of row 1, col 1 -->
      [<col name=""></col>[<col name=""></col>]]
    </row>
    [<row> ... </row>[<row> ... </row>]]
  </rowset>
</resultset>
```

The configuratioBelow is an example of the configuration a JDBCResult in infofuze-config.xml:

```
<config>
  <!-- ... -->
  <results>
    <solrjResult name="my-solrj-result">
      <uri>http://localhost:8080/solr/demo</uri>
      <uniqueKeyFieldName>id</uniqueKeyFieldName>
      <sourceFieldName>source</sourceFieldName>
      <defaultFieldName>binary</defaultFieldName>
      <commit>true</commit>
      <optimize>true</optimize>
      <waitFlush>true</waitFlush>
      <waitSearcher>true</waitSearcher>
    </solrjResult>
  </results>
</config>
```

3.6 HTTPResult

{To be written}

3.7 NullResult

Output that is written to a NullResult has no destination and is the equivalent of the famous */dev/null*. All bytes or characters are ignored and lost in cyberspace. This result is particularly useful in a scenario where all data is written using the xsl:result-document or stx:result-document instructions and the primary result has no purpose.

Below is an example of the configuration a NullResult in infofuze-config.xml:

```
<config>
  <!-- ... -->
  <results>
    <nullResult name="my-null-result"/>
  </results>
</config>
```

4 Third party components

The following excellent 3rd party open source libraries are used by Infofuze:

- Apache Ant: build tool
- Apache Commons
  - CLI: command line argument parser
  - Codec: library of common encoders and decoders
  - Collections: library of container classes like maps, lists and sets.
  - Compress: library for working with ar, cpio, tar, zip, gzip and bzip2 streams
  - CSV: library for working with Comma Separated Values streams
  - Discovery: library for discovering, or finding, implementations for pluggable interfaces
  - Exec: library for executing external processes from Java
  - HttpClient: HTTP client library
  - IO: library of utilities to assist with developing IO functionality
  - Lang: extra methods for Java core classes
  - Logging: logging framework
- Apache JackrabbitWebDAV client
- Apache Lucene: full text search engine library
- Apache Log4J: logging framework
- Apache Maven: build tool
- Apache Solr and Solrj: enterprise search platform
- Apache Tika: toolkit for detecting and extracting metadata and structured text from files
- Apache Xalan: XSLT processor (XSLT version 1.0)
- Apache Xerces: XML parser
- iHarder.net Base64: Base 64 encoding en decoding library
- JCIFS: CIFS/SMB networking protocol client
- Joost: STX processor
- MiniPoolConnectionManager: JDBC connection pool
- Quartz Scheduler: job scheduling service
- Saxon: XQuery and XSLT processor (XSLT version 2.0)
- Slf4J: logging framework
- UnboundID: LDAP client
- URLRewrite: URL rewriting library
- Waffle: Windows Authentication Framework

5 Building Infofuze

After checking out the sourcecode from the Subversion repository at:

```
https://infofuze.svn.sourceforge.net/svnroot/infofuze
```

Infofuze can be build using Maven. Before Infofuze can be build three libraries has to be installed in your local Maven repository because the versions in the central Maven repository are way too old or do not exist. These two libraries are Saxon (HE), JCIFS and Waffle.

Saxon HE can be installed using the following two commands:

```
mvn install:install-file -DgroupId=net.sf.saxon -DartifactId=saxonhe -Dversion=9.3.0_2j
-Dpackaging=pom -Dfile=saxonhe-9.3.0_2j.pom

mvn install:install-file -DgroupId=net.sf.saxon -DartifactId=saxonhe -Dversion=9.3.0_2j
-Dpackaging=jar -Dfile=saxonhe-9.3.0_2j.jar
```

JCIFS can be installed using the following two commands:

```
mvn install:install-file -DgroupId=org.samba.jcifs -DartifactId=jcifs -Dversion=1.3.15
-Dpackaging=pom -Dfile=jcifs-1.3.15.pom

mvn install:install-file -DgroupId=org.samba.jcifs -DartifactId=jcifs -Dversion=1.3.15
-Dpackaging=jar -Dfile=jcifs-1.3.15.jar
```

Waffle can be installed using the following commands:

```
mvn install:install-file -DgroupId=net.java.dev.jna -DartifactId=jna -Dversion=3.2.7
-Dpackaging=pom -Dfile=jna-3.2.7.pom

mvn install:install-file -DgroupId=net.java.dev.jna -DartifactId=jna -Dversion=3.2.7
-Dpackaging=jar -Dfile=jna.jar

mvn install:install-file -DgroupId=net.java.dev.jna -DartifactId=platform -Dversion=3.2.7
-Dpackaging=pom -Dfile=platform-3.2.7.pom

mvn install:install-file -DgroupId=net.java.dev.jna -DartifactId=platform -Dversion=3.2.7
-Dpackaging=jar -Dfile=platform.jar

mvn install:install-file -DgroupId=waffle-jna -DartifactId=waffle-jna -Dversion=1.3
-Dpackaging=pom -Dfile=waffle-jna-1.3.pom

mvn install:install-file -DgroupId=waffle-jna -DartifactId=waffle-jna -Dversion=1.3
-Dpackaging=jar -Dfile=waffle-jna.jar
```

The pom files can be found under the directory *<infofuze>/trunk/local-repository* and the jar files can be downloaded from http://saxon.sourceforge.net/ and http://jcifs.samba.org/ and http://waffle.codeplex.com/ . For newer versions the commands and the pom files must be changed along with the version numbers of the artifacts *jcifs*, *saxonhe* and *waffle-jna* in the master pom.

After installing the two libraries in your local Maven repository the sources can be build running the command-line:

```
mvn clean install
```

in your local subversion working directory *<infofuze>/trunk.*

6 Deploying Infofuze

After building Infofuze with Maven three .jar and two .war files are generated:

- infofuze-core-X.X-SNAPSHOT.jar
- infofuze-cli-X.X-SNAPSHOT-executable.jar
- infofuze-web-core-X.X-SNAPSHOT.jar
- infofuze-web-backend-X.X-SNAPSHOT.war
- infofuze-web-frontend-X.X-SNAPSHOT.war

(X.X is the version number).

6.1 infofuze-core jar

This jar contains the core classes of Infofuze and is used by all the other artifacts. This library is a "J2SE only" jar; there are no dependencies between this jar and any J2EE application server specific classes.

6.2 infofuze-cli jar

This is an runnable "uber jar" contaning all 3rd party dependencies and the infofuze-core jar. The jar has the command line interface that is described in section 1.6.1. When running this jar two system properties must be specified: *infofuze.infofuze.home* pointing to the infofuze home directory containing the configuration files and a JCIFS specific system property *java.protocol.handler.pkgs* with the value *jcifs.*

The following is a valid transformation command when run from the directory *<infofuze>/trunk/infofuze-cli* and java version 1.6:

```
java -jar -Dinfofuze.infofuze.home=..\infofuze-home -Djava.protocol.handler.pkgs=jcifs
-Xmx512m ./target/infofuze-cli-0.1-SNAPSHOT-executable.jar -s my-source -r my-result
```

providing the configuration file contains a definition for a source *my-source* and a result *my-result*.

6.3 infofuze-web-core jar

This jar contains all the core classes that have dependencies with J2EE application server specific classes like the abstract servlet and web application listener classes.

6.4 infofuze-web-backend war

This war is the web application in which the scheduler is started and the transformation jobs are executed. The war should be to be deployed to any Java application server of servlet container but so far it only has been tested on Apache Tomcat version 6.0.32. The following steps are necessary to deploy the war to Tomcat 6.0.32:

- Download the latest binary version of Apache Xalan, and extract the libraries *resolver.jar*, *serializer.jar*, *xercesImpl.jar*, *xml-apis.jar* from the download. Create a new directory *<tomcat>/endorsed* and copy the libraries to this directory. This is necessary to make sure that this version of Xalan and Xerces are used by the application and not the version that is part of the Java JRE.
- Download the latest binary version of JCIFS, and extract the library *jcifs-X.X.X.jar* from the download. Create a new directory *<tomcat>/server-lib* and copy the library to this directory.

- Download binary version 1.3 of Waffle (Windows Authentication Framework) and extract waffle-jna.jar, commons-logging-1.1.1.jar, jna.jar and platform.jar to *<tomcat>/lib.*
- In *<tomcat>/conf/catalina.properties*:
  - specify: *server.loader=${catalina.home}/server-lib/*.jar*
  - Add to the property *common.loader* the classpath to the *infofuze-home* directory, for instance:
    *common.loader=**${catalina.base}/../infofuze-home,** ${catalina.base}/lib, ${catalina.base}/lib/*.jar, ${catalina.home}/lib,${catalina.home}/lib/*.jar*
- Add the following configuration XML within the Engine tag after the Host tag:

```
<Context path="/infofuze" docBase="infofuze"/>
<Context path="/solr" docBase="solr">
 <Valve className="org.apache.catalina.valves.RemoteAddrValve"
allow="127\.0\.0\.1"/>
</Context>
```

- TODO: URIEncoding="UTF-8" in server.xml
- Add a text file *setenv.bat* (Windows) or *setenv.sh* (Unix/Linux) to the directory *<tomcat>/bin* (make sure the file *setenv.sh* has the proper execute rights under Unix/Linux). The contents of the file should be:
  - On Windows:
    set JAVA_OPTS=-server -Xmx256m -Dinfofuze.infofuze.home=..\..\infofuze-home -Djava.protocol.handler.pkgs=jcifs -Dlog4j.configuration=config/log4j.properties
  - On Unix/Linux:
    export JAVA_OPTS="-server -Xmx256m -Dinfofuze.infofuze.home=../../infofuze-home -Djava.protocol.handler.pkgs=jcifs -Dlog4j.configuration=config/log4j.properties"

  Alter the maximum amount of memory and/or the path of the *infofuze-home* to your situation. In this example the infofuze home directory is a sibling directory of the tomcat base directory.
- Optionally, edit the file *<infofuze-home>/config/log4j.properties* to change the location of the log file.

Tomcat can now be started by running the command:

catalina start

in the directory *<tomcat>/bin.*

6.5 infofuze-web-frontend war

This war is not yet in use.