XSLT: Using External Functions

Edin Pezerovic
Vienna University of Economics and Business Administration

June 13, 2007

Bachelor Course Paper
Department of Business Informatics
Prof. Dr. Rony G. Flatscher

Table of Content

1 Introduction

The introduction gives a short overview of the intention of the bachelor thesis and delimits its problem area.

1.1 Abstract

Nowadays most data in IT business is exchanged through XML files. The structure of these files is often not suitable for processing on both sides. Therefore an intermediary step between the exchange of data and the processing has to be introduced. This step is called transformation which changes the structure of the underlying XML file.

The XSL-Transformation (XSLT) provides this kind of functionality. The Apache Software Foundation has implemented an open source XSLT project called Xalan which additionally offers the possibility to call external functions if the specified functionality of XSLT does not suffice.

External functions can be programmed in various scripting languages using the Bean Scripting Framework of Apache as intermediary between the transformer and the scripting code.

This bachelor thesis focuses on the implementation of such external functions. The main point is to show how different business requirements can be solved through calls to external functions within the process of the transformation.

It is not intended to implement production ready programs, as this would expand the size of the bachelor thesis extremely. Expressing the shortage of these examples they are called nutshells.

1.2 Keywords

Bean Scripting Framework, BSF, BSF4Rexx, EXSLT, External Functions, JavaScript, JRuby, JudoScript, OORexx, Transformation, Xalan, XSLT

2 Description of the Technologies Used

The chapter presents the scripting languages used within the nutshells. A short introduction followed by the installation instructions is given. Furthermore the XSL Transformation is introduced.

2.1 The Extensible Stylesheet Language (XSL)

2.1.1 Overview

> "*The extensible stylesheet language is a family of recommendations for defining XML document transformation and presentation.*"[W3C03b]

The XSL family is part of the W3C XML Activity [W3C03a] which has the aim to develop and maintain the XML specification. XSL consists of three parts:

- XML Path Language (XPath)
- XSL Transformations (XSLT)
- XSL Formatting Objects (XSL-FO)

2.1.2 XML Path Language

> "*XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations and XPointer. The primary purpose of XPath is to address parts of an XML document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans.*"[W3C03d]
>
> *"XML Pointer Language (XPointer) supports addressing into the internal structures of XML documents." [W3C03e]*

With XPath it is possible to locate a node (element or attribute) within a XML document. This is accomplished by using the XPath syntax for locating nodes. XPath defines a context node which presents the node where the process of locating starts.

The examples below show location paths using XPath syntax:

- *child::para* selects the *para* element children of the context node
- *attribute::name* selects the *name* attribute of the context node
- /child::doc/child::chapter selects the *chapter* element children of the *doc* (top) node

Facilitating the expressions, XPath provides an abbreviated syntax for locating nodes. Following examples show the same location paths as the above ones by using the abbreviated syntax:

- *para* selects the *para* element children of the context node
- *@name* selects the *name* attribute of the context node
- */doc/chapter* selects the *chapter* element children of the *doc* (top) node

XPath provides functions for manipulating strings, numbers and booleans. Below some examples of the specified functions:

- *concat(...):* returns the concatenation of the arguments
- *substring(...):* returns a substring of the first argument
- *not(...):* returns true if the argument is false and false otherwise
- *string(...):* converts an object to a string
- *number(...):* converts an object to a number
- *round(...):* rounds a number to the closest integer

The complete XPath specification can be found on the W3C homepage [W3C03d].

2.1.3 XSL Transformations

> "*A transformation expressed in XSLT describes rules for transforming a source tree into a result tree.*"[W3C03b]

The main purpose of XSLT is the transformation of an input XML file to an output XML file with a different structure. If transformations are expressed using XSLT they are called stylesheets.

> "*XSL stylesheets are written in the XSLT language. An XSL stylesheet contains instructions for transforming XML documents into XML, HTML, XHTML or plain text. In structural terms, an XSL stylesheet specifies the transformation of one tree of nodes (the XML input) into another tree of nodes (the output or transformation result).*"[AXJ07a]

The following figure shows the transformation process of the XML tree:

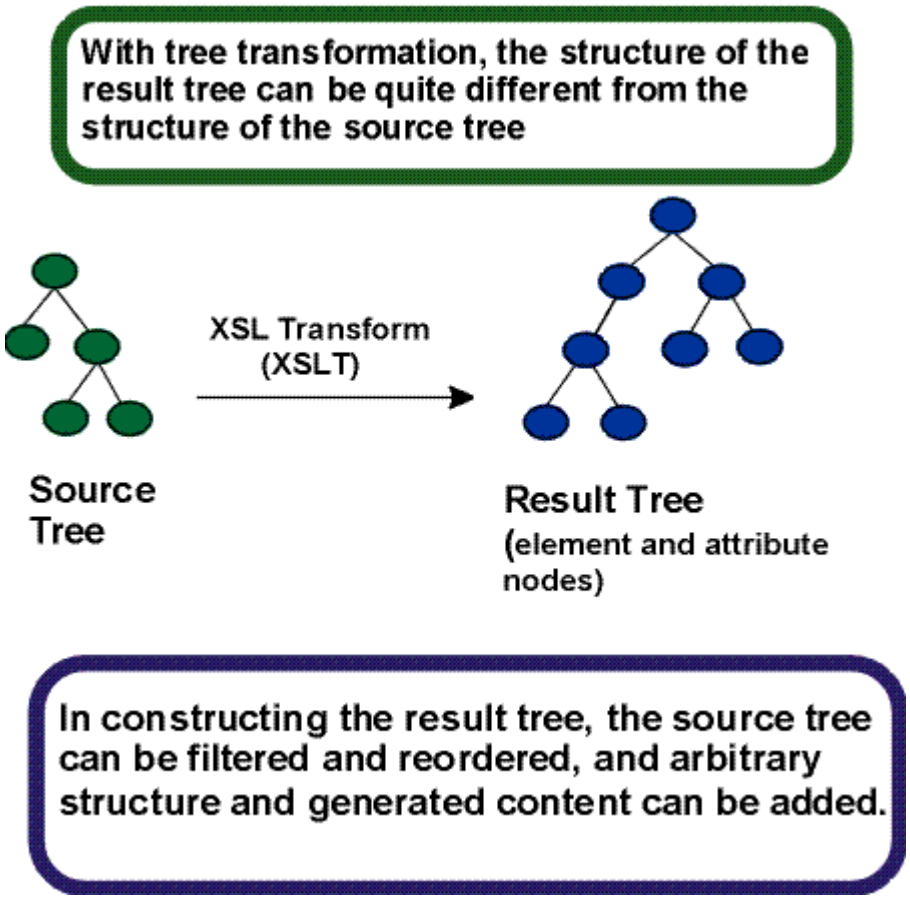


Figure 2.1.: Tree Transformation [W3C03c]

A stylesheet consists of templates and transformation rules. A template transforms a node set of the input XML by locating it through a XPath expression.

This XML file is used to illustrate a simple transformation:

```
<?xml version="1.0"?>
<agencies>
  <agency key="AMTRAK" />
  <agency key="DISA" />
…
  <agency key="NASA" />
  <agency key="VA" />
</agencies>
```

The XML file is transformed using this XSL stylesheet:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  version="1.0">
 <xsl:template match="/">
   <xsl:for-each select="agencies/agency">
     <xsl:value-of select="@key"/>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

The table above shows a simple stylesheet. It contains one template which operates on nodes beginning with the root node expressed by the XPath expression "/". Within the template two elements are defined. The first one ("for-each") executes its tag body for each "agency" child tag of an "agencies" parent tag contained in the input XML file. The "agencies" tag has to be located as a child of the root node. The root node is a virtual node and is not a visible tag within the XML file.

For every "agency" tag the element "value-of" is executed outputting the content of the attribute "key" ("@key") to the resulting XML file.

As the result of the transformation following file is created:

```
AMTRAK
DISA
…
NASA
VA
```

For more information on XSL Transformation see [W3C03b].

2.1.4 XSL Formatting Objects

The primary purpose of XSL Formatting Objects (XSL-FO) is to produce formatted results suitable for presentation. XSL-FO defines a set of formatting properties such as controlling page width, margin or letter spacing.

Formatting objects is accomplished by two steps. Firstly the input XML file needs to be transformed using XSLT to a XSL-FO formatting file. Secondly the formatted output is produced.

Following figure shows an overview of this process:

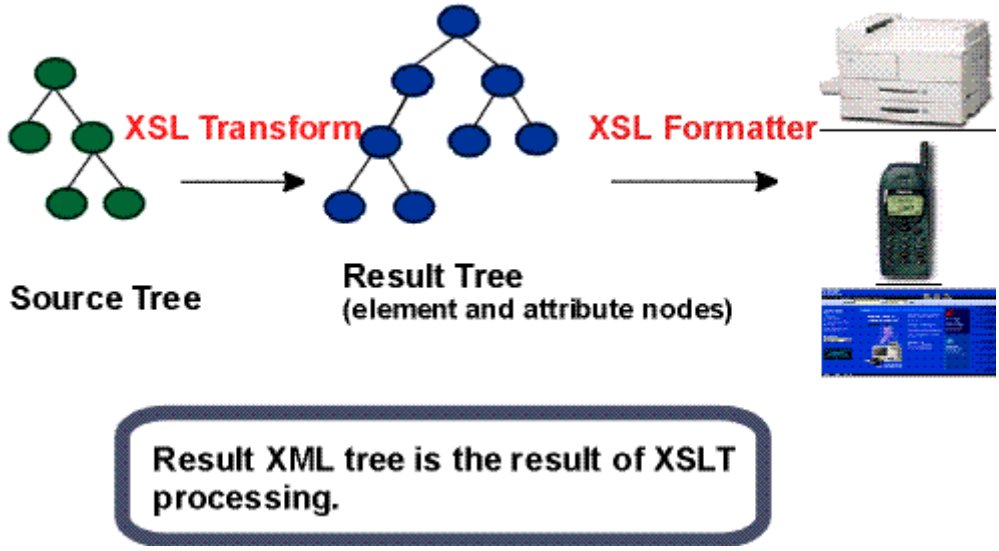


Figure 2.2.: Two Step Process [W3C03c]

Following this explanation, an example from the XSL-FO specification [W3C03c] is used. The first snippet shows the input XML file:

```
<doc>
  <p>This is the text of a paragraph that is going to be presented with the
      first line in small-caps.</p>
</doc>
```

This input file needs to be formatted using XSL-FO. Therefore the following XSLT stylesheet transforms it to a XSL-FO formatting file:

```
<?xml version='1.0'?>
<xsl:stylesheet   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  xmlns:fo="http://www.w3.org/1999/XSL/Format"
                  version='1.0'>
  <xsl:template match="p">
    <fo:block>
      <fo:initial-property-set font-variant="small-caps"/>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

Executing this transformation leads to the resulting XSL-FO formatting file:

```
<fo:block>
  <fo:initial-property-set font-variant="small-caps">
  </fo:initial-property-set>
  This is the text of a paragraph that is going to be presented with the
  first line in small-caps.
</fo:block>
```

This XSL-FO file is the input for a formatter. This formatter file could be used by Apache FOP [FOP07] to produce a PDF file. For further information on XSL-FO see the W3C homepage [W3C03c].

2.1.5 External Functions

The XSLT language offers a wide range of functionality. Nevertheless the required functions are sometimes missing. In these cases most XSLT implementations offer the possibility to extend the set of functions by supplying an API to call external functions.

Stylesheets containing external functions can be portable by following the proposals of the EXSLT.org.

> *"EXSLT is a community initiative to provide extensions to XSLT. […] We are trying to encourage the implementers of XSLT processors to use these extensions, so that your stylesheets can be more portable." [EXS07]*

2.2 Xalan-Java

2.2.1 Overview

> "*Xalan-Java fully implements XSL Transformations (XSLT) Version 1.0 and the XML Path Language (XPath) Version 1.0. XSLT is the first part of the XSL stylesheet language for XML. It includes the XSL Transformation vocabulary and XPath, a language for addressing parts of XML documents.*"[AXJ07a]

Xalan-Java represents the XSLT implementation by The Apache Foundation. At the time of writing it implements the version 1.0 of the XSLT specification.

Additionally it provides the possibility of calling external functions through so called Xalan-Java Extensions.

> "*Extensions written in Java are directly supported by Xalan-Java. For extensions written in languages other than Java, Xalan-Java uses the Bean Scripting Framework (BSF), an architecture for incorporating scripting into Java applications and applets.*"[AXJ07a]

The following code shows an example of an external functions definition:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="at.ac.wuwien.xsltnutshell.DatabaseQuery"
                extension-element-prefixes="props"
                version="1.0">
  <xalan:component prefix="props" elements="init" functions="getValue">
    <xalan:script lang="javaclass"
           src="at.ac.wuwien.xsltnutshell.DatabaseQuery"/>
  </xalan:component>
```

Xalan offers two possibilities to execute such a transformation. The difference between these types is the parsing process. Every stylesheet needs to be parsed before it can be applied to a XML file. If this transformation is required multiple times, the parsing of the document will be the bottleneck.

Therefore it is possible to store the parsed stylesheet in a template and reuse it for every transformation.

The following Java code (SimpleTransform example from Xalan [AXJ07a]) shows how the transformation will be initiated including the parsing:

```
/*
 * Copyright 1999-2004 The Apache Software Foundation.
…
 */
…
public class SimpleTransform
```

```
{
 public static void main(String[] args)
      throws TransformerException, TransformerConfigurationException,
      FileNotFoundException, IOException
 {
 // Use the static TransformerFactory.newInstance() method to instantiate
 // a TransformerFactory.
 TransformerFactory tFactory = TransformerFactory.newInstance();

 // Use the TransformerFactory to instantiate a Transformer that will work
 // with the stylesheet you specify. This method call also processes the
 // stylesheet into a compiled Templates object.
 Transformer transformer = tFactory.newTransformer(
           new StreamSource("birds.xsl"));

 // Use the Transformer to apply the associated Templates object to an XML
 // document (foo.xml) and write the output to a file (foo.out).
 transformer.transform(new StreamSource("birds.xml"),
             new StreamResult(new FileOutputStream("birds.out")));

 }
}
```

As mentioned in the source code, the transformer parses the stylesheet into a separate Templates object which is applied onto the input XML file. This Templates object is kept within the Transformer.

Reusing a Templates object, it is necessary to get a reference to it. The following Java code shows how this could be implemented:

```
 // Get an input stream for the XSL stylesheet
 StreamSource stylesheet = new StreamSource(xsl);

 // The TransformerFactory will compile the stylesheet and
 // put the result inside the Templates object
 TransformerFactory factory = TransformerFactory.newInstance();
 Templates templates = factory.newTemplates(stylesheet);

 // once the transformation process is needed, the template creates the
 // Transformer
 Transformer transformer = template.newTransformer();

 // the transformer reads the xml file and transforms it to the result
 // file
 Result result = new StreamResult(System.out);
 transformer.transform(xml, result);
```

This source code shows how a parsed Templates object can be obtained from the TransformerFactory. This object should be stored in the cache and reused for all subsequent transformations concerning this stylesheet. Templates objects are thread safe meaning that they can be used by multiple threads running concurrently. The Transformer object produced by the Templates object is not thread safe. Therefore the Transformer object must not be stored and reused.

Facilitating the execution of a simple transformation, Xalan-Java provides a Java class which takes a XML file, a XSL file and an output file name as parameters and performs the transformation. This class transforms the XML file using the first approach described above.

The following table shows the command to execute the spell checker nutshell using BSF4Rexx:

```
java -cp CLASSPATH org.apache.xalan.xslt.Process
      -IN nutshell6/words.xml
      -XSL nutshell6/spellcheckerOoRexx.xsl
      -OUT nutshell6/spellcheckerOoRexx.out
```

This command reads the “-XSL” file and parses it. Afterwards it transforms the “-IN” file and writes the result to the “-OUT” file.

2.2.2 Installation

The installation is as simple as downloading the Xalan-Java file from [AXJ07b] and extracting the content. For the nutshells all the jar-files from the root of the zip file are required.

2.3 JavaScript (Rhino)

2.3.1 Overview

Rhino is the open source implementation of the ECMA-262 ECMAScript Standard [ECM04] by Mozilla.org. Originally developed by Netscape it was released to Mozilla.org in April 1998 [MOZ06a].

JavaScript is best know for scripting HTML pages.

> "*Rhino is an implementation of the core language only and doesn't contain objects or methods for manipulating HTML documents.*" [MOZ06a]
>
> "*Additionally Rhino has implemented "JavaAdapters" which allows JavaScript to implement any Java interface or extend any Java class with a JavaScript object.*" [MOZ06a]

JavaScript can be used to access a SimpleDateFormat object provided by Java:

```
importPackage(Packages.at.ac.wuwien.xsltnutshell)
var outputFormat = "dd/MM/yyyy";

function parseDate(date, format) {
  parsedDate =
    new java.text.SimpleDateFormat(format).parse(date);
  return new java.text.SimpleDateFormat(outputFormat).
    format(parsedDate);
}
```

2.3.2 Installation

The complete Rhino implementation can be downloaded from the Mozilla homepage [MOZ06b] and unzipped. For the nutshells the js.jar file from the zip file is required.

2.4 The Bean Scripting Framework

2.4.1 BSF 2.4

> "*Bean Scripting Framework (BSF) is a set of Java classes which provides an easy to use scripting language support within Java applications. It also provides access to Java objects and methods from supported scripting languages.*" [BSF06a]

Within the nutshells BSF is used as a bridge between Xalan-Java and the code written in a supported scripting language. BSF currently supports several scripting languages and some of them are used to show the functionality of the nutshells.

2.4.2 BSF 3.0 – JSR-223

One part of the Java Standard Edition 6.0 [SUN07a] represents the ability to call code written in a scripting language. These scripting languages are called through an API specified in the JSR-223 specification [JCP07].

The team behind BSF has implemented this API in the upcoming 3.0 version of BSF.

At the time of writing BSF 3.0 is still in beta1 and the final and fully specification conforming implementation is scheduled for mid 2007. Therefore this paper will only show some nutshells with BSF 3.0 as the implementation is still subject to change.

2.4.3 Installation

The jar file can be downloaded from the BSF homepage [BSF06a] and unzipped to a folder. For the nutshells only the bsf.jar file from the lib directory within the zip file is required.

The installation of BSF 3.0 is slightly complex. BSF 3.0 can only be checked out from the Subversion repository [BSF06s] and has to be built using Maven [MVN07]. The resulting jar file is called bsf-3.0-beta1.jar. For more information on building the 3.0 version of BSF see the file "BUILDING" within the workspace checked out from Subversion.

Nevertheless building the binary distribution for BSF 3.0 was not successful. The first problem appeared to be with the back porting of jar files to JDK 1.4 by Retroweaver [RWV07]. After commenting out this part within the Maven build script some test cases did not succeed. Fortunately Prof. Flatscher provided the URL to a binary distribution generated by "ant elder" [BSF07b].

2.5 JudoScript

2.5.1 Overview

> "*JudoScript, or Judo for short, is a general-purpose, Java scripting and multi-domain language. A full-fledged general-purpose scripting language with full capability of Java scripting, JudoScript intimately supports most of today's key computing areas.*" [JDS07]

JudoScript emerged from an idea of a JDBC scripting language. Over the years many other scripting domains were introduced to Judo. Therefore it is internally called "domain-specific programming" [JDS07] as there are a plenty of functions and objects concerning different domains within the system of Judo.

It is possible to query a database table using the "db" domain:

```
function init (xslproc, elem) {
  db::connect to 'jdbc:postgresql://localhost:5432/xslt',
    'xslt', 'xslt';
  return null;
}

function getValue(key) {
  db::query qry:
  select value from properties where key = ?
  ; with @1:string = key;
  if (qry.next()) {
    return qry.value;
  }
  return '[[' + key + ']]';
}
```

2.5.2 Installation

The zip file can be downloaded from the JudoScript homepage [JDS07] and unzipped to a folder. For the nutshells only the judo.jar file from the root of the zip file is required.

2.6 OORexx

2.6.1 Overview

Open Object Rexx (OORexx) is an open source implementation of a freely available scripting language managed by RexxLA [OOR07]. It uses English-like statements and has fewer rules. Open Object Rexx has its origin in the scripting language of Rexx implemented by IBM between 1979 and 1982.

OORexx offers libraries for various domains like querying a database or accessing modules from OpenOffice.org. As a scripting language it is easily embeddable into a XSLT stylesheet. The linkage between XSLT and OORexx is possible through the Bean Scripting Framework [BSF06a] and BSF4Rexx [B4R07].

BSF4Rexx represents an implementation of a bridge between BSF and OORexx. It provides the functionality for calling scripts written in OORexx from within the BSF. At the time of writing BSF4Rexx only supports BSF version 2.4. Therefore all OORexx nutshells are tested with BSF 2.4. The BSF 3.0 support should be available by end of summer 2007.

> *"OpenOffice.org is a multiplatform and multilingual office suite and an open-source project. Compatible with all other major office suites, the product is free to download, use, and distribute."* [OOO07]

2.6.2 Installation

For the OORexx nutshells implemented in this bachelor thesis a few modules are required. The nutshells use OORexx to access the database and OpenOffice.org.

The binaries of OORexx can be downloaded from the OORexx homepage [OOR07] and installed by running the executable file. This installs the OORexx runtime and registers the runtime with the environment. Afterwards Rexx scripts can be executed by typing "rexx script.rex". The nutshells in this paper are implemented using the version 3.1.2.

Two nutshells use OpenOffice.org in the version 2.2 which can be downloaded from the homepage [OOO07] and installed by following the installation instructions.

The installation of BSF4Rexx is quite complex but the team of Prof. Flatscher [B4R07] maintains some installation scripts which hide the complexity from the developer. The correct order of the scripts is documented in two files included in the BSF4Rexx installation file. The text file "readmeBSF4Rexx.txt" explains the process of installing BSF4Rexx and the file "readmeOOo.txt" how the OpenOffice.org support is installed.

Those scripts also change the environment variables “PATH” and “CLASSPATH” through the script file “setEnvironment4OOo.cmd”. It is recommended to copy the new values of both environment variables to the corresponding system environment variables which facilitates the use of OORexx. After those installations the access to OpenOffice.org through BSF4Rexx can be tested using a provided test script calling “rexxj.cmd testOOo.rex” from the command line.

One nutshell uses BSF4Rexx and Rexx/SQL [RSQ07] to query a database. The Rexx/SQL installation can be downloaded from the Rexx/SQL homepage [RSQ07] and installed by unzipping the file. The Rexx/SQL directory has to be included into the PATH environment variable.

Accessing the database through Rexx/SQL, it should be registered as ODBC DataSource with Windows. Following figure shows the configuration window of the ODBC DataSource Manager (go to “Control Panel” ⇨ “Administrative Tools” ⇨ “Data Sources (ODBC)” ⇨ “Add”):

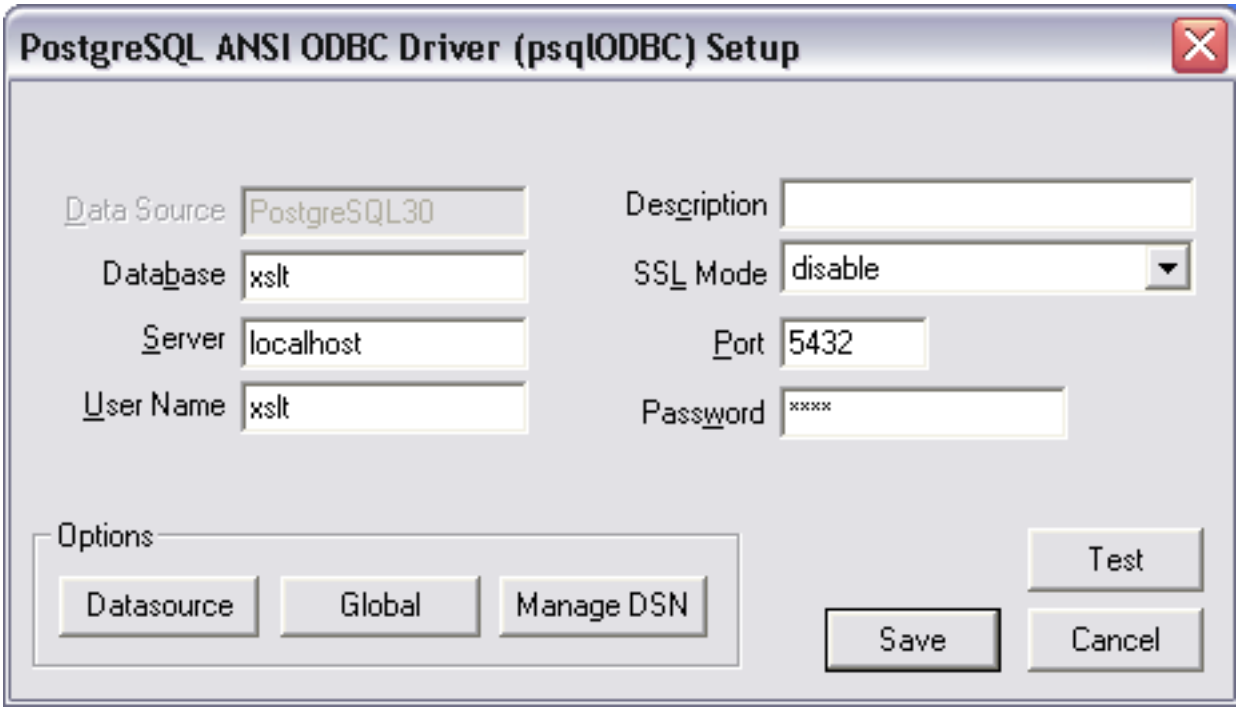


Figure 2.3.: ODBC Driver Set-up

A supplied test script can be executed through “rexxsql simple.cmd data=PostgreSQL30” which connects to the Data Source “PostgreSQL30” and writes the database name and version to the console.

2.7 JRuby

2.7.1 Overview

> "*JRuby is a 100% Java implementation of the Ruby programming language. It is Ruby for the JVM.*" [JRU07a]

The purpose of JRuby is providing the core classes of Ruby. Furthermore it supports the Rails framework. Through this support existing Ruby on Rails applications [ROR07] can be used for other applications.

The JRuby nutshell uses an existing object to read data from the database. Such objects are called ActiveRecords in JRuby:

```
class Properties < ActiveRecord::Base
end
```

2.7.2 Installation

The installation of JRuby is slightly complicated. First the actual version of JRuby has to be downloaded from the homepage [JRU07b] and unzipped to a directory (in this case to /Users/edin/jruby).

Afterwards two environment variables have to be set manually:

```
export JRUBY_HOME=/Users/edin/jruby
export PATH=/Users/edin/jruby/bin:$PATH
```

The variables need to be set for the next statement which installs Rails into the JRuby installation:

```
gem install rails -y
```

This takes a while to finish. Afterwards the JRuby installation is capable of running Ruby on Rails applications.

Using it within Xalan-Java and BSF, it is necessary to copy all jar files from $JRUBY_HOME/lib to your path.

3 Nutshells

This chapter contains the examples in various scripting languages. They are classified by functionality to facilitate a comparison between different languages.

3.1 Querying Databases

All examples in this chapter read additional information from the database.

The prerequisite to these nutshells is the existence of a database table containing the names of agencies. Following snippet shows the create statements for the user, database and table (PostgreSQL [POS07]):

```
1  create user xslt password 'xslt' ;
2  createdb xslt -O xslt;
3
4  CREATE TABLE properties (
5      key         varchar(40) PRIMARY KEY,
6      value       varchar(100)
7  );
```

Afterwards rows with abbreviations and names of different agencies are inserted into the table. Furthermore a XML file is used which only contains the abbreviations:

```
1  <?xml version="1.0"?>
2  <agencies>
3    <agency key="AMTRAK" />
4    <agency key="DARPA" />
5    <agency key="DCAA" />
6    <agency key="DEA" />
7    <agency key="FBI" />
8  …
9    <agency key="NSA" />
10   <agency key="USCIS" />
11   <agency key="USDA" />
12   <agency key="VA" />
13 </agencies>
```

Keeping the main focus on the XSL file and the binding of the scripting language, the nutshells all produce the following output:

```
1  <HTML>
2  <H1>JudoScript Example</H1>
3  <p>Here are the names of the Agencies:</p>
4  <p>[AMTRAK] National Railroad Passenger Corporation</p>
5  <p>[DARPA] Defense Advanced Research Projects Agency</p>
6  <p>[DCAA] Defense Contract Audit Agency</p>
7  <p>[DEA] Drug Enforcement Administration</p>
```

```
<p>[FBI] Federal Bureau of Investigation</p>
…
<p>[NSA] National Security Agency</p>
<p>[USCIS] U.S. Citizenship and Immigration Services</p>
<p>[USDA] Department of Agriculture</p>
<p>[VA] Department of Veterans Affairs</p>
</HTML>
```

The same files are used in the Using JRuby's ActiveRecord nutshell where Ruby on Rails (strictly speaking the ActiveRecord of Ruby on Rails) is used to query those names.

3.1.1 Nutshell Using EXSLT

> "*EXSLT is a community initiative to provide extensions to XSLT.*" [EXS07]
>
> "*Xalan-Java supports the EXSLT initiative to provide a set of standard extension functions and elements to XSLT users.*" [AXJ07a]

EXSLT is a built-in extension to Xalan-Java and offers many functions. This nutshell shows the use of the XConnection of EXSLT to connect to the database.

Executing queries against the database, EXSLT offers an API which can be embedded within a XSL stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                   version="1.0"
                   xmlns:sql="org.apache.xalan.lib.sql.XConnection"
                   extension-element-prefixes="sql">

<xsl:param name="driver" select="'org.postgresql.Driver'"/>
<xsl:param name="dburl"
  select="'jdbc:postgresql://localhost:5432/xslt'"/>
<xsl:param name="dbuser" select="'xslt'"/>
<xsl:param name="dbpwd" select="'xslt'"/>
<xsl:param name="query" select="'select value from properties
  where key = ?'"/>
   <xsl:template match="/">
     <HTML>

       <H1>EXSLT Example</H1>
       <p>Here are the names of the Agencies:</p>
       <xsl:for-each select="agencies/agency">
         <xsl:sort select="@key"/>
         <p>
         <xsl:text>[</xsl:text>
         <xsl:value-of select="@key"/>
         <xsl:text>] </xsl:text>
         <xsl:variable name="keyString"><xsl:value-of
             select="@key"/></xsl:variable>
         <xsl:variable name="db" select="sql:new($driver, $dburl,
                   $dbuser, $dbpwd)"/>
```

```
29          <xsl:value-of select="sql:addParameterWithType($db, $keyString,
30              'string')"/>
31          <xsl:variable name="resultset"  select="sql:pquery($db, $query,
32              'string')"/>
33          <xsl:value-of select="$resultset/sql/row-set/row"/>
34          <xsl:value-of select="sql:close($db)"/>
35          </p>
36        </xsl:for-each>
37      </HTML>
38    </xsl:template>
39
40  </xsl:stylesheet>
```

The line 4 defines the SQL namespace for EXSLT and line 5 marks the namespace as an extension element prefix. The lines 7 to 13 define some parameters for the query which could be overridden by calling a stylesheet.

The interesting part starts at line 25 where the key attribute from the current node is saved into a local variable. On line 27 a new select statement is created providing the parameters “driver”, “dburl” “dbuser” and “dbpwd”. The created select statement is populated with a parameter on line 29 using the saved “keyString”.

It is also possible to select the value from the attribute instead of saving it in between. This would pass a reference to the attribute node to the select statement instead of the contained value. Therefore this workaround is necessary.

On line 31 the SQL statement is executed and the result is stored in the variable result set. The statement on line 33 selects the value of the resulting row and writes it to the output file.
Finally the SQL statement is closed on line 34.

3.1.2 Nutshell Using Java

This nutshell shows the possibility of calling miscellaneous Java function from within a transformation. As in all nutshells in this chapter the properties table will be queried.

The XSL file below shows the use of Java for processing the XML file:

```
1   <?xml version="1.0"?>
2   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                   xmlns:xalan="http://xml.apache.org/xalan"
4                   xmlns:props="at.ac.wuwien.xsltnutshell.DatabaseQuery"
5                   extension-element-prefixes="props"
6                   version="1.0">
7
8     <xalan:component prefix="props"
9                      elements="init" functions="getValue">
10      <xalan:script lang="javaclass"
11          src="at.ac.wuwien.xsltnutshell.DatabaseQuery"/>
```

```
12    </xalan:component>
13
14    <xsl:template match="/">
15      <HTML>
16        <H1>Java Example</H1>
17        <props:init/>
18        <p>Here are the names of the Agencies:</p>
19        <xsl:for-each select="agencies/agency">
20          <xsl:sort select="@key"/>
21          <p>
22          <xsl:text>[</xsl:text>
23          <xsl:value-of select="@key"/>
24          <xsl:text>] </xsl:text>
25          <xsl:value-of select="props:getValue(@key)"/>
26          </p>
27        </xsl:for-each>
28      </HTML>
29    </xsl:template>
30  </xsl:stylesheet>
```

Calling an external function, a Xalan component has to be defined. Therefore the Xalan namespace is set at line 3 and on line 4 “props” is chosen as the namespace for the external function. The lines 8 to 12 specify the component to be of a Java class (line 10) with the class name as source file (line 11).

The main point is the definition of the methods which should be exposed. “Init” is identified as a method which should be exposed as an element and “getValue” as a function (line 9). On line 8 the connection is set to the namespace which is defined earlier.

Within the Java code some initialisation is needed. So the “init” method is called at line 17.

The last item left is the query of the value which is executed through the “getValue” method (line 25). In contrast to the nutshell using EXSLT, it is not necessary to save the value of the attribute to get its value passed to the method. If the external function is implemented in Java the value of the attribute is passed on to the external function instead of the reference to the attribute node.

The implementation in Java uses JDBC to query the database:

```
1   package at.ac.wuwien.xsltnutshell;
2
3   import java.sql.Connection;
4   import java.sql.DriverManager;
5   import java.sql.PreparedStatement;
6   import java.sql.ResultSet;
7   import java.sql.SQLException;
8
9   public class DatabaseQuery {
10    Connection c = null;
11    public void init(org.apache.xalan.extensions.XSLProcessorContext
12      context, org.w3c.dom.Element elem) {
13      try {
```

```
14        Class.forName("org.postgresql.Driver");
15        c = DriverManager.getConnection(
16          "jdbc:postgresql://localhost:5432/xslt",
17          "xslt", "xslt");
18      } catch (Exception e) {
19        e.printStackTrace();
20      }
21    }
22
23    public String getValue(String key) {
24      PreparedStatement stmt = null;
25      ResultSet rs = null;
26      try {
27        stmt = c.prepareStatement(
28          "select value from properties where key = ? ");
29        stmt.setString(1, key);
30        rs = stmt.executeQuery();
31        if (rs.next()) {
32          return rs.getString(1);
33        }
34      } catch (Exception e) {
35        e.printStackTrace();
36      } finally {
37        try {
38          if (rs != null)
39            rs.close();
40          if (stmt != null)
41            stmt.close();
42        } catch (SQLException e) {
43          e.printStackTrace();
44        }
45      }
46      return "<<" + key + ">>";
47    }
48  }
```

The “init” method (lines 11 to 21) gets a connection to the database and saves it in the property “c”. Reading the value from the database, the method “getValue” (lines 23 to 47) is called passing the key of the required value.

3.1.3 Nutshell Using JudoScript

This nutshell uses JudoScript to query a database table from within a transformation. Achieving this functionality, the “db” domain of JudoScript is used.

The database accessing code is embedded within the XSL file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="DatabaseQuery"
                extension-element-prefixes="props"
                version="1.0">

 <xalan:component prefix="props"
                  elements="init" functions="getValue">
   <xalan:script lang="judoscript">

     function init (xslproc, elem) {
       db::connect to 'jdbc:postgresql://localhost:5432/xslt',
         'xslt', 'xslt';
       return null;
     }

     function getValue(key) {
       db::query qry:
       select value from properties where key = ?
       ; with @1:string = key;
       if (qry.next()) {
         return qry.value;
       }
       return '[[' + key + ']]';
     }
   </xalan:script>
 </xalan:component>

 <xsl:template match="/">
   <HTML>
     <H1>JudoScript Example</H1>
     <props:init/>
     <p>Here are the names of the Agencies:</p>
     <xsl:for-each select="agencies/agency">
       <xsl:sort select="@key"/>
       <p>
       <xsl:text>[</xsl:text>
       <xsl:value-of select="@key"/>
       <xsl:text>] </xsl:text>
       <xsl:variable name="keyString">
         <xsl:value-of select="@key"/>
       </xsl:variable>
       <xsl:value-of select="props:getValue($keyString)"/>
       </p>
     </xsl:for-each>
   </HTML>
 </xsl:template>

</xsl:stylesheet>
```

Calling an external function, a Xalan component is defined. Therefore the Xalan namespace is specified at line 3 and on line 4 “props” is chosen as the namespace for the external function.

The lines 8 to 28 specify the component. On lines 8 and 9 the component is defined in the same way as for Java.

Instead of supplying the name of the source file the source code is inserted directly into the XSL file (lines 12 to 26).

The directive on line 9 states that “init” is a method exposed as an element and “getValue” is exposed as a function. On line 8 the connection to the namespace is set which is defined earlier.

Within the JudoScript a connection to the database is established (method “init” on lines 12 to 16) and the properties table for the supplied key is queried (method “getValue” on lines 18 to 26).

The “init” method is called at line 33 and the query of the value gets executed through the “getValue” method (line 44).

The “key” attribute value needs to be saved to the variable “keyString” to get the value passed on to the method instead of the reference to the attribute node itself (lines 41 to 43).

3.1.4 Nutshell Using BSF4Rexx

This nutshell shows the possibility of executing queries against the properties table by calling an OORexx script from within a transformation.

The OORexx script is implemented as an embedded Xalan component within the XSL file:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="DatabaseQuery"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8   <xalan:component prefix="props"
9                    elements="init cleanup" functions="getValue">
10    <xalan:script lang="rexx">
11      <![CDATA[
12        parse arg key
13        if key =="" then return ""
14        Call RxFuncAdd 'SQLLoadFuncs', 'rexxsql', 'SQLLoadFuncs'
15        Call SQLLoadFuncs
16        call init
17        agencyvalue = getValue(key)
18        call cleanup
19        return agencyvalue
20
```

```
21          init:
22            rc = SQLConnect( 'c1', '', '', 'PostgreSQL30')
23          return
24
25          cleanup:
26            rc = SQLDisconnect( 'c1' )
27          return
28
29          getValue:
30          parse arg key
31            query = 'select value from properties where key = ? '
32            rc = SQLCommand(stmt, query, "CHAR", key)
33            resultvalue = stmt.value.1
34          return resultvalue
35        ]]>
36      </xalan:script>
37    </xalan:component>
38
39    <xsl:template match="/">
40      <HTML>
41        <H1>ooRexx Example</H1>
42        <p>Here are the names of the Agencies:</p>
43        <xsl:for-each select="agencies/agency">
44          <xsl:sort select="@key"/>
45          <p>
46          <xsl:text>[</xsl:text>
47          <xsl:value-of select="@key"/>
48          <xsl:text>] </xsl:text>
49          <xsl:variable name="keyString">
50            <xsl:value-of select="@key"/>
51          </xsl:variable>
52          <xsl:value-of select="props:getValue($keyString)"/>
53          </p>
54        </xsl:for-each>
55      </HTML>
56    </xsl:template>
57
58  </xsl:stylesheet>
```

Executing queries against a table includes connecting to the database, executing the queries and disconnecting from the database. The flow is implemented in the OORexx script above where the “init” method connects to the database (lines 21 to 23).

The method “getValue” (lines 29 to 34) queries the database and returns the result from the select (line 34). Finally the script disconnects from the database using the method “cleanup” (lines 25 to 27).

The script above works as intended if executed as an OORexx command from the command line. As soon as it gets embedded to the XSLT stylesheet it behaves differently. Independent from the method called in the XSLT stylesheet (like on line 52) the complete script gets executed.

Therefore the lines 12 to 19 are included. The parameter supplied by Xalan is stored (line 12) and used to call the method “getValue” (line 17) after the connection to the database was established (line 16). On line 18 the script disconnects from the database. Omitting this line leads to a “too many open connections” exception, as all established connections stay open until the execution of the script finishes.

At the time Xalan parses the script, it is executed without a parameter which leads to a NullPointerException. Avoiding this, the code on line 13 is inserted to the script.

The XSLT stylesheet looks similar to the other database nutshells. First a Xalan component has to be defined. Therefore the Xalan namespace is set on line 3 and on line 4 “props” is chosen as the namespace for the external function. The lines 8 and 9 specify the component to be an OORexx script (line 10) with three methods (“init”, “getValue” and “cleanup”) and the source of the script embedded below.

This nutshell uses the library Rexx/SQL [RSQ07] which is added and initialised on lines 14 and 15. The library offers an API to connect to the database using the command SQLConnect (line 22), to query the database using SQLCommand (line 32) and to disconnect from it by using SQLDisconnect (line 26).

The result of the query can be retrieved from the statement after the execution (line 33). The term “value” indicates the name of the selected column and the last part of the statement “stmt.value.1” indicates the number of the row. The number of rows can be retrieved by executing the command “stmt.value.0”.

The lines 49 to 51 store the value of the attribute “key” in the variable “keyString” which is used to call the script on line 52. Without saving the value, Xalan would provide the reference to the attribute node instead of the value.

3.1.5 Nutshell Using JavaScript

This nutshell shows the JavaScript version of a transformation which queries the database for properties.

The following code shows the XSL file for processing the XML file containing the database access code using JavaScript:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="DatabaseQuery"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8
9   <xalan:component prefix="props"
```

```
10                      elements="init" functions="getValue">
11      <xalan:script lang="javascript">
12        <![CDATA[
13          var conn = null;
14
15          function init(xslProcessorContext, elem) {
16            java.lang.Class.forName("org.postgresql.Driver");
17            conn = java.sql.DriverManager.getConnection(
18              "jdbc:postgresql://localhost:5432/xslt", "xslt", "xslt");
19          }
20
21          function getValue(key) {
22            var stmt = conn.prepareStatement("select value from
23              properties where key = ? ");
24            stmt.setString(1, key);
25            var rs = stmt.executeQuery();
26            if (rs.next())
27              return rs.getString(1);
28            return "<<" + key + ">>";
29          }
30        ]]>
31      </xalan:script>
32    </xalan:component>
33
34    <xsl:template match="/">
35      <HTML>
36        <H1>Java Example</H1>
37        <props:init/>
38        <p>Here are the names of the Agencies:</p>
39        <xsl:for-each select="agencies/agency">
40          <xsl:sort select="@key"/>
41          <p>
42          <xsl:text>[</xsl:text>
43          <xsl:value-of select="@key"/>
44          <xsl:text>] </xsl:text>
45          <xsl:variable name="keyString">
46            <xsl:value-of select="@key"/>
47          </xsl:variable>
48          <xsl:value-of select="props:getValue($keyString)"/>
49          </p>
50        </xsl:for-each>
51      </HTML>
52    </xsl:template>
53
54  </xsl:stylesheet>
```

The XSL file above contains the JavaScript code embedded as Xalan component. The Xalan namespace is defined on line 3 and line 4 “props” is chosen as the namespace for the external function.

The lines 9 to 32 specify the component. On lines 9 and 10 two methods are defined. “Init” is identified as a method which should be exposed as an element and “getValue” is defined as a function (line 10). On the previous line the connection to the namespace is set which is defined earlier.

In contrast to Java the complete source code is embedded into the XSL file (lines 12 to 30).

The method “init” loads the database driver (line 16) and opens a new connection to the database (lines 17 and 18). The second exposed method “getValue” (lines 21 to 29) receives a parameter containing the key for the select. Afterwards it prepares (line 22) and executes the select statement (line 25) with the parameter as bind variable (line 24) and finally returns the result of the select statement.

The “init” method is called at line 37 through the use as XML tag and the value gets selected through the “getValue” method on line 48.

As the “value-of” tag returns the reference to the attribute the value has to be extracted previously. Therefore the value of the attribute is stored into a variable (lines 45 to 47) and afterwards passed on to the method instead of the reference to the attribute node itself (line 48).

3.2 Implementing Missing Functions

The XSLT specification provides a rich set of functions to the developer. Nevertheless sometimes they do not suffice and additional functions need to be developed. This chapter deals with such functions.

3.2.1 Formatting Date Using Java

This nutshell formats a date using a call out to a Java function from within the transformation. Showing a different possibility, the abbreviated syntax for extensions implemented in Java is used.

The input XML file contains a list of countries and a date per country. Additionally the date format is provided as attribute:

```
<?xml version="1.0"?>
<countries>
  <country date="20070405" format="yyyyMMdd">
    United States of America
  </country>
  <country date="20070406" format="yyyyMMdd">
    United Kingdom
  </country>
  <country date="20070407" format="yyyyMMdd">
    Brazil
  </country>
  <country date="20070408" format="yyyyMMdd">
    Cuba
  </country>
  <country date="20070409" format="yyyyMMdd">
    Chile
  </country>
  <country date="20070410" format="yyyyMMdd">
    Mexico
  </country>
  <country date="20070411" format="yyyyMMdd">
    Austria
  </country>
  <country date="20070412" format="yyyyMMdd">
    Sweden
  </country>
</countries>
```

As mentioned above the abbreviated syntax for calling Java is used in the XSL file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:java="http://xml.apache.org/xalan/java"
  exclude-result-prefixes="java" version="1.0">

  <xsl:template match="/">
    <timedCountries>
```

```
 8         <xsl:apply-templates select="/countries/country" />
 9       </timedCountries>
10    </xsl:template>
11
12    <xsl:template match="country">
13      <country>
14        <value>
15          <xsl:value-of select="." />
16        </value>
17        <xsl:variable name="date" select="string(@date)" />
18        <xsl:variable name="format" select="string(@format)" />
19        <xsl:variable name="formatter"
20          select="java:java.text.SimpleDateFormat.new($format)" />
21        <date>
22        <xsl:value-of select="java:parse($formatter, $date)" />
23        </date>
24
25      </country>
26    </xsl:template>
27
28  </xsl:stylesheet>
```

Calling an external function, a Java namespace is defined (line 3). Using this syntax would generate a root element containing the java namespace within the “timedCountries” tag. To prevent this, the namespace is excluded from the result (line 4).

On lines 17 and 18 the values of the supplied date and format are stored in variables. To parse the date from the XML file, a SimpleDateFormat is needed which is created on line 20 and stored in the variable called “formatter” (line 19).

The call to the method “parse” of the SimpleDateFormat takes the formatter and the date to format as parameters (line 22). The result of the “parse” method is written to the XML file.

The XML file below shows the result of the transformation:

```
 1  <?xml version="1.0" encoding="UTF-8"?>
 2  <timedCountries>
 3    <country>
 4      <value>United States of America</value>
 5      <date>Thu Apr 05 00:00:00 CEST 2007</date>
 6    </country>
 7    <country>
 8      <value>United Kingdom</value>
 9      <date>Fri Apr 06 00:00:00 CEST 2007</date>
10    </country>
11    …
12    <country>
13      <value>Austria</value>
14      <date>Wed Apr 11 00:00:00 CEST 2007</date>
15    </country>
```

```
16    <country>
17      <value>Sweden</value>
18      <date>Thu Apr 12 00:00:00 CEST 2007</date>
19    </country>
20  </timedCountries>
```

This nutshell aims to show how simple it is to call Java code and extend the functionality of XSLT.

3.2.2 Printing Enhanced Node Information Using EXSLT

Sometimes it is necessary to process information regarding the data within the XML file, e.g. the line number or the “System ID” of the XML file.

This nutshell shows the possible extension functions which Xalan-Java offers.

For this nutshell the following XML file is used as input:

```
1   <?xml version="1.0"?>
2   <countries>
3     <america filename="redirectOutputAmerica.out">
4       <country>United States of America</country>
5       <country>Mexico</country>
6       <country>Canada</country>
7       <country>Brazil</country>
8       <country>Cuba</country>
9     </america>
10    <europe filename="redirectOutputEurope.out">
11      <country>United Kingdom</country>
12      <country>Germany</country>
13      <country>Spain</country>
14      <country>Austria</country>
15      <country>Greece</country>
16    </europe>
17    <world>
18      <country>Madagascar</country>
19      <country>Egypt</country>
20      <country>China</country>
21    </world>
22  </countries>
```

Compared to the other nutshells, this XSL file is quite long but covers all supplied functions at once:

```
1   <?xml version="1.0"?>
2   <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3           xmlns:xalan="http://xml.apache.org/xalan"
4           xmlns:exslt="http://exslt.org/common"
5           xmlns:ni="xalan://org.apache.xalan.lib.NodeInfo"
6           exclude-result-prefixes="xalan exslt ni"
7           version="1.0">
8
```

```
 9    <xsl:template match="/">
10      <nodeset-countries>
11        <systemid>
12          <xsl:value-of select="ni:systemId(.)" />
13        </systemid>
14
15        <publicid>
16          <xsl:value-of select="ni:publicId(.)" />
17        </publicid>
18
19        <america-filename>
20          <xsl:variable name="xpathvalue">
21            /countries/america/@filename
22          </xsl:variable>
23          <xsl:value-of select="xalan:evaluate($xpathvalue)" />
24        </america-filename>
25
26        <europe-filename>
27          <xsl:variable name="xpathvalue">
28            /countries/europe/@filename
29          </xsl:variable>
30          <xsl:value-of select="xalan:evaluate($xpathvalue)" />
31        </europe-filename>
32
33        <xsl:for-each select="exslt:node-set(/countries)/*//*">
34          <country>
35            <tagname>
36              <xsl:value-of select="name(.)" />
37            </tagname>
38
39            <linenumber>
40              <xsl:value-of select="ni:lineNumber(.)" />
41            </linenumber>
42
43            <columnNumber>
44              <xsl:value-of select="ni:columnNumber(.)" />
45            </columnNumber>
46
47            <value>
48              <xsl:value-of select="." />
49            </value>
50
51            <valueparts>
52              <xsl:variable name="value">
53                <xsl:value-of select="." />
54              </xsl:variable>
55              <xsl:for-each select="xalan:tokenize($value)">
56                <part>
57                  <xsl:value-of select="." />
58                </part>
59              </xsl:for-each>
60            </valueparts>
61          </country>
```

```
62          </xsl:for-each>
63        </nodeset-countries>
64      </xsl:template>
65    </xsl:stylesheet>
```

The external functions in this nutshell cover three extensions of Xalan-Java with different namespaces (lines 3 to 5). The extension of NodeInfo needs an additional parameter on starting the transformation. This parameter is "-L" without which the result is always "-1" as the line number is not supplied.

In this nutshell the Xalan functions "evaluate" (lines 23 and 30) and "tokenize" (line 55) are used. The first one takes a XPath expression and evaluates the result. On line 20 the XPath expression is stored to a variable which later is used for the call to evaluate. This expression is hard coded but can be supplied easily by the XML file or a parameter to the XSL file.

The method "tokenize" takes a string, splits it into discrete nodes and returns the resulting node-set. A second parameter can be supplied indicating the delimiters. If they are absent all white space characters are used as delimiters.

The resulting XML part is shown below on lines 7 and 8 (result of the call to evaluate) and on lines 14 to 19 (result of the call to "tokenize" with more than one word contained in the string).

The node-set function specified by EXSLT (line 33) casts a tree to a node set. The same functionality can be implemented through the use of XPath.

The NodeInfo API provides the information regarding the node from the XML file which is currently processed. The method "lineNumber" (line 40) gets the line number of the node within the original XML file. Likewise the method "columnNumber" (line 44) returns the number of the column containing the node within the XML file. These methods can be used in case of debugging.

Furthermore the NodeInfo API offers the methods "systemId" (line 12) and "publicId" (line 16) to get the mentioned IDs from the XML file.

The XML file below shows the output from the transformation:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <nodeset-countries>
3     <systemid>
4       file:///workspace/XSLTExtfunc/nutshell2/redirectOutput.xml
5     </systemid>
6     <publicid />
7     <america-filename>redirectOutputAmerica.out</america-filename>
8     <europe-filename>redirectOutputEurope.out</europe-filename>
9     <country>
10      <tagname>country</tagname>
11      <linenumber>4</linenumber>
12      <columnNumber>14</columnNumber>
13      <value>United States of America</value>
```

```
14      <valueparts>
15        <part>United</part>
16        <part>States</part>
17        <part>of</part>
18        <part>America</part>
19      </valueparts>
20    </country>
21    …
22    <country>
23      <tagname>country</tagname>
24      <linenumber>20</linenumber>
25      <columnNumber>14</columnNumber>
26      <value>China</value>
27      <valueparts>
28        <part>China</part>
29      </valueparts>
30    </country>
31  </nodeset-countries>
```

3.2.3 Generating More Than One Output File Using EXSLT

A XSL transformation usually generates one (mostly XML) file. This limitation can be removed by the use of EXSLT to create more than one resulting file. This nutshell shows how.
The XML file below shows the input for this nutshell:

```
1   <?xml version="1.0"?>
2   <countries>
3     <america filename="redirectOutputAmerica.out">
4       <country>United States of America</country>
5       <country>Mexico</country>
6       <country>Canada</country>
7       <country>Brazil</country>
8       <country>Cuba</country>
9     </america>
10    <europe filename="redirectOutputEurope.out">
11      <country>United Kingdom</country>
12      <country>Germany</country>
13      <country>Spain</country>
14      <country>Austria</country>
15      <country>Greece</country>
16    </europe>
17    <world>
18      <country>Madagascar</country>
19      <country>Egypt</country>
20      <country>China</country>
21    </world>
22  </countries>
```

The data in the XML file contains three different tags, “america”, “europe” and “world”. This nutshell saves the content of each tag into a separate file.

The XSL file below is used for processing the XML file:

```
1  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2        version="1.0"
3        xmlns:redirect="http://xml.apache.org/xalan/redirect"
4        extension-element-prefixes="redirect">
5
6    <xsl:template match="/">
7      <world-countries>
8        <xsl:apply-templates />
9      </world-countries>
10   </xsl:template>
11
12   <xsl:template match="/countries/america">
13     <redirect:open select="@filename" />
14     <redirect:write select="@filename">
15       <american-countries>
16         <xsl:apply-templates />
17       </american-countries>
18     </redirect:write>
19     <redirect:close select="@filename" />
20   </xsl:template>
21
22   <xsl:template match="/countries/europe">
23     <redirect:open select="@filename" />
24     <redirect:write select="@filename">
25       <european-countries>
26         <xsl:apply-templates />
27       </european-countries>
28     </redirect:write>
29     <redirect:close select="@filename" />
30   </xsl:template>
31
32
33   <xsl:template match="country">
34     <country>
35       <xsl:apply-templates />
36     </country>
37   </xsl:template>
38
39 </xsl:stylesheet>
```

Xalan-Java offers the possibility to generate more than one file by using the external functionality of the “redirect” API. This API offers functions similar to a file library known from most programming languages.

At line 3 the namespace for the redirect functions is set. To get a writeable file it has to be opened with the “open” command supplying the file name (lines 13 and 23). On lines 14 to 18 and 24 to 28 the file is written with the call to the “write” function. The body of the write tag is evaluated and the result is saved to the file specified at the write tag. At the end of the generation the file needs to be closed (lines 19 and 29).

As a result of the transformation three files are created. The following XML file shows the standard output from the transformation (everything which is not redirected to another file):

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <world-countries>
3    <country>Madagascar</country>
4    <country>Egypt</country>
5    <country>China</country>
6  </world-countries>
```

All American countries are written to the file “redirectOutputAmerica.out”:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <american-countries>
3   <country>United States of America</country>
4   <country>Mexico</country>
5   <country>Canada</country>
6   <country>Brazil</country>
7   <country>Cuba</country>
8  </american-countries>
```

Finally all European countries are written to the file “redirectOutputEurope.out”:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <european-countries>
3    <country>United Kingdom</country>
4    <country>Germany</country>
5    <country>Spain</country>
6    <country>Austria</country>
7    <country>Greece</country>
8  </european-countries>
```

3.2.4 Spell Checking Using BSF4Rexx

This nutshell shows how a list of words can be spell checked using the spell checker of OpenOffice.org. Achieving this functionality, the nutshell uses BSF4Rexx and the API of OpenOffice.org to access the spell checker.

The code in this nutshell is based on an example of Ahammer Andreas [AHA05] which is included in the zip file of BSF4Rexx [B4R07]. It was changed to be used as a function returning one of the strings “NOT correct” or “correct” as a result of the spell checking.

The following snippet shows the XML file containing the list of words:

```
1  <?xml version="1.0"?>
2  <words>
3    <word value="gone" />
4    <word value="accumulator" />
5    <word value="advocate" />
6  …
7    <word value="truck" />
8    <word value="vest" />
9    <word value="wicked" />
10   <word value="yankee" />
11 </words>
```

The content of each value attribute (line 3) is spell checked by this nutshell.

The following code shows the XSL file for processing the XML file using OORexx as scripting language:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="WriterSpellchecker"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8    <xalan:component prefix="props" functions="isCorrectlySpelled">
9      <xalan:script lang="rexx">
10     <![CDATA[
11       parse arg word
12       if word =="" then return ""
13       call init
14       iscorrect = spellcheck(word)
15       return iscorrect
16
17       init:
18         -- Example 29
19         -- From Ahammer, Andreas: <http://wi.wu-wien.ac.at/
20           -- rgf/diplomarbeiten/index.html#bakk_07>
21         -- use the SpellChecker to check a word
22            /* initialise connection to server, get XContext */
23         xContext = UNO.connect()  -- connect to server and retrieve the
24                                   -- XContext object
25         -- retrieve XMultiComponentFactory
26         XMcf = xContext~getServiceManager
27
28            /* create the LinguServiceManager and the SpellChecker */
29         mxLinguSvcMgrName = xMcf~createInstanceWithContext
30           ("com.sun.star.linguistic2.LinguServiceManager", xContext)
31         xSpellChecker = mxLinguSvcMgrName~XLinguServiceManager~
32                         getSpellChecker
```

```
33
34              /*  load the required class "com.sun.star.lang.Locale"
35                  and set the language to US-english */
36            CALL UNO.loadClass "com.sun.star.lang.Locale"
37            aLocale = .UNO~Locale~new("en", "US", "")
38          return
39
40          spellcheck:
41          parse arg aWord
42
43            /* test the word if it is valid */
44            isCorrect = xSpellChecker~isValid(aWord, aLocale, .UNO~noProps)
45
46            wordCorrect = ""
47            IF isCorrect = 0 THEN wordCorrect = "NOT"
48          return wordCorrect "correct"
49
50          ::requires UNO.CLS    -- get UNO support
51       ]]>
52        </xalan:script>
53    </xalan:component>
54
55    <xsl:template match="/">
56      <HTML>
57        <H1>ooRexx Example</H1>
58        <p>Here are the words and the result, if they are
59             correctly spelled:</p>
60        <xsl:for-each select="words/word">
61          <xsl:sort select="@value"/>
62          <p>
63          <xsl:text>[</xsl:text>
64          <xsl:value-of select="@value"/>
65          <xsl:text>] </xsl:text>
66          <xsl:variable name="valueString">
67            <xsl:value-of select="@value"/>
68          </xsl:variable>
69          <xsl:value-of select="props:isCorrectlySpelled($valueString)"/>
70          </p>
71        </xsl:for-each>
72      </HTML>
73    </xsl:template>
74
75  </xsl:stylesheet>
```

OpenOffice.org supplies many functions which may be used in OORexx through the UNO.CLS library (line 23).

Executing many function calls to a slowly initialising library implicates that the initialising process should only be executed once for the complete handling. Therefore this script implements an “init” method to get a reference to the spell checker of OpenOffice.org (lines 17 to 38) and an extra method called “spellcheck” which supplies each word separately to the spell checker (lines 40 to 48).

The script above works as intended if executed as an OORexx command from the command line. As soon as it is embedded into a XSLT stylesheet it behaves differently. Independent from the method called in the XSLT stylesheet (like on line 69) the complete script is executed. Therefore the lines 11 to 15 are included. The parameter "word" supplied by Xalan is stored (line 11) and used to call the method "spellcheck" (line 14) after the initialisation is completed (line 13).

At the time Xalan parses the script, it is executed without a parameter which leads to a NullPointerException. Avoiding this, the code on line 12 is inserted to the script.

A Xalan component is defined using the Xalan namespace on line 3 and on line 4 "props" is choosen as the namespace for our external function. The lines 8 and 9 specify the component to be an OORexx script with one method "isCorrectlySpelled". This shows that the name of the method (line 40) is independent from the called method (line 69).

This nutshell uses the library UNO.CLS [B4R07] which gets installed with BSF4Rexx (line 50). The library offers an API to connect to the OpenOffice.org application using a XContext object (line 23). The XContent object provides an API to the SpellChecker of OpenOffice.org (lines 26 to 32).

On line 44 the SpellChecker is used to validate the supplied word. If the word is not spelled correctly, the result of the validation is "0".

The lines 66 to 68 store the content of the attribute "value" in the variable "valueString" which is used to call the script on line 69. Without saving the value Xalan would provide the reference to the attribute node instead of the value.

The resulting content of the transformation is shown in the following table:

```
 1  <HTML>
 2    <H1>ooRexx Example</H1>
 3    <p>
 4      Here are the words and the result, if they are correctly spelled:
 5    </p>
 6    <p>[accumulator] correct</p>
 7    <p>[advocate] correct</p>
 8    <p>[cop] correct</p>
 9    <p>[crib] correct</p>
10  …
11    <p>[switchback] correct</p>
12    <p>[truck] correct</p>
13    <p>[vest] correct</p>
14    <p>[wicked] correct</p>
15    <p>[yankee] NOT correct</p>
16  </HTML>
```

After the resulting file is written, the transformation process does not terminate. Instead it has to be killed.

3.2.5 Formatting Date Using JavaScript

This nutshell shows the possibility of formatting dates within a transformation using JavaScript.

For this nutshell the following XML file is used as input. It contains country entries with a date and the format string of the stated date:

```
<?xml version="1.0"?>
<countries>
  <country date="20070405" format="yyyyMMdd">
    United States of America
  </country>
  <country date="20070406" format="yyyyMMdd">
    United Kingdom
  </country>
  <country date="20070407" format="yyyyMMdd">
    Brazil
  </country>
  <country date="20070408" format="yyyyMMdd">
    Cuba
  </country>
  <country date="20070409" format="yyyyMMdd">
    Chile
  </country>
  <country date="20070410" format="yyyyMMdd">
    Mexico
  </country>
  <country date="20070411" format="yyyyMMdd">
    Austria
  </country>
  <country date="20070412" format="yyyyMMdd">
    Sweden
  </country>
</countries>
```

The snippet below shows the corresponding XSL file for processing the XML file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:parse="parseDate"
                extension-element-prefixes="parse"
                version="1.0">

  <xalan:component prefix="parse" functions="parseDate">
    <xalan:script lang="javascript">

      var outputFormat = "dd/MM/yyyy";

      function parseDate(date, format) {
        parsedDate =
```

```
15              new java.text.SimpleDateFormat(format).parse(date);
16            return new java.text.SimpleDateFormat(outputFormat).
17              format(parsedDate);
18          }
19
20      </xalan:script>
21    </xalan:component>
22
23    <xsl:template match="/">
24      <timedCountries>
25        <xsl:apply-templates select="/countries/country" />
26      </timedCountries>
27    </xsl:template>
28
29    <xsl:template match="country">
30      <country>
31        <value>
32          <xsl:value-of select="." />
33        </value>
34        <xsl:variable name="date" select="string(@date)" />
35        <xsl:variable name="format" select="string(@format)" />
36        <date>
37          <xsl:value-of select="parse:parseDate($date, $format)" />
38        </date>
39
40      </country>
41    </xsl:template>
42
43  </xsl:stylesheet>
```

On line 3 the namespace for Xalan and on line 4 the namespace for the external function is defined. Preventing the namespace from being output to the resulting XML file, the namespace is marked to be excluded (line 5).

The Xalan component defines just one method “parseDate” (lines 8 to 21). This method receives two parameters: The first string contains the date and the second one indicates the format of the first parameter (line 13). The date is parsed using a SimpleDateFormat (lines 14 and 15) and formatted (lines 16 and 17) using the defined format on line 11.

On line 37 the “parseDate” method is called supplying both parameters date and format. The values of both parameters need to be saved into variables to prevent Xalan from passing on the references to the attributes on method execution (lines 34 and 35).

The XML file below shows the result of the transformation:

```
<?xml version="1.0" encoding="UTF-8"?>
<timedCountries xmlns:xalan="http://xml.apache.org/xalan">
  <country>
    <value>United States of America</value>
    <date>05/04/2007</date>
  </country>
  <country>
    <value>United Kingdom</value>
    <date>06/04/2007</date>
  </country>
  <country>
    <value>Brazil</value>
    <date>07/04/2007</date>
  </country>
  <country>
    <value>Cuba</value>
    <date>08/04/2007</date>
  </country>
  <country>
    <value>Chile</value>
    <date>09/04/2007</date>
  </country>
  <country>
    <value>Mexico</value>
    <date>10/04/2007</date>
  </country>
  <country>
    <value>Austria</value>
    <date>11/04/2007</date>
  </country>
  <country>
    <value>Sweden</value>
    <date>12/04/2007</date>
  </country>
</timedCountries>
```

3.3 Nutshells Using Existing Applications

This chapter shows the possibility to capitalise existing applications by using BSF and a scripting language.

3.3.1 Using JRuby's ActiveRecord

The following nutshell shows how an existing Ruby on Rails application can be reused through BSF and JRuby.

The existing database table “properties” is used from the nutshells “Querying Databases”. The input and output files are similar to the files from the nutshells “Querying Databases”.

At first the Ruby on Rails (RoR) application is created. The following code generates the complete RoR application for the table:

```
1  rails Properties
2  script/generate scaffold properties Properties
```

The first line shows the creation of an Rails application called “Properties”. This constitutes the frame for the RoR application. The second line uses the generators of RoR to create the Ruby files for the “properties” table which includes the ActiveRecord (properties.rb):

```
1  class Properties < ActiveRecord::Base
2  end
```

For more information on application development with Ruby on Rails see [ROR07].

Using this Ruby ActiveRecord with BSF, some porting lines need to be coded:

```
1  require "rubygems"
2  gem 'ActiveRecord-JDBC'
3  require 'jdbc_adapter'
4  require "active_record"
5
6  ActiveRecord::Base::establish_connection(
7    :adapter => 'jdbc',
8    :driver => 'org.postgresql.Driver',
9    :url => 'jdbc:postgresql://localhost:5432/xslt',
10   :username=>"xslt",
11   :password=>"xslt")
```

The lines 1 through 5 mark which Ruby gems are required. Line 3 shows that the “jdbc_adapter” gem is needed because JRuby runs on Java. Within Ruby a gem like “postgres” would be used which builds on C. As JRuby runs within the Java Virtual Machine the “jdbc_adapter” has to be used instead to query the database. The lines 6 to 11 configure the JDBC Adapter.

The following snippet shows how to initialise and call the Ruby code:

```
1  …
2  public class RubyCaller {
3    private BSFManager manager;
4
5    public void init(…) throws Exception {
6      String engine = "org.jruby.javasupport.bsf.JRubyEngine";
7      String[] extensions = {"rb"};
8      BSFManager.registerScriptingEngine("ruby", engine, extensions);
9
10     manager = new BSFManager();
11     this.execFile("nutshell3/port.rb");
12     this.execFile("nutshell3/Properties/app/models/properties.rb");
13   }
14
15   public String getValue(String key) throws Exception{
16     ValueBean valuebean = new ValueBean();
17     valuebean.setKey(key);
18     manager.declareBean("valuebean", valuebean, ValueBean.class);
19     Object object = manager.eval("ruby", "(java)", 1, 1,
20       "Properties.find($valuebean.getKey()).value");
21     return (String) object;
22   }
23 …
```

The class RubyCaller encapsulates the code required to interface with JRuby. The lines 6 to 11 register the scripting language of “ruby” with the BSFManager. The lines 11 and 12 execute the ruby scripts used.

The ruby script on line 11 contains the porting code shown above. The ruby file “properties.rb” was generated by RoR.

The method getValue() on line 15 receives a string containing the key, reads the corresponding Properties object and returns the value.

On line 18 a bean is declared with the manager and two lines below it is used (“$valuebean.getKey()”). Through the BSFManager API the script is invoked (line 19) submitting the declared parameter. The returned object coincides with the value attribute of the Properties object.

The RubyCaller class is used by the XSL file:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="at.ac.wuwien.xsltnutshell.RubyCaller"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
```

```
8
9   <xalan:component prefix="props"
10                   elements="init" functions="getValue">
11     <xalan:script lang="javaclass"
12       src="at.ac.wuwien.xsltnutshell.RubyCaller"/>
13   </xalan:component>
14
15   <xsl:template match="/">
16     <HTML>
17       <H1>Ruby Example</H1>
18       <props:init/>
19       <p>Here are the names of the Agencies:</p>
20       <xsl:for-each select="agencies/agency">
21         <xsl:sort select="@key"/>
22         <p>
23         <xsl:text>[</xsl:text>
24         <xsl:value-of select="@key"/>
25         <xsl:text>] </xsl:text>
26         <xsl:value-of select="props:getValue(@key)"/>
27         </p>
28       </xsl:for-each>
29     </HTML>
30   </xsl:template>
31
32 </xsl:stylesheet>
```

In lines 9 to 13 the RubyCaller class is defined. Line 18 initialises the component and finally the getValue() method is called at line 26.

The next snippet shows another way to call the Ruby script:

```
1  …
2  <xalan:component prefix="props"
3                   elements="init" functions="getValue">
4    <xalan:script lang="ruby" ><![CDATA[
5      require "rubygems"
6      gem 'ActiveRecord-JDBC'
7      require 'jdbc_adapter'
8      require "active_record"
9
10     ActiveRecord::Base::establish_connection(
11       :adapter => 'jdbc',
12       :driver => 'org.postgresql.Driver',
13       :url => 'jdbc:postgresql://localhost:5432/xslt',
14       :username=>"xslt",
15       :password=>"xslt")
16
17     class Properties < ActiveRecord::Base
18     end
19
20     def init (arg0, arg1)
21     end
```

```
22
23      def getValue (key)
24        return Properties.find(key).value
25      end
26    ]]>
27    </xalan:script>
28  </xalan:component>
29  …
```

The problem in this case is shown on lines 17 and 18. The existing code has to be duplicated to be callable. Therefore the first option would be the appropriate one if a Ruby on Rails application is already existing.

3.3.2 ResourceBundle Using Java

This nutshell shows the possibility of using a properties file through Java to get localised text strings.

The Java class reads the names of agencies from the properties file which are contained in the input XML file:

```
1   <?xml version="1.0"?>
2   <agencies>
3     <agency key="AMTRAK" />
4     <agency key="DARPA" />
5     <agency key="DCAA" />
6     <agency key="DEA" />
7     <agency key="DFAS" />
8     <agency key="DHS" />
9     <agency key="DIA" />
10    <agency key="DISA" />
11  …
12    <agency key="NASA" />
13    <agency key="NIH" />
14    <agency key="NIMH" />
15    <agency key="NOAA" />
16    <agency key="NSA" />
17    <agency key="USCIS" />
18    <agency key="USDA" />
19    <agency key="VA" />
20  </agencies>
```

Each value of a “key” attribute corresponds to an entry in the properties file. The following snippet shows some entries:

```
AMTRAK=National Railroad Passenger Corporation
DARPA=Defense Advanced Research Projects Agency
DCAA=Defense Contract Audit Agency
DEA=Drug Enforcement Administration
DFAS=Defense Finance and Accounting Service
DHS=Department of Homeland Security
DIA=Defense Intelligence Agency
DISA=Defense Information Systems Agency
DLA=Defense Logistics Agency
DOC=Department of Commerce
DOD=Department of Defense
FBI=Federal Bureau of Investigation
…
NASA=National Aeronautics and Space Administration
NEI=National Eye Institute
NIAAA=National Institute on Alcohol Abuse and Alcoholism
NIDA=National Institute on Drug Abuse
NIDCD=National Institute of Deafness and Other Communication Disorders
NIDCR=National Institute of Dental and Craniofacial Research
NIDDK=National Institute of Diabetes and Digestive and Kidney Diseases
NOAA=National Oceanic and Atmospheric Administration
NSA=National Security Agency
USCIS=U.S. Citizenship and Immigration Services
USDA=Department of Agriculture
VA=Department of Veterans Affairs
```

An external function implemented in Java is used to get the corresponding text string from the properties file. The following XSL file is used for processing the XML:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="at.ac.wuwien.xsltnutshell.PropertiesQuery"
                extension-element-prefixes="props"
                version="1.0">


 <xalan:component prefix="props" functions="getValue">
   <xalan:script lang="javaclass"
       src="at.ac.wuwien.xsltnutshell.PropertiesQuery"/>
 </xalan:component>

 <xsl:template match="/">
   <HTML>
     <H1>Java Example</H1>
     <p>Here are the names of the Agencies:</p>
     <xsl:for-each select="agencies/agency">
       <xsl:sort select="@key"/>
       <p>
```

```
21          <xsl:text>[</xsl:text>
22          <xsl:value-of select="@key"/>
23          <xsl:text>] </xsl:text>
24          <xsl:value-of select="props:getValue(@key)"/>
25          </p>
26       </xsl:for-each>
27     </HTML>
28   </xsl:template>
29
30 </xsl:stylesheet>
```

The script above processes the XML file and retrieves the corresponding text string for each key entry. The result is written to the XML file.

On line 3 the Xalan-Java namespace and on line 4 the namespace for the external function are defined. Beneath the component is defined as a Java class (lines 9 to 12).

On line 24 the method “getValue” is executed. It is not necessary to save the attributes value in between as Java is used.

The implementation below shows the used Java class:

```
1  package at.ac.wuwien.xsltnutshell;
2
3  import java.util.ResourceBundle;
4
5  public class PropertiesQuery {
6
7      ResourceBundle bundle =
8        ResourceBundle.getBundle("at.ac.wuwien.xsltnutshell.agencies");
9
10     public String getValue(String key) {
11
12        return bundle.getString(key);
13
14     }
15
16 }
```

The Java class instantiates a ResourceBundle (lines 7 to 8) with the properties file mentioned above. Every time the method “getValue” is executed it retrieves the corresponding text string to the supplied key from the ResourceBundle (line 12).

The result is returned to Xalan (line 12).

The resulting XML file contains the key from the input XML file and the text string from the properties file:

```
<HTML>
<H1>Java Example</H1>
<p>Here are the names of the Agencies:</p>
<p>[AMTRAK] National Railroad Passenger Corporation</p>
<p>[DARPA] Defense Advanced Research Projects Agency</p>
<p>[DCAA] Defense Contract Audit Agency</p>
<p>[DEA] Drug Enforcement Administration</p>
<p>[DOI] Department of the Interior</p>
<p>[DOJ] Department of Justice</p>
<p>[ED] Department of Education</p>
<p>[EEOC] Equal Employment Opportunity Commission</p>
<p>[EPA] Environmental Protection Agency</p>
<p>[FAA] Federal Aviation Administration</p>
<p>[FBI] Federal Bureau of Investigation</p>
…
<p>[NASA] National Aeronautics and Space Administration</p>
<p>[NEI] National Eye Institute</p>
<p>[NIAAA] National Institute on Alcohol Abuse and Alcoholism</p>
<p>[NIST] National Institute of Standards and Technology</p>
<p>[NOAA] National Oceanic and Atmospheric Administration</p>
<p>[NSA] National Security Agency</p>
<p>[USCIS] U.S. Citizenship and Immigration Services</p>
<p>[USDA] Department of Agriculture</p>
<p>[VA] Department of Veterans Affairs</p>
</HTML>
```

3.3.3 Querying Calc Using BSF4Rexx

OpenOffice.org offers an API to create and manipulate a Calc file. This nutshell uses an existing Calc file to search for a string using an OORexx script. Therefore this nutshell uses BSF4Rexx and the API of OpenOffice.org to access the Calc file.

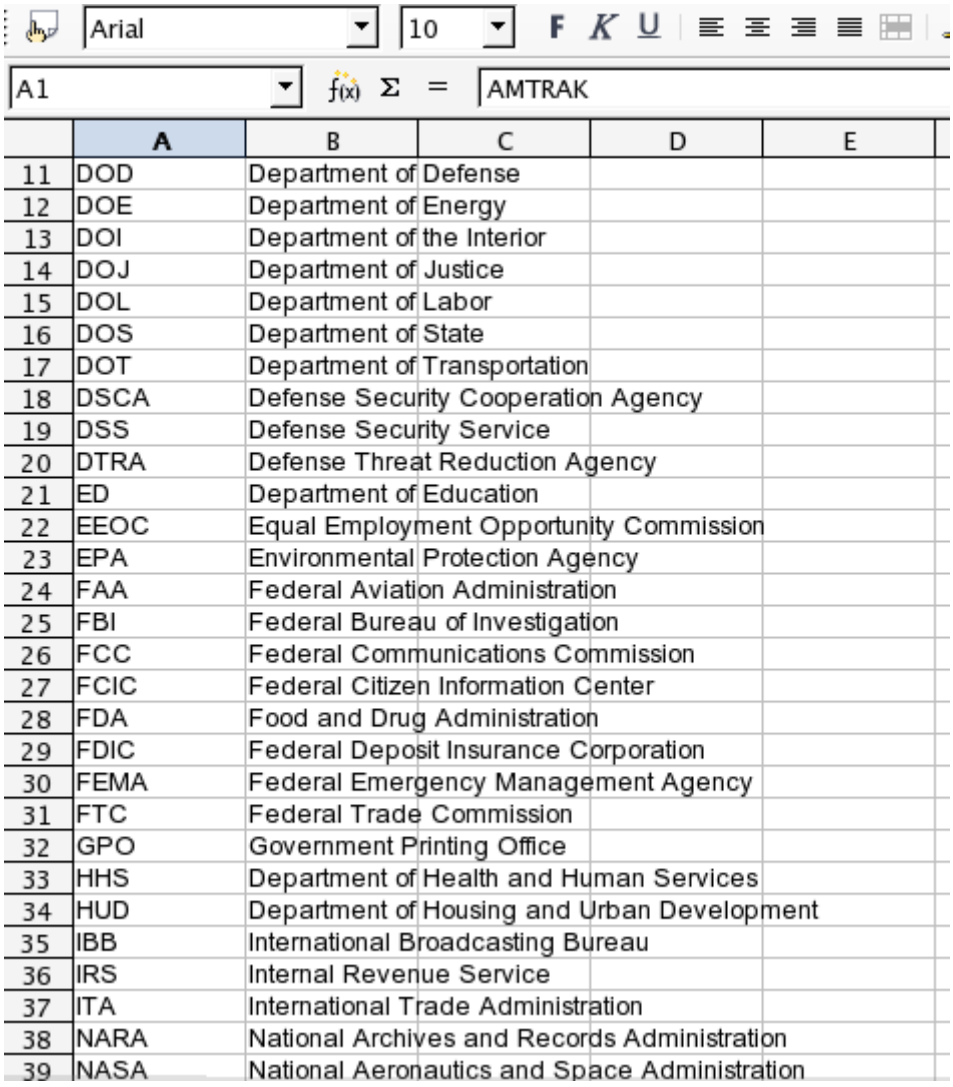
The key strings searched within the Calc file are supplied to the transformation by an input XML file:

```
<?xml version="1.0"?>
<agencies>
  <agency key="AMTRAK" />
  <agency key="DARPA" />
  <agency key="DCAA" />
  <agency key="DEA" />
…
  <agency key="NIGMS" />
  <agency key="NIH" />
  <agency key="NIMH" />
  <agency key="NINDS" />
  <agency key="NIST" />
  <agency key="NOAA" />
```

```
14    <agency key="NSA" />
15    <agency key="USCIS" />
16    <agency key="USDA" />
17    <agency key="VA" />
18  </agencies>
```

The content of each “key” attribute is looked up in the Calc file. The Calc file contains two columns. The first column contains the agencies key from the XML file while the second column states the names of the agencies.

The following picture shows a snapshot of the Calc file:



Arial | 10 | A1 | AMTRAK

	A	B	C	D	E
11	DOD	Department of Defense			
12	DOE	Department of Energy			
13	DOI	Department of the Interior			
14	DOJ	Department of Justice			
15	DOL	Department of Labor			
16	DOS	Department of State			
17	DOT	Department of Transportation			
18	DSCA	Defense Security Cooperation Agency			
19	DSS	Defense Security Service			
20	DTRA	Defense Threat Reduction Agency			
21	ED	Department of Education			
22	EEOC	Equal Employment Opportunity Commission			
23	EPA	Environmental Protection Agency			
24	FAA	Federal Aviation Administration			
25	FBI	Federal Bureau of Investigation			
26	FCC	Federal Communications Commission			
27	FCIC	Federal Citizen Information Center			
28	FDA	Food and Drug Administration			
29	FDIC	Federal Deposit Insurance Corporation			
30	FEMA	Federal Emergency Management Agency			
31	FTC	Federal Trade Commission			
32	GPO	Government Printing Office			
33	HHS	Department of Health and Human Services			
34	HUD	Department of Housing and Urban Development			
35	IBB	International Broadcasting Bureau			
36	IRS	Internal Revenue Service			
37	ITA	International Trade Administration			
38	NARA	National Archives and Records Administration			
39	NASA	National Aeronautics and Space Administration			

Figure 3.1.: Calc File Used for Look-up

The OORexx script used in this transformation is embedded within the XSL file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="CalcDocument"
                extension-element-prefixes="props"
                version="1.0">

 <xalan:component prefix="props" functions="getValue">
   <xalan:script lang="rexx">
   <![CDATA[
     parse arg key
     if key =="" then return ""
     call init
     i = getValue(key)
     return i

     init:
       -- Retrieve the Desktop object, we need its XComponentLoader
       --interface to load a new document
       oDesktop = UNO.createDesktop()
       xComponentLoader = oDesktop~XDesktop~XComponentLoader

       /* open the "properties.ods" from current folder/directory */
       url = ConvertToURL(directory() || "/nutshell6/properties.ods")
       xCalcComponent = xComponentLoader~loadComponentFromURL
            (url, "_blank", 0, .UNO~noProps)

       /* get first sheet in spreadsheet */
       xSheet = xCalcComponent~XSpreadSheetDocument~getSheets~
              XIndexAccess~getByIndex(0) ~XSpreadSheet
     return

     /*   this method searches the first column of the calc file for
          the supplied value.if found the value of the second column
          is returned */
     getValue:
     parse arg key

       output = ""
       eof = 0
       row = 0

       do while eof = 0
         -- get the first cell of the next row
         value = UNO.getCell(xSheet, 0, row)~getFormula()

         -- last row already passed and nothing found
         if value = "" then do
           eof = 1
         end
         -- key found and value of second cell returned
```

```
52              else if value = key then do
53                eof = 1
54                output = UNO.getCell(xSheet, 1, row)~getFormula()
55              end
56              -- not found and EOF not reached, continue searching
57              else do
58                row = row + 1
59              end
60            end
61          return output
62
63          ::requires UNO.CLS    -- get UNO support
64        ]]>
65        </xalan:script>
66    </xalan:component>
67
68    <xsl:template match="/">
69      <HTML>
70        <H1>ooRexx Example</H1>
71        <p>Here are the names of the Agencies:</p>
72        <xsl:for-each select="agencies/agency">
73          <xsl:sort select="@key"/>
74          <p>
75          <xsl:text>[</xsl:text>
76          <xsl:value-of select="@key"/>
77          <xsl:text>] </xsl:text>
78          <xsl:variable name="keyString">
79            <xsl:value-of select="@key"/>
80          </xsl:variable>
81          <xsl:value-of select="props:getValue($keyString)"/>
82          </p>
83        </xsl:for-each>
84      </HTML>
85    </xsl:template>
86
87  </xsl:stylesheet>>
```

OpenOffice.org supplies many functions which may be used in OORexx through the UNO.CLS library (line 20). This nutshell uses UNO.CLS to open an existing Calc file in OpenOffice.org and to search it for supplied strings.

Executing many function calls to a slowly initialising library implicates that the initialising process should only be executed once for the complete handling. Therefore this script implements an “init” method to get a reference to the opened Calc file (lines 17 to 31) and an extra method called “getValue” which searches each string separately within the Calc file (lines 36 to 61).

The script above works as intended if executed as an OORexx command from the command line. As soon as it is embedded into the XSLT stylesheet it behaves differently. Independent which method is called in the XSLT stylesheet (like on line 81), the complete script is executed.

Therefore the lines 11 to 15 are included. The parameter supplied by Xalan is stored (line 11) and used to call the method “getValue” (line 14) after the initialisation is completed (line 13).

Additionally Xalan parses the script by executing it with no parameter which leads to a NullPointerException. Avoiding this, the code on line 12 is inserted to the script.

The XSLT stylesheet looks similar to the other nutshells using existing applications. The Xalan namespace is set on line 3 and on line 4 “props” is chosen as the namespace for the external function.

The lines 8 and 9 specify the component to be an OORexx script with one method “getValue”. This shows that the name of the method (line 36) is independent from the called method (line 81).

This nutshell uses the library UNO.CLS [B4R07] which is installed with BSF4Rexx (line 63). The library offers an API to connect to the OpenOffice.org application using a XComponentLoader (line 21). In this nutshell the Calc file is opened through the use of the XComponentLoader (lines 21 to 30).

The method “getValue” iterates through all entries in the first column (lines 43 to 60). It reads the value (line 45) and compares it with the supplied key. If both strings match, the value from the second column is returned (line 54 and 61).

The lines 78 to 80 store the value of the attribute “key” in the variable “keyString” which is used to call the script on line 80. Without saving the value, Xalan would provide the reference to the attribute node instead of the value.

The following table shows the resulting content of the transformation:

```
 1  <HTML>
 2  <H1>ooRexx Example</H1>
 3  <p>Here are the names of the Agencies:</p>
 4  <p>[AMTRAK] National Railroad Passenger Corporation</p>
 5  <p>[DARPA] Defense Advanced Research Projects Agency</p>
 6  <p>[DCAA] Defense Contract Audit Agency</p>
 7  <p>[DEA] Drug Enforcement Administration</p>
 8  …
 9  <p>[NIGMS] National Institute of General Medical Sciences</p>
10  <p>[NIH] National Institutes of Health</p>
11  <p>[NIMH] National Institute of Mental Health</p>
12  <p>[NINDS] National Institute of Neurological Disorders and Stroke</p>
13  <p>[NIST] National Institute of Standards and Technology</p>
14  <p>[NOAA] National Oceanic and Atmospheric Administration</p>
15  <p>[NSA] National Security Agency</p>
16  <p>[USCIS] U.S. Citizenship and Immigration Services</p>
17  <p>[USDA] Department of Agriculture</p>
18  <p>[VA] Department of Veterans Affairs</p>
19  </HTML>
```

One problem emerges after the transformation finishes. The transformation process does not get terminated after the resulting file is written. Instead it has to be killed. Furthermore the started instance of OpenOffice.org stays open even after the killing of the transformation process.

3.3.4 ResourceBundle Using JavaScript

This nutshell uses JavaScript to read localised text strings from a properties file.

The keys for the JavaScript code are supplied by the following XML snippet:

```
<?xml version="1.0"?>
<agencies>
  <agency key="AMTRAK" />
  <agency key="DARPA" />
  <agency key="DCAA" />
  <agency key="DEA" />
  <agency key="DFAS" />
  <agency key="DHS" />
  <agency key="DIA" />
  <agency key="DISA" />
…
  <agency key="NASA" />
  <agency key="NIH" />
  <agency key="NIMH" />
  <agency key="NOAA" />
  <agency key="NSA" />
  <agency key="USCIS" />
  <agency key="USDA" />
  <agency key="VA" />
</agencies>
```

The following snippet shows some entries from the properties file used in this nutshell:

```
AMTRAK=National Railroad Passenger Corporation
DARPA=Defense Advanced Research Projects Agency
DCAA=Defense Contract Audit Agency
DEA=Drug Enforcement Administration
DFAS=Defense Finance and Accounting Service
DHS=Department of Homeland Security
DIA=Defense Intelligence Agency
DISA=Defense Information Systems Agency
DLA=Defense Logistics Agency
DOC=Department of Commerce
DOD=Department of Defense
FBI=Federal Bureau of Investigation
…
NASA=National Aeronautics and Space Administration
NEI=National Eye Institute
NIAAA=National Institute on Alcohol Abuse and Alcoholism
NIDA=National Institute on Drug Abuse
NIDCD=National Institute of Deafness and Other Communication Disorders
```

```
NIDCR=National Institute of Dental and Craniofacial Research
NIDDK=National Institute of Diabetes and Digestive and Kidney Diseases
NOAA=National Oceanic and Atmospheric Administration
NSA=National Security Agency
USCIS=U.S. Citizenship and Immigration Services
USDA=Department of Agriculture
VA=Department of Veterans Affairs
```

Each key in the XML file corresponds to a property from the file above. An external function implemented in JavaScript is used to get the corresponding text string from the properties file. The JavaScript code needed for the processing is embedded within the XSL file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="ResourceBundle"
                extension-element-prefixes="props"
                version="1.0">

 <xalan:component prefix="props" functions="getValue">
   <xalan:script lang="javascript">

      var bundle = java.util.ResourceBundle.getBundle
         ('at.ac.wuwien.xsltnutshell.agencies');

      function getValue(key) {
         return bundle.getString(key);
      }

   </xalan:script>
 </xalan:component>

 <xsl:template match="/">
   <HTML>
     <H1>JavaScript Example</H1>
     <p>Here are the names of the Agencies:</p>
     <xsl:for-each select="agencies/agency">
       <xsl:sort select="@key"/>
       <p>
       <xsl:text>[</xsl:text>
       <xsl:value-of select="@key"/>
       <xsl:text>] </xsl:text>
       <xsl:variable name="keyString">
         <xsl:value-of select="@key"/>
       </xsl:variable>
       <xsl:value-of select="props:getValue($keyString)"/>
       </p>
     </xsl:for-each>
   </HTML>
 </xsl:template>

</xsl:stylesheet>
```

The script above processes the XML file and retrieves the corresponding text string for each key entry. The result is written to the XML file.

On line 3 the Xalan-Java namespace and on line 4 the namespace for the external function are defined. Beneath the component is defined to be implemented in JavaScript (lines 8 to 19).

On line 34 the method “getValue” is executed. As the calling script is implemented in JavaScript the value of the attribute has to be buffered in a variable (lines 31 to 33).

The JavaScript code instantiates a ResourceBundle (lines 11 to 12) with the properties file mentioned above. Every time the method “getValue” is executed it retrieves the corresponding text string to the supplied key from the ResourceBundle (line 15) and returns it.
The following XML snippet shows a part of the output from the transformation:

```
 1  <HTML>
 2  <H1>JavaScript Example</H1>
 3  <p>Here are the names of the Agencies:</p>
 4  <p>[AMTRAK] National Railroad Passenger Corporation</p>
 5  <p>[DARPA] Defense Advanced Research Projects Agency</p>
 6  <p>[DCAA] Defense Contract Audit Agency</p>
 7  <p>[DEA] Drug Enforcement Administration</p>
 8  <p>[DFAS] Defense Finance and Accounting Service</p>
 9  <p>[DHS] Department of Homeland Security</p>
10  <p>[DIA] Defense Intelligence Agency</p>
11  <p>[DISA] Defense Information Systems Agency</p>
12  …
13  <p>[NASA] National Aeronautics and Space Administration</p>
14  <p>[NEI] National Eye Institute</p>
15  <p>[NIAAA] National Institute on Alcohol Abuse and Alcoholism</p>
16  <p>[NIGMS] National Institute of General Medical Sciences</p>
17  <p>[NIH] National Institutes of Health</p>
18  <p>[NSA] National Security Agency</p>
19  <p>[USCIS] U.S. Citizenship and Immigration Services</p>
20  <p>[USDA] Department of Agriculture</p>
21  <p>[VA] Department of Veterans Affairs</p>
22  </HTML>
```

3.4 Nutshells Calling Web Services

Web services offer a standardised method for exposing functionality to the web. These nutshells use Axis 1.4 [AXI07] to call a web service provided by Jensen Data Systems [JDI07] which converts temperatures from Fahrenheit to Celsius.

3.4.1 Nutshell Using Java

This nutshell shows the possibility of calling a web service through Java and Axis 1.4 [AXI07]. As an example a web service converting between Fahrenheit and Celsius is used.

For this nutshell the following XML file is used as input:

```
1  <?xml version="1.0"?>
2  <temperatures>
3    <city name="Los Angeles" temperature="63"/>
4    <city name="Bakersfield" temperature="75"/>
5    <city name="Burbank" temperature="61"/>
6    <city name="Fresno" temperature="77"/>
7    <city name="Long Beach" temperature="67"/>
8    <city name="Palm Springs" temperature="85"/>
9    <city name="Crescent City" temperature="52"/>
10   <city name="Needles" temperature="95"/>
11 </temperatures>
```

The temperature in the XML file is supplied in Fahrenheit. The XSL file for processing the XML file contains the reference to the Java class calling the web service using Axis:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="at.ac.wuwien.xsltnutshell.WebServiceCaller"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8    <xalan:component prefix="props"
9                     elements="init" functions="getCelsius">
10     <xalan:script lang="javaclass"
11         src="at.ac.wuwien.xsltnutshell.WebServiceCaller"/>
12   </xalan:component>
13
14   <xsl:template match="/">
15     <HTML>
16       <H1>Java Example</H1>
17       <props:init/>
18       <p>Here are the temperatures of the Cities:</p>
19       <xsl:for-each select="temperatures/city">
20         <xsl:sort select="@name"/>
21         <p>
22         <xsl:text>[</xsl:text>
```

```
23          <xsl:value-of select="@name"/>
24          <xsl:text>] </xsl:text>
25          Temperature <xsl:value-of select="@temperature"/> °F -
26          <xsl:value-of select="props:getCelsius(@temperature)"/> °C
27          </p>
28        </xsl:for-each>
29      </HTML>
30    </xsl:template>
31
32  </xsl:stylesheet>
```

The script above reads the XML file, converts the temperatures to Celsius and outputs the result. This is accomplished by a call to the web service through Java. Therefore a component for Xalan is defined which points to the implementing Java class (lines 8 to 12).

On line 17 the initialisation routine of the Java class is called and finally at line 26 the method "getCelsius" is executed. As Java is used it is not necessary to save the attribute value in between.

The Java class implements the methods "init" and "getCelsius":

```
1   package at.ac.wuwien.xsltnutshell;
2   import java.util.HashMap;
3
4   public class WebServiceCaller {
5
6     DynamicInvoker dynamicInvoker = null;
7
8     public void init(org.apache.xalan.extensions.XSLProcessorContext
9           context, org.w3c.dom.Element elem) {
10      try {
11        dynamicInvoker = new DynamicInvoker("http://developerdays.com/cgi-
12              bin/tempconverter.exe/wsdl/ITempConverter");
13      } catch (Exception e) {
14        e.printStackTrace();
15      }
16    }
17
18    public String getCelsius(String fahrenheit) {
19      try {
20        HashMap map = dynamicInvoker.invokeMethod("FtoC",
21          "ITempConverterPort", new String[] {null, null, fahrenheit});
22        return String.valueOf(map.get("return"));
23      } catch (Exception e) {
24        e.printStackTrace();
25      }
26      return "<<" + fahrenheit + ">>";
27    }
28  }
```

The Java class uses the DynamicInvoker provided by Apache Axis (lines 6 and 11). This class facilitates the access of web services by supplying a easy API. On line 11 the WSDL is connected to the invoker which is afterwards used to invoke the method (line 20).

The result of the invocation is returned as a HashMap containing the key “return” (line 22).

The following XML file shows the output from the transformation containing the temperatures in Fahrenheit and Celsius:

```
1  <HTML>
2    <H1>Java Example</H1>
3    <p>Here are the temperatures of the Cities:</p>
4    <p>[Bakersfield] Temperature 75 °F - 23 °C</p>
5    <p>[Burbank] Temperature 61 °F - 16 °C</p>
6    <p>[Crescent City] Temperature 52 °F - 11 °C</p>
7    <p>[Fresno] Temperature 77 °F - 25 °C</p>
8    <p>[Long Beach] Temperature 67 °F - 19 °C</p>
9    <p>[Los Angeles] Temperature 63 °F - 17 °C</p>
10   <p>[Needles] Temperature 95 °F - 35 °C</p>
11   <p>[Palm Springs] Temperature 85 °F - 29 °C</p>
12 </HTML>
```

3.4.2 Nutshell Using JudoScript

This nutshell shows the possibility of calling a web service using JudoScript. As an example a web service which converts between Fahrenheit and Celsius is used.

For this nutshell the following XML file is used as input:

```
1  <?xml version="1.0"?>
2  <temperatures>
3    <city name="Los Angeles" temperature="63"/>
4    <city name="Bakersfield" temperature="75"/>
5    <city name="Burbank" temperature="61"/>
6    <city name="Fresno" temperature="77"/>
7    <city name="Long Beach" temperature="67"/>
8    <city name="Palm Springs" temperature="85"/>
9    <city name="Crescent City" temperature="52"/>
10   <city name="Needles" temperature="95"/>
11 </temperatures>
```

The temperature in the XML file is supplied in Fahrenheit.

The XSL file for processing the XML file embeds the JudoScript code:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="WebserviceCall"
                extension-element-prefixes="props"
                version="1.0">

 <xalan:component prefix="props"
                  elements="init" functions="getCelsius">
   <xalan:script lang="judoscript">

     wsdl = null;

     function init (xslproc, elem) {
       wsdl = wsdl::'http://developerdays.com/cgi-
                bin/tempconverter.exe/wsdl/ITempConverter';
       return null;
     }

     function getCelsius(fahrenheit) {
       return wsdl.FtoC(fahrenheit);
     }
   </xalan:script>
 </xalan:component>

 <xsl:template match="/">
   <HTML>
     <H1>JudoScript Example</H1>
     <props:init/>
     <p>Here are the temperatures of the Cities:</p>
     <xsl:for-each select="temperatures/city">
       <xsl:sort select="@name"/>
       <p>
       <xsl:text>[</xsl:text>
       <xsl:value-of select="@name"/>
       <xsl:text>] </xsl:text>
       <xsl:variable name="temperatureString">
         <xsl:value-of select="@temperature"/>
       </xsl:variable>
       Temperature <xsl:value-of select="@temperature"/> °F -
       <xsl:value-of select="props:getCelsius($temperatureString)"/>
       °C
       </p>
     </xsl:for-each>
   </HTML>
 </xsl:template>

</xsl:stylesheet>
```

The script above reads the XML file, converts the temperatures to Celsius and outputs the result. Calling a web service with JudoScript, the WSDL file needs to be linked to the "wsdl" domain. This is accomplished on lines 15 and 16. Afterwards the web service can be called like a method on the "wsdl" object (line 21). Internally JudoScript uses Axis 1.4 [AXI07] to call the web service.

The workaround saving the value of the attribute on lines 37 to 39 is required. Otherwise the reference to the attribute would be supplied instead of the value.

The following XML file shows the output from the transformation:

```
1  <HTML>
2    <H1>JudoScript Example</H1>
3    <p>Here are the temperatures of the Cities:</p>
4    <p>[Bakersfield] Temperature 75 °F - 23 °C</p>
5    <p>[Burbank] Temperature 61 °F - 16 °C</p>
6    <p>[Crescent City] Temperature 52 °F - 11 °C</p>
7    <p>[Fresno] Temperature 77 °F - 25 °C</p>
8    <p>[Long Beach] Temperature 67 °F - 19 °C</p>
9    <p>[Los Angeles] Temperature 63 °F - 17 °C</p>
10   <p>[Needles] Temperature 95 °F - 35 °C</p>
11   <p>[Palm Springs] Temperature 85 °F - 29 °C</p>
12 </HTML>
```

3.4.3 Nutshell Using JavaScript

Web services provide functionality to the internet through the SOAP protocol. In this nutshell the functionality of a web service which converts between Fahrenheit and Celsius is used. The calling code is implemented with JavaScript.

The input XML file contains the name and the temperature of Californian cities:

```
1  <?xml version="1.0"?>
2  <temperatures>
3    <city name="Los Angeles" temperature="63"/>
4    <city name="Bakersfield" temperature="75"/>
5    <city name="Burbank" temperature="61"/>
6    <city name="Fresno" temperature="77"/>
7    <city name="Long Beach" temperature="67"/>
8    <city name="Palm Springs" temperature="85"/>
9    <city name="Crescent City" temperature="52"/>
10   <city name="Needles" temperature="95"/>
11 </temperatures>
```

The temperature in the XML file is supplied in Fahrenheit.

The web service calling code in JavaScript is embedded within the XSL file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xalan="http://xml.apache.org/xalan"
                xmlns:props="WebserviceCall"
                extension-element-prefixes="props"
                version="1.0">

 <xalan:component prefix="props"
                  elements="init" functions="getCelsius">
   <xalan:script lang="javascript">

     importPackage(Packages.at.ac.wuwien.xsltnutshell)
     var dynamicInvoker = null;

     function init (xslproc, elem) {
       dynamicInvoker = new DynamicInvoker(
         "http://developerdays.com/cgi-
         bin/tempconverter.exe/wsdl/ITempConverter");
       return null;
     }

     function getCelsius(fahrenheit) {
       var stringArray =
         java.lang.reflect.Array.newInstance(java.lang.String, 3);
       stringArray[2] = fahrenheit;
       var map = dynamicInvoker.invokeMethod("FtoC",
         "ITempConverterPort", stringArray);
       return map.get("return");
     }
   </xalan:script>
 </xalan:component>

 <xsl:template match="/">
   <HTML>
     <H1>JavaScript Example</H1>
     <props:init/>
     <p>Here are the temperatures of the Cities:</p>
     <xsl:for-each select="temperatures/city">
       <xsl:sort select="@name"/>
       <p>
       <xsl:text>[</xsl:text>
       <xsl:value-of select="@name"/>
       <xsl:text>] </xsl:text>
       <xsl:variable name="temperatureString">
         <xsl:value-of select="@temperature"/>
       </xsl:variable>
       Temperature <xsl:value-of select="@temperature"/> °F -
       <xsl:value-of select="props:getCelsius($temperatureString)"/>
       °C
       </p>
     </xsl:for-each>
```

```
52      </HTML>
53    </xsl:template>
54
55  </xsl:stylesheet>
```

The script above reads the XML file, converts the temperatures to Celsius and generates the result. The JavaScript code uses the same DynamicInvoker class as the Java version. The web service is called using Axis 1.4 [AXI07] which facilitates the execution.

Lines 15 to 20 show how the DynamicInvoker is instantiated through JavaScript and further down on lines 26 and 27 how a method is invoked. The lines 23 to 25 create an array of string objects and populate the last one with the parameter in Fahrenheit.

The workaround saving the value of the attribute on lines 44 to 46 is needed. Otherwise the reference to the attribute would be supplied instead of the value.

The resulting XML file contains the name of the city and the temperatures in Fahrenheit and Celsius:

```
 1  <HTML>
 2    <H1>JavaScript Example</H1>
 3    <p>Here are the temperatures of the Cities:</p>
 4    <p>[Bakersfield] Temperature 75 °F - 23 °C</p>
 5    <p>[Burbank] Temperature 61 °F - 16 °C</p>
 6    <p>[Crescent City] Temperature 52 °F - 11 °C</p>
 7    <p>[Fresno] Temperature 77 °F - 25 °C</p>
 8    <p>[Long Beach] Temperature 67 °F - 19 °C</p>
 9    <p>[Los Angeles] Temperature 63 °F - 17 °C</p>
10    <p>[Needles] Temperature 95 °F - 35 °C</p>
11    <p>[Palm Springs] Temperature 85 °F - 29 °C</p>
12  </HTML>
```

3.5 How to Upgrade to BSF 3.0

This chapter focuses on the process of upgrading existing nutshells from BSF 2.4 to BSF version 3.0.

BSF 3.0 implements the JSR 223 specification [JCP07]. The API of BSF 3.0 is completely different compared to the version 2.4 of BSF.

The following two sub chapters explain how a BSF 2.4 compatible Xalan XSL file can be upgraded to BSF 3.0. They differ in the way such XSL files are implemented.

3.5.1 Upgrading XSL Files with Embedded Scripts

The most scripts in this bachelor thesis are implemented using embedded scripts. The following snippet shows a nutshell using embedded JavaScript code:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:parse="parseDate"
5                  extension-element-prefixes="parse"
6                  version="1.0">
7
8    <xalan:component prefix="parse" functions="parseDate">
9      <xalan:script lang="javascript">
10
11       var outputFormat = "dd/MM/yyyy";
12
13       function parseDate(date, format) {
14         parsedDate =
15           new java.text.SimpleDateFormat(format).parse(date);
16         return new java.text.SimpleDateFormat(outputFormat).
17           format(parsedDate);
18       }
19
20     </xalan:script>
21   </xalan:component>
22
23 …
24 </xsl:stylesheet>
```

An embedded script is contained within the Xalan script tag as shown on lines 9 to 20. Porting such XSL files is straight forward by replacing the bsf.jar with the new version bsf-all-3.0-beta1.jar. Such embedded scripts work out of the box after exchanging the jar file.

This works because Xalan acts as a kind of abstraction to the invocation of BSF and therefore automatically invokes the script through BSF using the appropriate API.

3.5.2 Upgrading XSL Files with Separate Scripts

The second possibility to call external functions is by supplying a separate file which is called by Xalan. The following snippet shows the definition of an external function which is stored in separate files using Ruby:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="at.ac.wuwien.xsltnutshell.RubyCaller"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8    <xalan:component prefix="props"
9                     elements="init" functions="getValue">
10     <xalan:script lang="javaclass"
11       src="at.ac.wuwien.xsltnutshell.RubyCaller"/>
12   </xalan:component>
13 …
14 </xsl:stylesheet>
```

Lines 10 to 12 show the definition of the separate file “at.ac.wuwien.xsltnutshell.RubyCaller”.

The following code shows the file RubyCaller:

```
1  …
2  public class RubyCaller {
3    private BSFManager manager;
4
5    public void init(…) throws Exception {
6      String engine = "org.jruby.javasupport.bsf.JRubyEngine";
7      String[] extensions = {"rb"};
8      BSFManager.registerScriptingEngine("ruby", engine, extensions);
9
10     manager = new BSFManager();
11     this.execFile("nutshell3/port.rb");
12     this.execFile("nutshell3/Properties/app/models/properties.rb");
13   }
14
15   public String getValue(String key) throws Exception{
16     ValueBean valuebean = new ValueBean();
17     valuebean.setKey(key);
18     manager.declareBean("valuebean", valuebean, ValueBean.class);
19     Object object = manager.eval("ruby", "(java)", 1, 1,
20       "Properties.find($valuebean.getKey()).value");
21     return (String) object;
22   }
23 …
```

This file is implemented using the BSF 2.4 API. Every exchange between the scripting language and the calling Java code is done through the class BSFManager. On line 8 the scripting engine is registered and on line 10 the BSFManager is instantiated. Now the scripting engine is ready to receive processing instructions from Java.

Calling a function within the scripting language, the API of the BSFManager is used (lines 19 and 20). The “eval” method of the BSFManager evaluates the supplied string and returns the result. In the example above the method “find” of the Ruby ActiveRecord “Properties” is called.

Furthermore it is possible to pass on a value as a bean to the context of the scripting language. Every bean in the context can be accessed as variable within the called script. On line 18 the bean called “valuebean” is stored in the context using the “declareBean” method of the BSFManager.

The implementation for BSF version 3.0 differs in the API used:

```
1  package at.ac.wuwien.xsltnutshell;
2
3  import java.io.FileReader;
4  import javax.script.ScriptEngine;
5  import javax.script.ScriptEngineManager;
6
7  public class RubyCallerBSF30 {
8   private ScriptEngine rubyengine;
9
10  public void init(org.apache.xalan.extensions.XSLProcessorContext
11            context, org.w3c.dom.Element elem) throws Exception {
12     ScriptEngineManager manager = new ScriptEngineManager();
13     rubyengine = manager.getEngineByName("ruby");
14     rubyengine.eval(new FileReader("nutshell3/port.rb"));
15     rubyengine.eval(new FileReader(
16               "nutshell3/Properties/app/models/properties.rb"));
17  }
18
19  public String getValue(String key) throws Exception{
20     rubyengine.put("valuebean", key);
21     Object object =
22            rubyengine.eval( "Properties.find($valuebean).value");
23     return (String) object;
24  }
25 }
```

The main focus in the code above is on the shortage and cleanliness. This code uses the JSR 223 API to communicate with the scripting engine. The class ScriptEngine serves the same purpose as the BSFManager did in the BSF 2.4 version. The only difference is that the ScriptEngineManager is required to instantiate the engine (line 13).

Again the “eval” method is used to execute a script written in a supported scripting language (lines 14, 15 and 22).

Passing on a bean from Java to the script, it is connected with the scripting engine through their “put” method as if it was a HashMap (line 20). Within the script code the bean is exposed as variable (line 22 for a Ruby variable example).

Finally the snippet of the corresponding XSL file including the changes to call the BSF 3.0 version of RubyCaller (RubyCallerBSF30) is shown below:

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                  xmlns:xalan="http://xml.apache.org/xalan"
4                  xmlns:props="at.ac.wuwien.xsltnutshell.RubyCallerBSF30"
5                  extension-element-prefixes="props"
6                  version="1.0">
7
8    <xalan:component prefix="props"
9                     elements="init" functions="getValue">
10     <xalan:script lang="javaclass"
11          src="at.ac.wuwien.xsltnutshell.RubyCallerBSF30"/>
12   </xalan:component>
13 …
14 </xsl:stylesheet>
```

Only the name of the Java class containing the call to Ruby through BSF 3.0 changed (lines 4 and 11).

4 Conclusion

In business nowadays XML plays an important role for exchanging data. It both represents the data in machine and human readable form. Changes within the structure and node set of the XML file remain complex and difficult to achieve using standard XML manipulating APIs.

Facilitating this process, the World Wide Web Consortium (W3C) [W3C03a] develops and maintains the XSL specification which aims to standardise the transformation process.

This bachelor thesis deals with the transformation specification of XSL called XSL Transformations (XSLT) [W3C03b]. Although the specification defines a comprehensive set of functions some requirements cannot be met. With the use of external functions even these cases may be accomplished.

Implementing such external functions is not limited to a certain programming language. Through the use of Xalan-Java [AXJ07a] and the Bean Scripting Framework [BSF06a] scripts written in many scripting languages can be executed within the process of transformation.

This paper shows how such external functions can be used. Many businesses have sets of libraries containing the business rules which should not be reimplemented. By using these libraries, duplication of source code can be avoided.

The nutshells show that the access of web services or the data in the database is feasible. They also show that problems could arise depending on the scripting language used. If the database access is programmed using a scripting language which is not capable of caching the connection, processing times will degrade excessively.

Therefore the used scripting language should be chosen wisely. Using the wrong alternative could easily lead to a badly performing transformation. For example, it makes no sense to write a JRuby application from scratch to get access to the database. But if such an application already exists, using it is recommended.

Many other use cases could be accomplished this way like calling services implemented using Enterprise Java Beans [SUN07b] or .NET [MSN07]. Even calls to PHP [PHP07] and inclusion of binary content into the resulting XML is possible.

Building on the results of this bachelor thesis, future studies of the author will focus on the area of using Domain Specific Languages (DSL) [MFW07] within transformation processes. This could solve such requirements as calculating limits for offers. Using DSL expressions like “14.workdays.from.now” could be converted to a date on the nearest workday 14 days in the future. Such DSL could be made using Groovy [GRO07] or calls to the Google Data API [GDA07].

The paper shows that it is possible to reuse existing code even within a XSL Transformation.

References

[AHA05]: Andreas Ahammer, *OpenOffice.org Automation:Object Model, Scripting Languages,"Nutshell"-Examples*, Bachelor Thesis, Vienna University of Economics and Business Administration, Vienna, Austria, 2005

[AXI07]: Apache Software Foundation, *Axis 1.4 Homepage*, UrlDate(2007-04-23**), URL** http://ws.apache.org/axis/

[AXJ07a]: Apache Software Foundation, *Xalan-Java Homepage*, UrlDate(2007-05-01**), URL** http://xml.apache.org/xalan-j

[AXJ07b]: Apache Xalan-Java, *Download page*, UrlDate(2007-04-02**), URL** http://www.apache.org/dyn/closer.cgi/xml/xalan-j

[B4R07]: Prof. Rony G. Flatscher and students, *BSF4Rexx homepage*, UrlDate(2007-05-22**), URL** http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx

[BSF06a]: Apache Software Foundation, *Bean Scripting Framework Homepage*, UrlDate(2007-04-02**), URL** http://jakarta.apache.org/bsf

[BSF06s]: Apache Software Foundation, *Subversion Repository for BSF*, UrlDate(2007-05-02**), URL** http://svn.apache.org/repos/asf/jakarta/bsf

[BSF07b]: Apache Software Foundation, *BSF 3.0 binary distribution*, UrlDate(2007-05-16**), URL** http://people.apache.org/~antelder/bsf/3.0-beta1/

[ECM04]: Ecma International, *Standard ECMA-262 : ECMAScript Language Specification*, UrlDate(2007-04-02**), URL** http://www.ecma-international.org/publications/standards/Ecma-262.htm

[EXS07]: EXSLT Org, *Homepage*, UrlDate(2007-05-20**), URL** http://exslt.org

[FOP07]: Apache Software Foundation, *Formatting Objects Processor*, UrlDate(2007-05-27**), URL** http://xmlgraphics.apache.org/fop

[GDA07]: Google Inc., *Google Data API*, UrlDate(2007-05-27**), URL** http://code.google.com/apis/gdata/reference.html

[GRO07]: The Codehaus Opensource Software Community, *Groovy Script and DSL*, UrlDate(2007-05-27**), URL** http://groovy.codehaus.org/Writing+Domain-Specific+Languages

[JCP07]: Java Community Process, *Scripting for the Java Platform (Link to JSR-223)*, UrlDate(2007-05-02**), URL** http://www.jcp.org/en/jsr/detail?id=223

[JDI07]: Jensen Data Systems Inc., *Web Service Temperature Converter*, UrlDate(2007-05-18**), URL** http://www.xmethods.com/ve2/ViewListing.po?key=uuid:396577C1-EE97-6A65-AC0B-307B2C6943FA

[JDS07]: JudoScript.Com, *Homepage*, UrlDate(2007-05-02**), URL** http://www.judoscript.com/judo.html

[JRU07a]: JRuby Wiki, *Homepage*, UrlDate(2007-05-18**), URL** http://www.headius.com/jrubywiki/index.php/About_JRuby

[JRU07b]: The Codehaus Opensource Software Community, *JRuby Homepage*, UrlDate(2007-05-18**), URL** http://jruby.codehaus.org

[MFW07]: Martin Fowler, *Domain Specific Language*, UrlDate(2007-05-25**), URL** http://www.martinfowler.com/bliki/DomainSpecificLanguage.html

[MOZ06a]: Mozilla Org, *Rhino Homepage*, UrlDate(2007-04-02**), URL** http://www.mozilla.org/rhino/

[MOZ06b]: Mozilla Org, *Rhino Download Link*, UrlDate(2007-04-02**), URL** ftp://ftp.mozilla.org/pub/mozilla.org/js/rhino1_6R5.zip

[MSN07]: Microsoft Developers Network, *Dot Net Framework*, UrlDate(2007-05-27**), URL** http://msdn.microsoft.com/netframework

[MVN07]: Apache Software Foundation, *Maven Project Homepage*, UrlDate(2007-05-02**), URL** http://maven.apache.org/download.html

[OOO07]: OpenOffice.org, *Homepage*, UrlDate(2007-05-22**), URL** http://www.openoffice.org

[OOR07]: Rexx Language Association, *OORex homepage*, UrlDate(2007-05-22**), URL** http://www.oorexx.org

[PHP07]: The PHP Group, *PHP homepage*, UrlDate(2007-05-27**), URL** http://www.php.net

[POS07]: PostgreSQL Global Development Group, *Homepage*, UrlDate(2007-04-17**), URL** http://www.postgresql.org/

[ROR07]: Bill Walton and Curt Hibbs, *Ruby on Rails Tutorial*, UrlDate(2007-05-22**), URL** http://www.onlamp.com/pub/a/onlamp/2006/12/14/revisiting-ruby-on-rails-revisited.html

[RSQ07]: Mark Hessling, *Rexx/SQL homepage*, UrlDate(2007-05-22**), URL** http://rexxsql.sourceforge.net/index.html

[RWV07]: Sourceforge project, *Retroweaver Homepage*, UrlDate(2007-05-02**), URL** http://retroweaver.sourceforge.net

[SUN07a]: Sun Microsystems, Inc., *Java Standard Edition*, UrlDate(2007-05-02**), URL** http://java.sun.com/javase/

[SUN07b]: Sun Microsystems, Inc., *Java Enterprise Edition*, UrlDate(2007-05-27**), URL** http://java.sun.com/javaee

[W3C03a]: World Wide Web Consortium, *Homepage*, UrlDate(2007-04-02**), URL** http://www.w3.org

[W3C03b]: World Wide Web Consortium, *XSL Transformation (XSLT)*, UrlDate(2007-04-15**), URL** http://www.w3.org/TR/xslt

[W3C03c]: World Wide Web Consortium, *XSL Formatting Objects (XLS-FO)*, UrlDate(2007-05-15**), URL** http://www.w3.org/TR/xsl

[W3C03d]: World Wide Web Consortium, *XML Path Language (XPath)*, UrlDate(2007-05-22**), URL** http://www.w3.org/TR/xpath

[W3C03e]: World Wide Web Consortium, *XPointer Homepage*, UrlDate(2007-05-24**), URL** http://www.w3.org/TR/WD-xptr