Content syndication with the Project ROME Java API by using the open object scripting language ooRexx with BSF4ooRexx

-

Project Rome in combination with BSF4ooRexx

Martin Stoppacher

2010

Table of Contents

Table of figures

Table of code

Table of abbreviations

abbreviation	*description*
BSF	Bean Scripting Framework
ROME	Referring to Project ROME (Rss and atOM utiliEs)
BSF4ooREXX	Bean Scripting Framework for open Object Rexx
RSS	Really Simple Syndication (name of a web syndication format)
ATOM	Name of a web syndication format
Feed	Referring to any type of web feed
PR	Project Rome
JC	Java Classes
JDOM	Java library used to process XML data
DOM	Document object model (Java library used to process XML data)
SAX	Simple API fr XML (Java library used to process XML data)
DTD	Document type definition
DC	Dublin Core
IETF	Internet Engineering Task Force
RDF	Resource Description Framework
NS	Name Space
Oak	Abject application kernel
JRE	Java Runtime Environment
JDK	Java Development Kid
JSE	Java Standard Edition
JEE	Java Enterprise Edition
JME	Java Mobile Edition
API	Application Programming Interface
RSP	Rexx Server Pages
JSP	Java Server Pages
BSF4Rexx	Bean Scripting Framework for Rexx

1.0 Abstract

This text provides its reader a basic understanding and knowledge of concepts, usage and possibilities of the "Project ROME"(1) (Java) API (2). Furthermore the text provides a description and documentation about interfacing Java (and therefore the Project ROME API) with ooRexx (4) by using its extension BSF4ooRexx.
Specifically the text deals with the functionalities Project ROME is providing, as well as its technical concepts and relations. This functionalities get extended by using BSF4ooRexx and therefore demonstrate how to use an external Java API in open object Rexx.

As Project ROME deals with the processing of content syndication (3) the first part of the paper covers the explanation of concepts and basics related to content syndication.

The second part deals with the processing of content syndication via Project Rome by using the Java programming language. Project Rome will be described and a few example programs in Java will explain its functionalities.

The following parts cover the usage of ooRexx (4) with BSF4ooRexx (5). Therefore a short introduction to this topic will be given and afterwards the usage of Project Rome with BSF4ooRexx will be explained through examples.

The final part is built by a collection of several example programs which demonstrate the usage of Project Rome. First some Java examples are shown and second the corresponding ooRexx examples demonstrate how it is possible to use the same classes within ooRexx.

---

1)	Project ROME	Rss and aTOM utilitiEs	"set of open source Java tools for processing RSS and Atom feeds" [ROME09]
2)	API	- Application Programming Interface –	An Interface implemented by a software with the purpose to enables the interaction with other software
3)	Content Syndication		Distribution and sharing of content or information
4)	ooRexx	open Opject Rexx	Object oriented version of the Rexx scripting language
5)	BSF4ooRexx		Bean Scripting Framework for ooRexx

2.0 Introduction

“What is project ROME” this Question, taken from the website of Project ROME (https://rome.dev.java.net/), should be the first step on the way to a better understanding of what this Project deals with.

The first explanation can be found on the webpage of Project ROME: **"...ending syndication feed confusion by supporting all of 'em. " [ROME09]** This give us the first feeling about the content and purpose of Project ROME. Obviously we are talking about syndication feeds and the included confusion which take place within all its different types. (e.g. RSS 1.0, RSS 2.0, ATOM).

2.1 About this paper

2.1.1 Project ROME

The aim of PR (Project ROME) reflects the attempt to solve the syndication feed confusion problem by implementing a Java API (ROME API) which allows us to parse, generate, translate and publish syndication feeds in all of its different formats. Therefore this can be called a ubiquitous way in dealing with syndication and opens new perspectives and possibilities. ROME is using the JDOM XML Parser to create a general Java understandable version of a syndication feed. This is the point because due to this we are able to make translation between all the types possible. Now we know about Rome and that it translates, parses … syndication feeds, therefore let us take a short look at the API of Project ROME. On the left side you can see the projects packages. Obviously there are different packages one for the in and output and others for the processing of a syndication feed. We will discuss this in detail later, at this point it should be noticed that a short look at the documentation [ROME09] is recommended to understand the basics of the short examples given later. The Project ROME Packages are provided within a jar file and can simply be implemented by adding them to the Java resources.

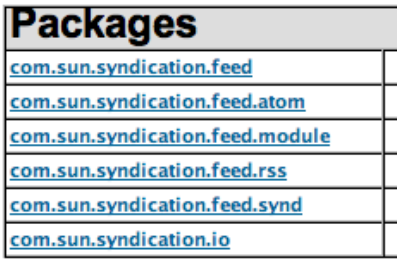

Figure 1) PR Packages **[ROME09]**

Why should we use Project ROME to process syndication feeds. First of all we have to face widely spread standards of Feeds, but why is this a problem, isn't it enough if everybody is using the standard he or she wants to use. Basically not, simply think about a connection between two webpages where one is using supplied feed information from the other but needs to provide an other standard. Therefore you simply need to translate the feeds. Or think about a feed collector that

stores several feeds in a text document. Due to all the different types you also need a translation into a general readable form. Further think about a company that supplies different types of standards, wouldn't it ease the way of dealing with the information if internally a general format is handled? Even more examples and possibilities can be thought, and the paths seam to be endless. Thus we have seen that it would be a big advantage to have a library which provides this features.

2.1.2 PR and BSF4ooRexx

Regarding this trends or prospectives we will focus on an other and very interesting way of extending *this given Java classes by using BSF4ooRexx. BSF a bean scripting framework for ooRexx (open objec*t Rexx) an enhancement of classic Rexx which is a "powerful, full-featured programming language which has a human-oriented syntax" [ooR09]. BSF4ooRexx targets on both, a usage of ooRexx within Java code and the implementation of Java classes in ooRexx scripts by taking advantage of bean scripting. BSF the Bean Scripting Framework is an open source software developed by IBM that allows Java programs to invoke programs written in other languages than Java. The following figure shows the Interfacing between (oo)Rexx and Java by using BSF4Rexx. This example is taken by intention because it shows the basic concept of interfacing Java via Rexx and vice versa. [Flat01] BSF4Rexx is written in C++ and is using the RexxEngine to invoke Rexx scripts out of Java and The RexxAndJava part to call Java via Rexx scripts.

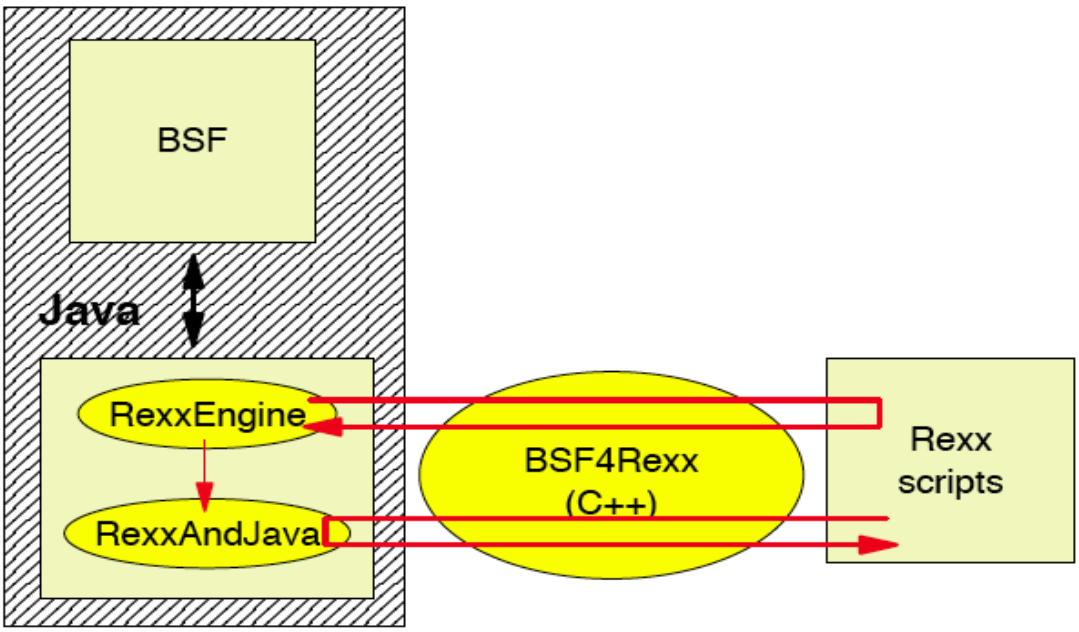

Figure 2) Interfacing (oo) Rexx with IBM's BSF Bean Scripting Framework [Flat01]

For further informations you can go to the BSF4ooRexx part of this paper or directly to [Flat01] and related papers named in the reference part to acquire a deeper understanding about Bean Scripting. In this paper we will focus on the newest version of BSF4ooRexx [BSF4ooR]. Regarding this version the basic concept has been extended with some features and new concepts mentioned in the BSF4ooRexx part of this paper and in several of the referenced papers. [Flat03] … [Flat09_b], [Aha05], [Burger05]

2.2 Research Question

How can we use the PR API with ooRexx via BSF4ooRexx? The Aim is to describe the related frameworks in short terms and provide essential references for the understanding and comprehension of those. Further we will create some basic example programs to enable the reader a rather fast entrance in the area of interfacing Java via Rexx and using PR to create, parse, translate … syndication feeds.

3.0 A short introduction into content syndication

Content Syndication describes a specific type of XML applications. At its beginning “Content Syndication” was used in a business to business context e.g. for editorial columns(1). As resources get scarce most distributors of information (newspapers …) licence its content by a vendor the “content syndicator”.
According to this content syndication deals with the distribution of website material. [wiki09_d] It refers to providing information e.g. summaries, recently added content … from a website.
As “content syndication” is strongly related to XML we want to refer to some resources as we are not able to cover all of the XML specifications in this paper. For further information about XML documents please research the following resources. [wiki09_a] [McLJEd03] [w3c09] [BMcL02]

According to [BMcL02] the Web changed content syndication in four dramatic ways:

1. thousands of millions of content providers [BMcL02]
2. eliminated entry barriers and enabled content providers to became there own syndicators [BMcL02]
3. thousands of new distribution outlets [BMcL02]
4. no geographical barriers and thus a strong competition [BMcL02]

Know we know about a short part of the power embedded in the syndication feed context but what can PR “Project Rome” [ROME09] offer us in this context? PR is a Java based library supporting all of the RSS and ATOM format standards. With PR we are able to parse generate and translate between feeds by using Java with the Rome API (*rome1.0.jar* current version 2009-12-22). As a Java developer you may need to parse, create … feeds to offer or retrieve some web content. This could be a single file within a homepage or the aggregation of several feeds. Regarding this we will explain the most common formats of syndication feeds in the next few pages before we start with PR and its API. Now we have seen some general information about the business context of content syndication and how powerful it can be but let us think a bit further what if Java is not the end. What if we are able to use the Java classes in other programming languages? E.g. we are able to use Java classes in the Rexx programming language by using it (Rexx) with BSF4Rexx [BSF4R] or in it's current and newest version ((oo) Rexx 4.0 [BSF4ooR]) called BSF4ooRexx.

---

1) Web feeds, on web pages, are in general linked with the word "Subscribe" or with the unofficial web feed logo . =>  Figure 3) common syndication feed logo [wiki09_d]

2) Some Hint: it is useful to use a XML creator for writing plane code e.g. (Link 29)

The two oo's imply that it can only be used with the newest version of ooRexx 4.0 [ooR09]. This is a big advantage in the usage of PR, as Rexx is an interpreted language PR becomes know a scriptable project and it gets very easy to use the JC (Java classes) in Rexx. The paper focus on the creation of such Rexx programs but first will will continue with the introduction to syndication feeds as there are RSS and ATOM.

3.1 Types of Syndication Feeds

There are many different types of syndication feeds e.g. there are nine different RSS variants. The PR supported types are: RSS 0.9, RSS 0.91, RSS 0.91, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, Atom 1.0

The next chapter (4) deals with the parsing, translating, creating … of this types by using PR. In this chapter we are explaining the principle structure of such feeds. As we are not able to deal with all of the different available types we will focus and explain some representatives (as there are RSS1.0, 2.0 and Atom 0.3).

3.1.1 RSS Syndication Format

RSS defines a *sequenced list of content* [BmcL02] called a channel. Within the *channel* there are *items* usually located with an URL. Additionally some metadata can be provided and the feed can specify an image. RSS was created by Netscape for the use on its MyNetscape portal. The trick was that the users had to at channels of content and so Netscape wanted to use a proper way to represent those channels.

Basically the different definitions can be divided into two types, those who are based on the Resource Description Framework (RDF) and those who are not.

Based on RDF	**Not Based on RDF**
RSS 0.9	RSS 0.91
RSS 1.0	RSS 0.92
	RSS 2.0

Figure 4) Dividing of RSS feeds [BmcL02]

The following examples will show this difference by using first the RSS 1.0 type and afterwards the RSS 2.0 type. Additionally you can try all types on your own by using the sample program from the appendix (prog. bsf9_b)

3.1.1.1 RSS 1.0 example:

This example was created with ooRexx using BSF4ooRexx and the PR API, you can find the full example code in the appendix of this paper (Appendix B; e.g. program 11)

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns="http://purl.org/rss/1.0/"  >
        <channel rdf:about="http://martinstoppacher.com">
                <title>This is a simple Sample_RSS1.0</title>
                <link>http://martinstoppacher.com/</link>
                <description>My favorite webpages; format:_rss_1.0
                </description>
                <!--example of how to add an image
                <image rdf:about=" --   >
                        <title> --- </title>
                        <url> --- </url>
                        <link> --- </link>
                </image>
                -->
                <items>
                        <rdf:Seq>
                            <rdf:li rdf:resource="https://rome.dev.java.net/" />
                            <rdf:li rdf:resource="http://java.sun.com/" />
                        </rdf:Seq>
                </items>
                <dc:creator>Martin Stoppacher</dc:creator>
                <dc:language>English</dc:language>
        </channel>
                <item rdf:about="https://rome.dev.java.net/">
                        <title>The Project Rome Web Page</title>
                        <link>https://rome.dev.java.net/</link>
                        <description>ROME Web Page</description>
                        <dc:creator>dev.java.net</dc:creator>
                        <dc:date>2010-01-02T23:00:00Z</dc:date>
                </item>
                <item rdf:about="http://java.sun.com/">
                        <title>Java 1.4 Documentation</title>
                        <link>http://java.sun.com/j2se/1.4.2/docs/api/</link>
                        <description>Documentation of the Java 1.4 API
                        </description>
                        <dc:creator>java.sun.com/</dc:creator>
                        <dc:date>2010-01-02T23:00:00Z</dc:date>
                </item>
</rdf:RDF>
```

code 1) RSS 1.0 example (created with PR [ROME09] using BSF4ooRexx [BSF4ooR])

It is rather easy to find the before mentioned channel and item objects within the example. In addition an image implementation example is given in the example code as well as several meta-data informations.

3.1.1.2 RSS 2.0 example:

This example was created with ooRexx using BSF4ooRexx and the PR API, you can find the full example code in the appendix of this paper (Appendix B; e.g. program 11)

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:dc="http://purl.org/dc/elements/1.1/" version="2.0">
        <channel>
                <title>This is a simple Sample</title>
                <link>http://martinstoppacher.com/</link>
                <description>My favorite webpages; format:_ rss_2.0
                </description>
                <language>English</language>
                <dc:creator>Martin Stoppacher</dc:creator>
                <dc:language>English</dc:language>
                <!-- example of how to add an image
                <image>
                        <title> --  </title>
                        <url> --- </url>
                        <link> --- </link>
                </image>
                -->
                <item>
                        <title>The Project Rome Web Page</title>
                        <link>https://rome.dev.java.net/</link>
                        <description>ROME Web Page</description>
                        <category domain="https://rome.dev.java.net/">Java
                        </category>
                        <pubDate>Sat, 02 Jan 2010 23:00:00 GMT</pubDate>
                        <guid isPermaLink="false">https://rome.dev.java.net/
                        </guid>
                        <dc:creator>dev.java.net</dc:creator>
                        <dc:date>2010-01-02T23:00:00Z</dc:date>
                </item>
                <item>
                        <title>Java 1.4 Documentation</title>
                        <link>http://java.sun.com/j2se/1.4.2/docs/api/</link>
                        <description>Documentation of the Java 1.4 API
                        </description>
                        <category
                            domain="http://java.sun.com/j2se/1.4.2/docs/api/">
                                Documentation
                        </category>
                        <pubDate>Sat, 02 Jan 2010 23:00:00 GMT</pubDate>
                        <guid isPermaLink="false">http://java.sun.com/</guid>
                        <dc:creator>java.sun.com/</dc:creator>
                        <dc:date>2010-01-02T23:00:00Z</dc:date>
                </item>
        </channel>
</rss>
```

code 2) RSS 2.0 example (created with PR [ROME09] using BSF4ooRexx [BSF4ooR])

Obviously the RSS 2.0 standard seams to be a light weight compared to RSS 1.0. The next point provides a short comparison of this two standards.

3.1.1.3 A short comparison

It is very obvious that, although the main objects stay the same, the syntactical difference is significant. The most obvious once are the root elements and the namespaces. Within RSS 1.0 there are two namespace definitions first the definition of the RDF namespace (xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#") and second the general definition for the RSS 1.0 feed (xmlns="http://purl.org/rss/1.0/"). Compared to RSS 2.0 there is one more NS definition than in the second example. This is due to the fact that the elements in RSS 2.0 are not using the RDF framework. The one definition found in the RSS 2.0 example is just for the DC elements (xmlns:dc="http://purl.org/dc/elements/1.1/" version="2.0") used within the RSS 2.0 Feed.

One further point is that in RSS 2.0 there is no separated list of resource identifiers within the channel elements because all the items are included in it.

An other interesting development is the inclusion of HTML markup within the syndication feed elements. This can be done in two ways: [BmcL02]

First: by XML-escaping the markup e.g. `<b>`
Secound: by using the CDATA block

```
<description>
<![CDATA[<i>More</i> news about the <b>Example</b> project]]>
</description>
```

3.1.2 ATOM Syndication Format

The ATOM syndication standard, compared to the RSS standard, provides a merge of the both mentioned RSS formats (RSS1.0 and 2.0). It takes the simplicity of the RSS 2.0 type, as it is also using no RDF (3.2) definition and the structured aspects from the RSS1.0 standard. [wiki09_c], [BmcL02]
The specification is being developed by the Atom Pub Working Group which is a part of the IETF (1).

1) Internet Engineering Task Force develops and promotes Internet standards

3.1.2.1 ATOM 0.3 example:

This example was created with ooRexx using BSF4ooRexx and the PR API, you can find the full example code in the appendix of this paper (Appendix B; e.g. program 11)

```
<?xml version="1.0" encoding="UTF-8"?>
<feed            xmlns="http://purl.org/atom/ns#"
                 xmlns:dc="http://purl.org/dc/elements/1.1/" version="0.3"     >
                 <title>This is a simple Sample</title>
                 <link rel="alternate" href="http://martinstoppacher.com" />
                 <tagline>My favorite webpages; format:_ atom_0.3</tagline>
                 <dc:creator>Martin Stoppacher</dc:creator>
                 <dc:language>English</dc:language>
        <entry>
                 <title>The Project Rome Web Page</title>
                 <link rel="alternate" href="https://rome.dev.java.net/" />
                 <author>
                         <name>dev.java.net</name>
                 </author>
                 <id>https://rome.dev.java.net/</id>
                 <modified>2010-01-02T23:00:00Z</modified>
                 <issued>2010-01-02T23:00:00Z</issued>
                 <summary type="text/plain" mode="escaped">ROME Web Page
                 </summary>
                 <dc:creator>dev.java.net</dc:creator>
                 <dc:date>2010-01-02T23:00:00Z</dc:date>
        </entry>
        <entry>
                 <title>Java 1.4 Documentation</title>
                 <link rel="alternate"
                             href="http://java.sun.com/j2se/1.4.2/docs/api/" />
                 <author>
                         <name>java.sun.com/</name>
                 </author>
                 <id>http://java.sun.com/</id>
                 <modified>2010-01-02T23:00:00Z</modified>
                 <issued>2010-01-02T23:00:00Z</issued>
                 <summary type="text/plain" mode="escaped">
                         Documentation of the Java 1.4 API
                 </summary>
                 <dc:creator>java.sun.com/</dc:creator>
                 <dc:date>2010-01-02T23:00:00Z</dc:date>
        </entry>
</feed>
```

code 3) ATOM 0.3 example (created with PR [ROME09] using BSF4ooRexx [BSF4ooR])

The first thing you can recognise in this example is the change of the object names into *feed* and *entry*. ATOM like RSS 2.0 is not using the RDF framework. The *Author* element is also a difference to the prior examples as well as *modified, issued or summary* … .

3.2 Document Type Definition and Modules

Like XML RSS feeds are extensible through DTD (1) called RSS modules. E.g. In RSS 1.0 the specification defines three modules: *Dublin Core*, *Syndication* and *Content*. Modules per definition are extended namespaces additional to the namespace of the host RSS document. Modules are frequently used within syndication feeds. You can see some samples above in the RSS and ATOM examples. A module is defined by adding a new namespace element (e.g. xmlns:dc) to the feed. Some modules are restricted to a specific type of feed but most are working with all types.

To explain one of the most common modules we will describe Dublin Core in short terms here. The main advantage of DC (and therefore DTD) is the ability to create the same terms within a variety of formats. Just take a look at the examples and it is obvious that all the DC elements are the same. Therefore DC is a metadata definition schema, more information can be found on http://purl.org/dc (Link 11)
DC defines 15 elements (http://purl.org/dc/elements) , as there are: title, creator, subject, description, publisher, contributor, date, type,format, identifier, source, language, relation, coverage and rights. This elements can also be used as a replacement of the RSS or ATOM elements.

An other example is the namespace extension for the itunes music store. Users need to implement additional elements to RSS 2.0 feeds to use the podcast directory to get listed within the store.

Namespace: http://www.itunes.com/dtds/podcast-1.0.dtd
Documentation: http://www.apple.com/itunes/podcast/techspecs.html

---

RDF Resource Description Framework: is a w3c specification; designed as a metadata model (Link 9);

DTD Document Type Definition: is a set of markup declarations therefore it is a kind of XML schema (Link 10)

4.0 Project ROME, Rss and atOM utilitiEs

This chapter covers content syndication processing vie Project Rome and its related technologies. It presumes a basic understanding about content syndication and XML. For Further information about XML and syndication feeds please research the following resources: [wiki09_a] [McLJEd03] [w3c09][BmcL02]

Based on the informations given in the examples before someone should be able to generate his/her own Feed either in the RSS format or the ATOM format. However, it is also possible to take advantage of one of the Java based XML APIs as there are SAX and DOM or like ROME itself does JDOM. With the newest version of BSF4ooRexx (version 4.0 [BSF4ooR]) there came a view examples about using the SAX and DOM APIs. Additionally in chapter 4 and 5 there are two examples included explaining the usage of SAX either with Java and ooRexx.
As it is not very convenient to use classes like *Document* and *Element* in a feed related context the usage of a more feed supporting library is recommended. The standard library in this context is PR [ROME09]. The API of PR is providing a more comfortable and feed based set of names like *Channel* or *Item.* *What is Procekt Rome?* =>
"ROME is a Java based open source RSS library supporting all of the RSS and ATOM format standards." [ROME09]. **The by default supported types are:**

• RSS 0.9	• RSS 0.94
• RSS 0.91 Netscape	• RSS 1.0
• RSS 0.91 Userland	• RSS 2.0
• RSS 0.92	• Atom 0.3
• RSS 0.93	• Atom 1.0

Figure 5) Table of PR default RSS and ATOM Feeds [ROME09]

Project ROME supports generating, parsing and translating between the different types or syndication feeds. ROME, in particular the ROME API, can be found at [ROME09]. Project ROME is currently available under version 1.0 and is delivered as a jar file which needs to be included into the Java resources or the classpath. ROME is based on JDom, used to convert the feeds into a general Java readable format. Jdom can be found at [JDOM09]. You also need to include "jdom.jar" into your Java resource folder. More information about jar files is provided within the next view points.

4.0.1 Java prerequisites

As mentioned above it is necessary to include the provided .jar files into the Java library. Due to this we will give a short introduction into Java and the related jar files.

4.0.1.1 Java

Java was developed at the beginning of the 90ies by Sun Microsystems. First called Oak (Object application kernel) the language was nearly bound to die but with the development of the internet and the upcoming need of platform independent languages Oak became an interesting language. The name Oak, due to some conflicts with name rights has been changed to Java. Java is a full object oriented language. The main difference to other languages like C++ or Pascal is the platform independent paradigm. After Java code is written it gets compiled by its compiler ("javac" command in a shell) and something called bytecode is produced out of the java code. E.g. if you compile the file "test.java" by typing in the command "javac test.java" the compiler produces a file called "test.class" which could be executed by every Java interpreter. The interpreter is the only platform depending tool someone need to run the Java bytecode. Nowadays almost every operating system comes with a version of JRE (Java Runtime Environment) and therefore java bytecode can be executed on almost every system. In the following graphic you can see a schema of how Java is working.

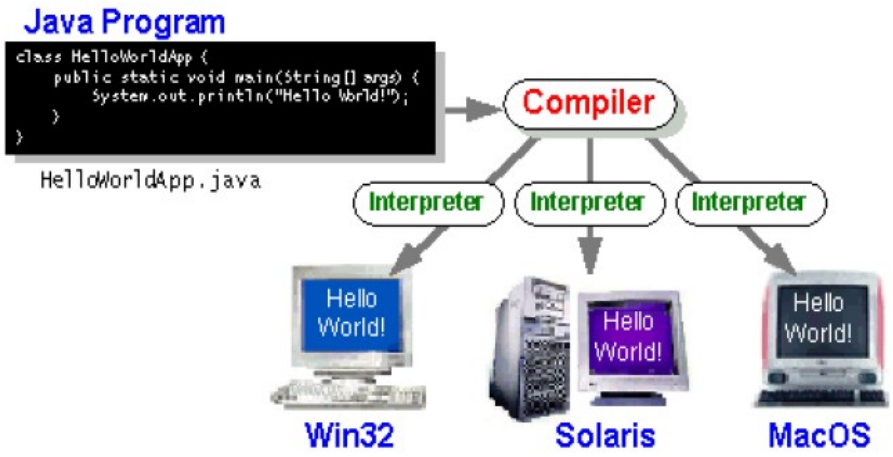

Figure 6) Schema of Java operations [Prem06]

Java is delivered in many different versions and can be downloaded under (Link 15). The different versions can be divided into different types:

Java SE	Java Standard Edition; This is the choice for standard users. If you want to develop programs in Java you should use the JDK which includes the necessary support or otherwise the JRE which enables just the running of Java programs.
Java EE	Java Enterprise Edition; This is the choice for companies which develop web-shops, … . This version includes a Java Application Server, that helps developing Java-based web services and much more.
Java ME	Java Mobile Edition; This version is designed for different parts of small devices. Including mobile phones, handheld devices and others.

Figure 7) Java version types [Prem06]

Java is a full object oriented language and also fully documented under (Link 16).In addition it is possible to download the Java documentation under (Link 15). The documentation consists of a full description of all classes that are delivered with the relating Java version. This classes can be included in the java code by creating import statement in the code like "import java.sql.*;", which includes all java.sql classes. We will see later how we can do this using BSF4ooRexx. The simplest Java program is the hello world program.

```
public class hello      {
        public static void main (String[] args) { System.out.println("Hello,
world!\n"); }
                        }
```

Code 4) Java Hello World

The next example is a program used to connect to a MySql database to retrieve some information from the database. This example is also used by intention because it is necessary to include an additional library i.e. the Java MySQL database connector (mysql-connector-java-5.1.10-bin.jar) available under (Link 16). The corresponding ooRexx example can be found in the Appendix C.

```
import java.sql.*;
public class mysql {
        public static void main(String[] args) {
        String uid = " /*MySQL UserID*/        ";
        String url = "jdbc:mysql://localhost/  /*database name*/  ";
        try {
                DriverManager.registerDriver(new com.mysql.jdbc.Driver());
/*
register Driver    */
                Connection conn =
                      DriverManager.getConnection(url, "root","/* Password*/");
                Statement stmt =
                        conn.createStatement ();     /* get the connection to
the DB       */
                ResultSet rset =
```

```
                stmt.executeQuery("select NR, NAME from mytest");
/*                                                      example querry     */
                System.out.println("NR_____NAME");
                while (rset.next()) {
                System.out.print(rset.getString("nr")); /* query some results */
                System.out.print("_____");
                System.out.println(rset.getString("NAME"));
                }
        }catch (SQLException e) {
                System.out.println(e);
        }
} }
```

Code 5) Java MySql connection; parts are taken from (Link 14)

Like MySQL, Project ROME provides an external library which needs to be included either in the Java library folder or by adding the location of the external library to the classpath variables. Under Windows you can do this within the system properties. For more information on how to include the classpath take a look at (Link 17).

4.0.1.2 A short excursus to Jar files

Regarding the PR API which is provided as Jar file we included this short excursus on Jar files. Jar files are packaged with the ZIP file format. Therefore it is usable as lossless data compression, archiving, decompression and archive unpacking (Link 13). Java adds some very convenient features regarding jar files to your system. E.g. you can use the following commands to interact with those files.

Operation	**Command**
To create a JAR file	jar cf jar-file input-file(s)
To view the contents of a JAR file	jar tf jar-file
To extract the contents of a JAR file	jar xf jar-file
To extract specific files from a JAR file	jar xf jar-file archived-file(s)
To run an application packaged as a JAR file (version 1.1)	jre -cp app.jar MainClass
To run an application packaged as a JAR file (version 1.2 -- requires Main-Class manifest header)	java -jar app.jar
To invoke an applet packaged as a JAR file	<applet code=AppletClassName.class archive="JarFileName.jar " width=width height=height > </applet>

Figure 8 List of commands regarding Jar files (Link 13)

More specific information is available under (Link 13). E.g. the command “jar tf rome-1.0.jar” invoke the output of the full content of the specified Jar file. In the following example we retrieved the content of the PR API that is provided within the rome-1-0-jar file. The output in the graphic is not the full content of the file just the beginning.

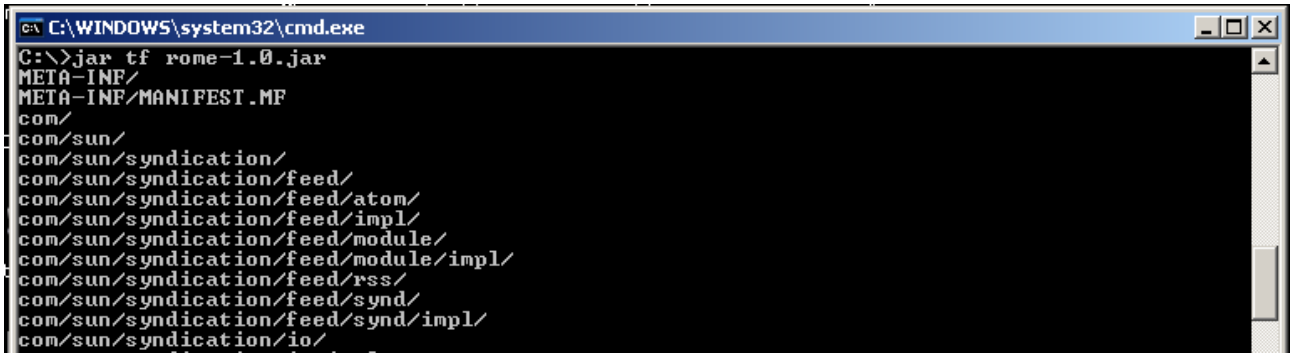

Figure 9 Output of the “jar tf rome-1.0.jar” command

4.0.2 PR data models

One of the most important things in PR is to think in data models. Rome includes three of them.:

Model 1)	The RSS data model (abstract)	(com.sun.syndication.feed.rss)
Model 2)	The Atom data model (abstract)	(com.sun.syndication.feed.atom)
Model 3)	A format independent data model	(com.sun.syndication.feed.synd)

Figure 10) Table of PR data models

These are the default models of PR. More specific information about the models and it's classes is provided under chapter 4.1 “The PR API” For each model PR is providing an implementation class called “*SyndFeedImpl*”. To create a “*SyndFeed*” object (A format independent data model) you can construct a new “*SyndFeedImpl*” object by parsing a Channel (RSS) or Feed (ATOM) through the “*SyndFeedImpl*” object constructor. For each of the models there are several features included to set, retrieve … the feed content but as shown in the previous chapter there are more namespaces included in the final XML feed presentation and therefore also more features implemented into PR. E.g. Prior we used DC to implement <dc:date></dc:date> into our feed examples, DC is one of the included PR modules. The mechanism to include more features is called modules There are several predefined extension modules listed at the PR wiki webpage (Link 12). Furthermore it is possible to create new modules on its own.

iTunes Podcasting	Apples extensions for listing in the iTunes podcast directory
GeoRSS	For adding location information to RSS/Atom
Slash	Used by Slash-based blogs
Google Base	For working with the Google Base types and custom tagsets
Content	For using content:encoded tags
Creative Commons	A unified module for working with the RDF and RSS/Atom Creative Commons License information
Yahoo! MediaRSS	For working with Yahoo! MediaRSS feeds (and Flickr Photostreams)
iPhoto Photocasting	For working with Apple iPhoto Photocasts
A9 OpenSearch	For working with Amazon/OpenSearch.org results
Microsoft Simple List Extensions	for sorting and grouping entries
Microsoft Simple Sharing Extensions	for synchronizing across bi-directional RSS and OPML feeds
Yahoo! Weather	For use with the Yahoo Weather API

Figure 11) List of PR Modules (2010-01-04) (Link 12)

This Modules came as .jar files as described before under 4.0.1 and need to be included into the classpath, if you want to use them. There are also methods included in the SyndFeed, SyndEntry, Feed, Channel, Entry and Item classes of PR (three in each) which relate to this modules.

Method 1	getModules()	Returns a list of module objects
Method 2	setModules(List list)	Replaces the current list of module obects
Method 3	getModules(String uri)	Returns the module object associated with a particular namespace URI

Figure 12) List of PR included methods relation PR modules [ROME09], [BMcL02]

Unfortunately a precise description would exceed this paper but for interested people there is a lot of information and examples at [BmcL02] and [ROME09]

4.0.3 The theory behind PR

On the PR wiki webpage a really nice article is published regarding the PR parsing process written by Dave Johnson and copied to the PR wiki. The next view pages are related to this article (Link 19) explaining the internal process of the PR library.

As mentioned before PR is based on data models i.e. an idealized one and abstract models of a syndication feed. ROME can parse any syndication feed into any of this formats. Internally PR creates a intermediate object model for specific news feed formats [ROME09] (Link 19) RSS or ATOM. This process of converting the feed into this intermediate model the "Wire Feed" is done by a JDOM based parser. For that reason it is necessary to include the Jdom library into the Java resources. The "Wire Feed" is parsed by Jdom but there is the idealized model too. The conversion in this case is done by PR itself. The next graphic describes what happens during the parsing.

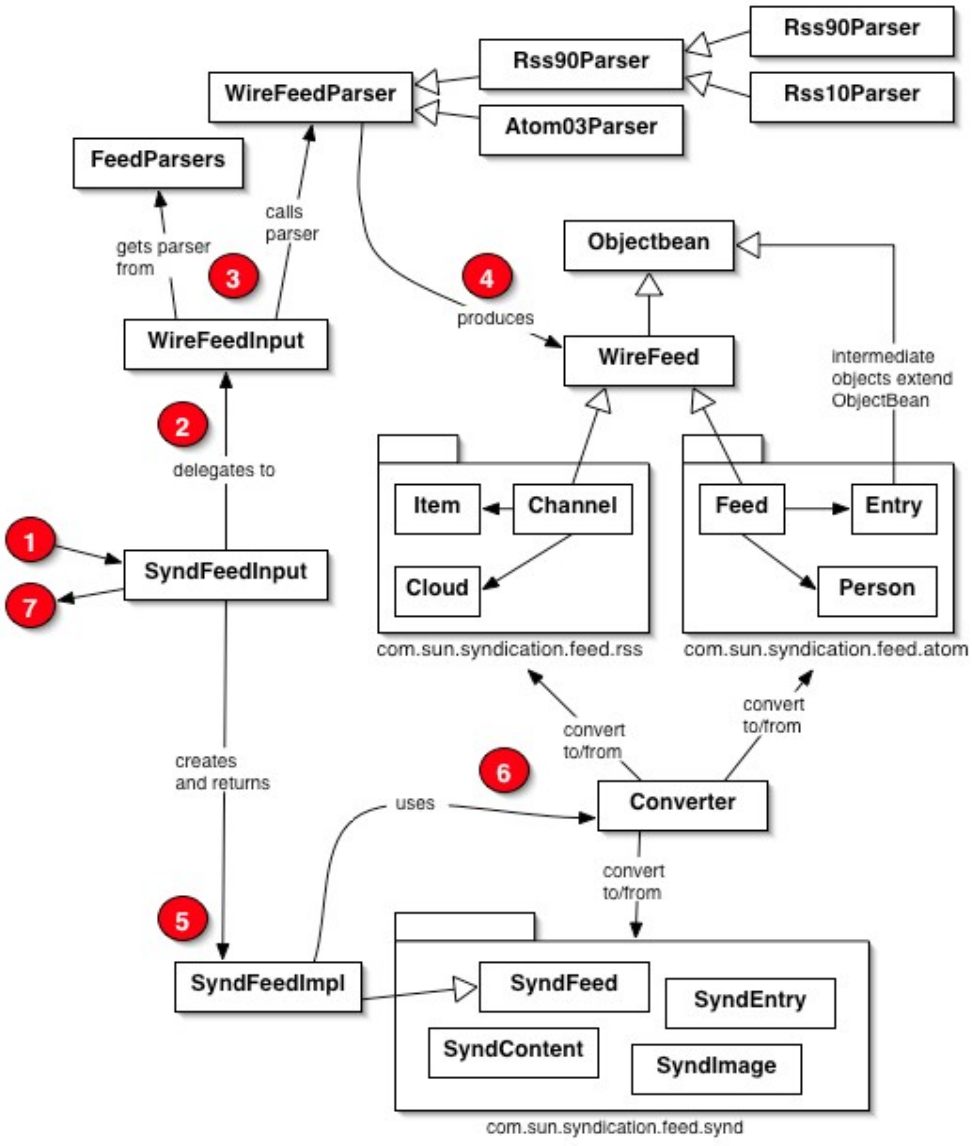

Figure 13) internal process of PR (Project ROME) [ROME09] (Link 19)

1. *SyndFeedInput* is called to parse a syndication feed. (com.sun.syndication.io.SyndFeedInput) The following three lines of code show the simplicity of doing syndication feed parsing. Only this three lines of code are necessary to parse a newsfeed.

```
URL feedUrl = new URL(args[0]);
        SyndFeedInput input = new SyndFeedInput();
        SyndFeed feed = input.build(new XmlReader(feedUrl));
```

Code 6) Creating a SyndFeedInput [Rome09]

2. After the *SyndFeedInput* is called it delegates to *WireFeedInput* to do the actual parsing. (com.sun.syndication.io.WireFeedInput)

3. The PluginManager of the *WireFeedInput* is of the class FeedParsers. The manager picks the right parser to parse the feed and calls the it for parsing the feed.

4. Now this parser parses the feed into the intermediate object model "Wire Feed" using Jdom. In the case of RSS the *WireFeed* is of class Channel, in the case of ATOM it is in the class Feed. Finally *WireFeedInput* returns a *WireFeed*.

5. The *SyndFeedInput* is using the returned *WireFeedInput* to create a *SyndFeedImpl* which implements *SyndFeed* which represents the format independent model.

6. Now we created a *WireFeed* out of the parsed syndication feed (4) and an empty *SyndFeed* (5). The class *SyndFeedImpl* now uses a converter to convert between the *WireFeed* and the *SyndFeed* or in other words between the abstract model and the independent model.

7. Finally *SyndFeedImpl* returns the parsed syndication feed within the *SyndFeed*.

All done!

4.0.4 A short excursus to SAX (Simple API for XML)

As it is also possible to use other opportunities to parse XML documents we peovide a short example of a SAX XML parser in Java. Syndication Feeds are represented through XML documents and therefore it is possible to use the SAX XML parser.

Several other libraries are available like SAX and DOM these are libraries delivered with Java but there are more available like JDOM which is used within PR as mentioned above. To achieve a deeper understanding of SAX just visit the webpage of the SAX project (Link 17). Nevertheless PR is the standard library for syndication feed processing and should therefore be used in such a context.

```
import java.io.FileReader;
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;


public class MySAXApp extends DefaultHandler
{
    public static void main (String args[])
        throws Exception
    {
        XMLReader xr = XMLReaderFactory.createXMLReader();
        MySAXApp handler = new MySAXApp();
        xr.setContentHandler(handler);
        xr.setErrorHandler(handler);
        for (int i = 0; i < args.length; i++) {

/*                            Parse each file provided on the command line.  */

            FileReader r = new FileReader(args[i]);
            xr.parse(new InputSource(r));
        }
    }
    public MySAXApp ()
    {
        super();
    }

/*                                                      Event Handler     */

    public void startDocument ()
        {       System.out.println("Start document");        }

    public void endDocument ()
        {       System.out.println("End document");  }

    public void startElement
                        (String uri, String name,String qName, Attributes atts)
    {   if ("".equals (uri))
                System.out.println("Start element: " + qName);
        else
```

```
                System.out.println("Start element: {" + uri + "}" + name);
    }


    public void endElement (String uri, String name, String qName)
    {   if ("".equals (uri))
                System.out.println("End element: " + qName);
        else
                System.out.println("End element:   {" + uri + "}" + name);
    }

    public void characters (char ch[], int start, int length)
    {
        System.out.print("Characters:    \"");
        for (int i = start; i < start + length; i++) {
            switch (ch[i]) {
            case '\\':
                System.out.print("\\\\");
                break;
            case '"':
                System.out.print("\\\"");
                break;
            case '\n':
                System.out.print("\\n");
                break;
            case '\r':
                System.out.print("\\r");
                break;
            case '\t':
                System.out.print("\\t");
                break;
            default:
                System.out.print(ch[i]);
                break;
            }
        }
        System.out.print("\"\n");
    }
}
```

Code 7) SAX Java code example, retrieved from (Link 17)

This example shows that it is also possible to achieve parsing of syndication feeds with the SAX library but it is very obvious that a lot of additional coding needs to be done. This is due to all the definitions of names that need to be done within the code to customise the code to syndication feed parsing.

4.1 The Project ROME API

As shown in the introduction the PR API is divided into different packages which enable to perform the before mentioned data models, modules, parsing, translating , and so on. The java documentation of PR can either be researched online [ROME09] or in its downloaded version. The downloadable version is provided as a Jar file and can be processes as shown under 4.0.1.2.

4.1.1 Overview

Before we spoke about the data models of PR. Now the models become real in form of the Java documentation. The three models are represented through the packages com.sun.syndication.feed.rss, com.sun.syndication.feed.atom and com.sun.syndication.feed.synd as shown in Figure 10. The next thing we spoke about were the PR modules which are used to imclude external resources like DC into PR. The package used for this is represented by com.sun.syndication.feed.module.

As there are 6 packages in PR 2 are missing the first is the com.sun.syndication.feed package. This package is just used for simple copy operations and therefore is useful for moving from one implementation of a bean interface to another.

The last package is an essential one. The com.sun.syndication.io package is used for inputting and outputting a syndication feed as an appropriate XML document. There are several classes and interfaces defined to parse and generate different input and output types. Under 4.0.3 we created a SyndFeedInput() from the com.sun.syndication.io package to parse a syndication feed. The SyndFeedInput() class returns a SyndFeed that is an independent object model.

These Packages include all classes to parse, create, translate syndication feeds in all and between all different formats. The following figure is an overview of the ROME API and shows the packages with there interfaces and classes and the description for them. As we are not able to provide a description of every detail of the PR API it is recommended to the reader to skim through the API and research it for a deeper detailed knowledge. However, provided in the Appendix C, there are three UML diagrams [BMcL02] summarizing the PR model packages. These diagrams are very helpful while programming (it is recommended to print them and us them while programming).

Project Rome Packages (current version: Java 1.0 API; 2010-01-07)

com.sun.syndication.feed

Interfaces	CopyFrom	
Class Summary	WireFeed	Parent class of the RSS (Channel) and Atom (Feed) feed beans.

com.sun.syndication.feed.atom

Class Summary	Category	Bean for category elements of Atom feeds.
	Content	Bean for content elements of Atom feeds.
	Entry	Bean for entry elements of Atom feeds.
	Feed	Bean for Atom feeds.
	Generator	Bean for the generator element of Atom feeds.
	Link	Bean for link elements of Atom feeds.
	Person	Bean for person elements of Atom feeds.

com.sun.syndication.feed.module

Interfaces	DCModule	Dublin Core Module.
	DCSubject	Subject of the Dublin Core ModuleImpl.
	Extendable	Objects that can have modules are Extendable.
	Module	Base class for modules describing Metadata of feeds.
	SyModule	Syndication ModuleImpl.
Class Summary	DCModuleImpl	Dublin Core ModuleImpl, default implementation.
	DCSubjectImpl	Subject of the Dublin Core ModuleImpl, default implementation.
	ModuleImpl	Base class for modules describing Metadata of feeds, default implementations.
	SyModuleImpl	Syndication ModuleImpl, default implementation.

com.sun.syndication.feed.rss

Class Summary	Category	Bean for categories of RSS feeds.
	Channel	Bean for RSS feeds.
	Cloud	Bean for clouds of RSS feeds.
	Content	Bean for item descriptions of RSS feeds.
	Description	Bean for item descriptions of RSS feeds.
	Enclosure	Bean for item enclosures of RSS feeds.
	Guid	Bean for item GUIDs of RSS feeds.
	Image	Bean for images of RSS feeds.
	Item	Bean for items of RSS feeds.
	Source	Bean for item sources of RSS feeds.
	TextInput	Bean for text input of RSS feeds.

com.sun.syndication.feed.synd

Interfaces	Converter	Interface that defines the functionality to convert a SyndFeedImpl to a real feed (RSS or Atom) and vice versa.
	SyndCategory	Bean interface for categories of SyndFeedImpl feeds and entries.
	SyndContent	Bean interface for content of SyndFeedImpl entries.
	SyndEnclosure	
	SyndEntry	Bean interface for entries of SyndFeedImpl feeds.
	SyndFeed	Bean interface for all types of feeds.
	SyndImage	Bean interface for images of SyndFeedImpl feeds.
	SyndLink	Represents a link or enclosure associated with entry.
	SyndPerson	Bean interface for authors and contributors of SyndFeedImpl feeds and entries.
Class Summary	SyndCategoryImpl	Bean for categories of SyndFeedImpl feeds and entries.
	SyndContentImpl	Bean for content of SyndFeedImpl entries.
	SyndEnclosureImpl	
	SyndEntryImpl	Bean for entries of SyndFeedImpl feeds.
	SyndFeedImpl	Bean for all types of feeds.
	SyndImageImpl	Bean for images of SyndFeedImpl feeds.
	SyndLinkImpl	Represents a link or an enclosure.
	SyndPersonImpl	Bean for authors and contributors of SyndFeedImpl feeds and entries.

com.sun.syndication.io

Interfaces	DelegatingModuleGenerator	Adds the ability to give a direct wire feed generator reference to plugin modules that need to call back to their wire feed to complete generation of their particular namespace.
	DelegatingModuleParser	Adds the ability to give a direct wire feed reference to plugin modules that need to call back to their wire feed to complete parsing of their particular namespace.
	ModuleGenerator	Injects module metadata into a XML node (JDOM element).
	ModuleParser	Parses module metadata from a XML node (JDOM element).
	WireFeedGenerator	Generates an XML document (JDOM) out of a feed for a specific real feed type.
	WireFeedParser	Parses an XML document (JDOM) into a feed bean.
Class Summary	SAXBuilder	

	SyndFeedInput	Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C DOM Document or JDom DOcument) into an SyndFeedImpl.
	SyndFeedOutput	Generates an XML document (String, File, OutputStream, Writer, W3C DOM document or JDOM document) out of an SyndFeedImpl..
	WireFeedInput	Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).
	WireFeedOutput	Generates an XML document (String, File, OutputStream, Writer, W3C DOM document or JDOM document) out of an WireFeed (RSS/Atom).
	XmlReader	Character stream that handles (or at least attemtps to) all the necessary Voodo to figure out the charset encoding of the XML document within the stream.
Exception Summary		
	FeedException	Exception thrown by WireFeedInput, WireFeedOutput, WireFeedParser and WireFeedGenerator instances if they can not parse or generate a feed.
	ParsingFeedException	Exception thrown by WireFeedInput instance if it can not parse a feed.
	XmlReaderException	The XmlReaderException is thrown by the XmlReader constructors if the charset encoding can not be determined according to the XML 1.0 specification and RFC 3023.

Figure 14) Project Rome Packages (current version: Java 1.0 API; 2010-01-07) [ROME09]

4.1.2 Basic functions

All this packages include a lot of classes providing methods for all the different formats of syndication feeds. As mentioned, the packages are representing the data models within them a superset of features (e.g. in the independent model its a superset of RSS and Atom features) is placed (Take a look at Appendix C for the UML Diagrams of the data models).

Before explaining some of the model functions we will focus on one of the essential packages, the com.sun.syndication.io package, providing all the necessary classes for inputting and outputting to and from the models. We will first explain the functions of PR in short terms and then proceed with some Java coded examples.

4.1.2.1 Inputting

As syndication feeds are representing XML data any XML parser can be used to parse a feed either RSS or ATOM. Before we saw an example of a SAX parser but due to the high amount of RSS types it is useful to use a specialised library.
To input a feed we have to use the com.sun.syndication.io.* package. The classes *WireFeedInput* and *SyndFeedInput* can input feeds from different sources. The process of parsing a feed is described under 4.0.3. *WireFeedInput* is determining the type of the retrieved feed by checking its content (attributes, namespaces, elements). *SyndFeedInput* delegates the parsing to *WireFeedInput* and converts the resulting *WireFeed* to a *SyndFeed*.
Input sources for *WireFeedInput* and *SyndFeedInput*:

- java.io.Reader
- java.io.File
- org.xml.sax.InputSource
- org.w3c.dom.Document
- org.jdom.Document

4.1.2.2 Outputting

Like the input classes the classes *WireFeedOutput* and *SyndFeedOutput* are included in the com.sun.syndication.io.* package and provide the main outputting of syndication feeds. Both include methods to output to different objects.

- java.lang.String
- java.io.File
- java.io.Writer
- org.w3c.dom.Document
- org.jdom.Document

The most important thing is that the *SyndFeedOutput* can not be used without using the .setFeedType() method. If this is not used it is necessary to create a *WireFeed* and use *WireFeedOutput* instead.

4.1.2.3 Creating a Feed

The last point, after inputting and outputting, is the creation of syndication feeds. As there are different data models there are also different ways to create a feed. However, it is a good choice to use the independent data model to create it, regarding the prior mentioned reasons (Take a look at Appendix C for the UML diagram). To create a syndication feed by using the independent model we have to use its representation package the com.sun.syndication.feed.synd.* . E.g. To create a feed we can use the following classes.

- SyndContent()
- SyndContentImpl()
- SyndEntry()
- SyndEntryImpl()
- SyndFeed()
- SyndFeedImpl()

The top level of the feed is created by the SyndContendImpl() which creates a *SyndFeed*. With the included methods we are able to set the values of the feed (Title, Author, …). For in- and output of the created feed we can use the described com.sun.syndication.io.* package with its classes. However, as mentioned before we have to use the .setFeedType() method if we are using the independent data model (com.sun.syndication.feed.synd.*).

4.2 Using Project ROME (Java examples)

The following pages include some example snippets of Java code using PR to parse, create, translate … syndication feeds. The full code of this snippets can be found in the Appendix A. We will describe the functionality of the different snippets in short and link to the full code in the Appendix and to the PR API.

To easy the access to the related resources we add some information in a short table above the code. The fist column is just the name / number of the code snippet related to the number of the example in the table of content and to the corresponding program number in the Appendix. The second is the name of the related file in the Appendix. Beneath the code you can find the number of the code snippet as it is named in the "table of code".

name	*related program in the Appendix A*

4.2.1 Java example 1 - Feed Reader

In this first example you can see a simple feed parser or feed reader. This code is used to parse a feed from a defined source as described under 4.0.3. First we create a java.net.URL to use it with the com.sun.syndication.io.XmlReader to create a source for the build method of the *SyndFeedInput*. Finally the created *SyndFeed* is printed to the system with System.out.println(feed). However, this is just the java System.out.println statement and is therefore just printing the pure XML content of the retrieved source. This is the same as the method .tostring() within the *SyndFeed* class.

Java example 1	FeedReader.java

```
/*   **********       Java example code snippet; code(8)      **********      */
 URL feedUrl = new URL(args[0]);            /*  creates a string with the URL */
        SyndFeedInput input = new SyndFeedInput();        /* new input object */
        SyndFeed feed = input.build(new XmlReader(feedUrl));
/*                                 reads feed from URL and puts it into feed   */
        System.out.println(feed);             /*   outputs file to system     */
/*      ***************************************************************      */
```

code 8) Basic construction of a FeedReader (Java)

The next figure shows the corresponding output of the full java code example in the appendix which is using this snippet. The URI is delivered to the program as an argument, as defined above (arg[0]). After calling the *FeedReader* class with the java interpreter the program starts printing to the screen. As you can see it is printing the pure content of the *SyndFeed*.

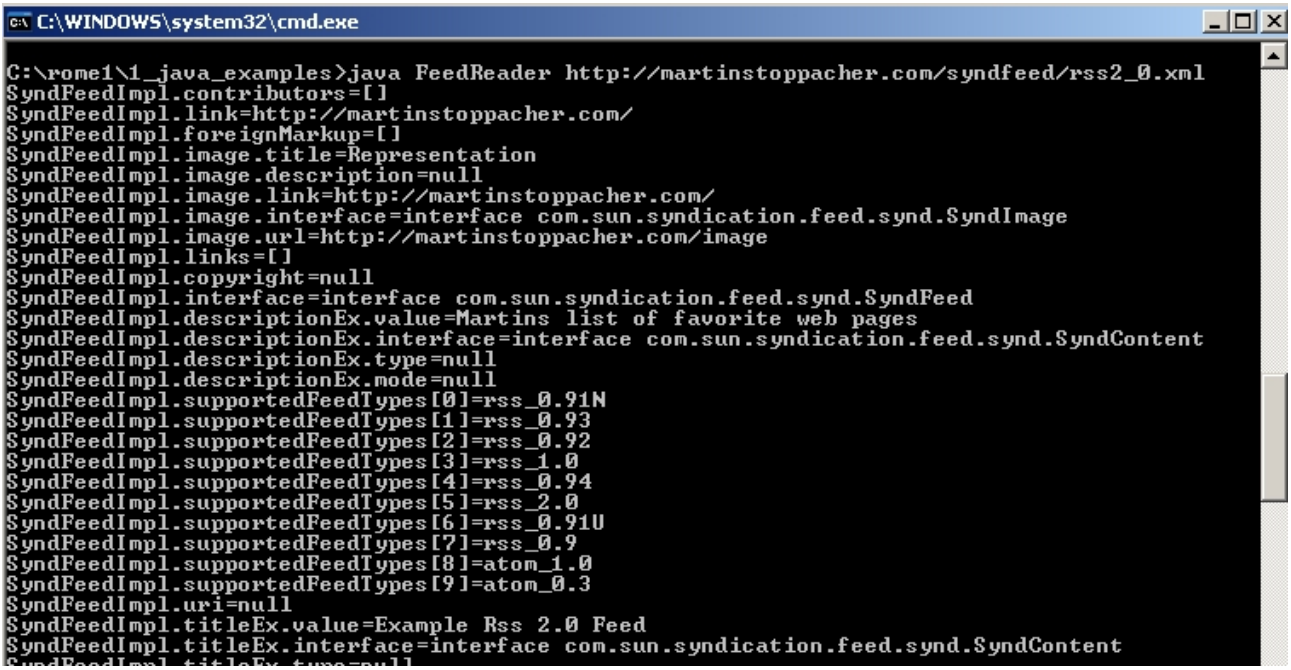

Figure 15) Output example for code 8 / Java Example 1

4.2.2 Java Example 2 - Feed Writer

The second example adds the ability to save the retrieved file to the system. First a new java.io.FileWriter is built for the use in the output method of the com.sun.syndication.io.SyndFeedOutput. After writing the writer gets closed and a message is printed to inform about the written file.

Java example 2	FeedReader_with_writer.java

```
/*   **********       Java example code snippet; code(9)      **********      */

                Writer writer = new FileWriter(fileName);
/*                          a new writer Object with a specified file name */
                SyndFeedOutput output = new SyndFeedOutput();
/*                                                        new output Object */
                output.output(feed,writer);   /* outputs the feed to the file */
                writer.close();

                System.out.println("The feed has been written"
                                   +"to the file ["+fileName+"]");
/*      ***************************************************************      */
```

code 9) FeedReader example with a writer function (Java)

In the figure you can see the simple output of the System.out.println statement. The file is saved in the same directory as the program.

```
C:\rome1\1_java_examples>java FeedReader2 http://martinstoppacher.com/syndfeed/rss2_0
The feed has been written to the file [test]
```

Figure 16) Output example for code 9 / Java Example 2

4.2.3 Java Example 3 - Feed Reader with a system output

The next example deals with two things. First we can see that the .set.FeedType method is used to determine the output of the *SyndFeed* and second that there is a *SyndFeedOutput* to create a XML tree representation on the screen. The retrieving of the feed works as described after using the .setFeedType method to set the type of the feed to a chosen one. The default values PR is using are rss_0.9, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0 & atom_0.3. The output method of the *SyndFeedOutput* finally prints the feed by using java.io.FileWriter. In this case it would also be possible to use the .outputString(feed,new PrintWriter()) without the System.out .

Java example 3	FeedReader_with_output.java

```
/*   *********        Java example code snippet; code(10)       *********      */
URL feedUrl = new URL(feedType);

        SyndFeedInput input = new SyndFeedInput();
        SyndFeed feed = input.build(new XmlReader(feedUrl));
        feed.setFeedType(outputType);
/*                    feed converter, set the type to the chosen outputType */

                SyndFeedOutput output = new SyndFeedOutput();
/*                                                  Tree Output of the feed */
                output.output(feed,new PrintWriter(System.out));
/*                                                          to the system */
/*      ************************************************************      */
```

code 10) FeedReader example with an output (Java)

```
C:\WINDOWS\system32\cmd.exe

C:\rome1\1_java_examples>java FeedReader2 http://martinstoppacher.com/syndfeed/rss2_0
The feed has been written to the file [test]

C:\rome1\1_java_examples>java FeedReader3 http://martinstoppacher.com/syndfeed/rss2_0
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:dc="http://purl.org/dc/elements/1.1/" version="2.0">
  <channel>
    <title>fm4.ORF.at</title>
    <link>http://fm4.orf.at</link>
    <description>fm4.ORF.at: Berichte und Kommentare zu Musik, Popkultur, Film, Literatur, G
ames und Politik.</description>
    <language>de</language>
    <copyright>Copyright ® 2010 ORF Online und Teletext GmbH &amp;amp; Co KG</copyright>
    <pubDate>Mon, 11 Jan 2010 18:30:03 GMT</pubDate>
    <dc:creator>ORF Online und Teletext GmbH &amp;amp; Co KG</dc:creator>
    <dc:publisher>ORF österreichischer Rundfunk, Wien</dc:publisher>
    <dc:date>2010-01-11T18:30:03Z</dc:date>
    <dc:language>de</dc:language>
    <dc:rights>Copyright ® 2010 ORF Online und Teletext GmbH &amp;amp; Co KG</dc:rights>
    <image>
      <title>ORF.at</title>
      <url>http://rss.orf.at/images/orf.at-small.gif</url>
      <link>http://rss.orf.at</link>
    </image>
    <item>
      <title>Afrika-Cup-Log, 4. Favoriten-Peinlichkeiten.</title>
      <link>http://fm4.orf.at/stories/1636203/</link>
      <description>Wie sich der Zwerg Malawi gegen den algerischen Riesen erheben konnte; wi
e der ivoirische Koloss sich selber ein Bein stellte.</description>
      <pubDate>Mon, 11 Jan 2010 18:26:07 GMT</pubDate>
      <guid isPermaLink="false">http://fm4.orf.at/stories/1636203/</guid>
      <dc:subject />
      <dc:date>2010-01-11T18:26:07Z</dc:date>
    </item>
    <item>
      <title>S³ditalien û Ausschreitungen gegen AfrikanerInnen</title>
      <link>http://fm4.orf.at/stories/1636206/</link>
```

Figure 17) Output example for code 10 / Java Example 3

The figure shows the output of a retrieved feed. The URI is provided to the program by an argument.

4.2.4 Java Example 4 - Different outputs

The 4th example is about the usage of different *SyndFeedInputs*. As we are creating instances of java classes we are able to create more than one. In this example the .setFeedType method is used to set the type of one *SyndFeedInput* to a specific type whereas the other one retrieves the feed with the original type.

Java example 4	FeedReader_combination.java

```
/*   *********        Java example code snippet; code(11)      *********      */
        SyndFeedInput input = new SyndFeedInput();
/*                                                          new input object */
        SyndFeed feed = input.build(new XmlReader(feedUrl));
/*                              reads feed from URL and puts it into feed */
        feed.setFeedType(outputType);

        SyndFeedInput input2 = new SyndFeedInput();       /* new input object */
        SyndFeed feed2 = input2.build(new XmlReader(feedUrl));
/*      ************************************************************      */
```

code 11) Setting different output types (SyndFeed) (Java)

4.2.5 Java Example 5 - Setting the feed type

This short snippet shows how to use the .setFeedType() method. This was already used within the examples before.

Java example 5	FeedConverter.java

```
/*   *********        Java example code snippet; code(12)      *********      */
        feed.setFeedType(outputType);
/*                        this line converts the feed into a specific format */
/*      ************************************************************      */
```

code 12) Set the output type (Java)

4.2.6 Java Example 6 - Feed aggregator

The next snippet shows how to aggregate some feeds to one single. This is done by using some of the get methods within the *SyndFeedImpl* and by adding them to a pre created feed. First we create a new feed by using the *SyndFeedImpl* class. Don't forget that we have to set a feed type in the case of creating a feed with the independent data model. After creating the feed a loop is started which goes through all the provided URIs one by one and retrieves there entries and add them to the pre created feed.

Java example 6	FeedAggregator.java

```
/*   *********        Java example code snippet; code(13)      *********      */

        SyndFeed feed = new SyndFeedImpl();
/*                                        creates a new syndfeed object */
        feed.setFeedType(outputType);         /* specifies which type of feed */

        feed.setTitle("Aggregated Feed");        /* specification of the feed */
        feed.setDescription("Anonymous Aggregated Feed");          /*   ***   */
        feed.setAuthor("Martin Stoppacher");                       /*   ***   */
        feed.setLink("http://martinstoppacher.com");               /*   ***   */

        List entries = new ArrayList();           /* Creates an enties Object */
        feed.setEntries(entries);  /* connects the entries Object to the feed */

                for (int i=1;i<args.length;i++) { /* loop to insert all feeds */
                URL inputUrl = new URL(args[i]);

                SyndFeedInput input = new SyndFeedInput();
                SyndFeed inFeed = input.build(new XmlReader(inputUrl));

                entries.addAll(inFeed.getEntries());
/*                add the entries of the fetched "infeed" to the created feed */
        }
        SyndFeedOutput output = new SyndFeedOutput();    /* new output object */
        Output.output(feed,new PrintWriter(System.out));
/*                                                       output to the system */
/*      ************************************************************      */
```

code 13) Aggregate a feed (Java)

The URIs can by provided in different way, either by using arguments or by using a grafical interface. Some examples of GUI are provided in the BSF4ooRexx part of this paper.

4.2.7 Java Example 7 - Using a loop with the aggregator

This example is an extension to the previous one. It shows the possibility of retrieving arguments and aggregating them to a String by using a “for” loop.

Java example 7	FeedAggregator2.java

```
/*   ********        Java example code snippet; code(14)      **********      */
        String sep = "__";
        for (int i=2;i<args.length;i++) {
        sources = sources + sep + args[i];
        }
                feed.setTitle("Aggregated_Feed_with the type_"+outputType);
/*      ************************************************************      */
```

code 14) Using the arguments within a feed (Java)

4.2.8 Java Example 8 - Feed Creator

This example provides an introduction into feed creation. As suggested under 4.1.2.3 we are using the independent model. First the *SyndFeedImpl* creates a new *SyndFeed*. Then we use the methods to set the values or this feed. After creating the main parts we start to create our entries by creating an instance of *SyndEntryImpl* and using its methods to add all the necessary entries to it. After finishing the entries object is added to a prior created array. Each single entry is added to this array and in the end the array gets added to the *SyndFeed*. After the creation of the *SyndFeed* it can be processed as described by outputting it or saving it to a directory, it is also possible to create a *WireFeed* out of the *SyndFeed* and than print it via *WireFeedOutput*. This is the only possible way if we would not be using .setFeedType method.

Java example 8	FeedCreator.java

```
/*   *********        Java example code snippet; code(15)      *********      */
        SyndFeed feed = new SyndFeedImpl();              /*   new feed object */
        feed.setFeedType(feedType);           /* define the type of the feed */

        feed.setTitle("This is a simple Sample");
        feed.setLink("martinstoppacher.com");
        feed.setDescription("My favorite webpages; format:_"+feedType);

        List entries = new ArrayList();              /* entries array object */
        SyndEntry entry;
        SyndContent description;

        entry = new SyndEntryImpl();               /* an other entry opbject */
        entry.setTitle("The ooRexx Web Page");
        entry.setLink("http://www.oorexx.org/");
```

```
        entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
        description = new SyndContentImpl();
        description.setType("text/html");
        description.setValue("<p>Open Object Rexx version 4.0.0”
                             +”is now available."
                +" Visit the announcement page for More information."
                +" </p><p>For details check the <a
href=\"http://www.oorexx.org/\">ooRexx"
                +" Web Page</a></p>");
        entry.setDescription(description);
// add the description object to the entry object
        entries.add(entry);     /* adds the entry object to the entries array */

        feed.setEntries(entries);     // add the entries array to the  feed

        SyndFeedOutput output = new SyndFeedOutput();
// simple output to the system
        output.output(feed,new PrintWriter(System.out));

        /*                                          // optional writer part
        Writer writer = new FileWriter(fileName);
        SyndFeedOutput output = new SyndFeedOutput();
        output.output(feed,writer);
        writer.close();

        System.out.println("The feed has been written to”
                           +”the file ["+fileName+"]");
        */
/*      ****************************************************************     */
```

code 15) Creating a feed (Java)

The next figure presents the output of the full example in the Appendix. The feed type is set through the argument and the created feed is printed to the system.

```
C:\rome1\1_java_examples>java FeedWriter rss_1.0
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns="http://purl.org/rss/
1.0/" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel rdf:about="martinstoppacher.com">
    <title>This is a simple Sample</title>
    <link>martinstoppacher.com</link>
    <description>My favorite webpages; format:_rss_1.0</description>
    <items>
      <rdf:Seq>
        <rdf:li />
        <rdf:li />
        <rdf:li />
      </rdf:Seq>
    </items>
  </channel>
  <item rdf:about="https://rome.dev.java.net/">
    <title>The Project Rome Web Page</title>
    <link>https://rome.dev.java.net/</link>
    <description>ROME Web Page</description>
    <dc:date>2009-12-25T23:00:00Z</dc:date>
  </item>
  <item rdf:about="http://java.sun.com/j2se/1.4.2/docs/api/">
    <title>Java 1.4 Documentation</title>
    <link>http://java.sun.com/j2se/1.4.2/docs/api/</link>
    <description>Documentation of the Java 1.4 API</description>
    <dc:date>2009-12-25T23:00:00Z</dc:date>
  </item>
  <item rdf:about="http://www.oorexx.org/">
    <title>The ooRexx Web Page</title>
    <link>http://www.oorexx.org/</link>
    <description>&lt;p&gt;Open Object Rexx version 4.0.0 is now available. Visit the announc
ement page for More information.&lt;/p&gt;&lt;p&gt;For details check the &lt;a href="http://
www.oorexx.org/"&gt;ooRexx Web Page&lt;/a&gt;&lt;/p&gt;</description>
    <dc:date>2009-12-25T23:00:00Z</dc:date>
  </item>
</rdf:RDF>
```

Figure 18) Output example for code 15 / Java Example 8

4.2.9 Java Example 9 - Using the aggregation with a Feed Creator

The final example adds some information to the previous example. It shows how to add an external resource to the created feed. This is working like a primitive feed Aggregator from example 6.

Java example 9	FeedWriter_with_Aggregator.java

```
/*   *********        Java example code snippet; code(16)     **********      */
        feed.setEntries(entries);       /* add the entries array to the  feed */

        for (int i=1;i<args.length;i++) {         /* loop to insert all feeds */
                URL inputUrl = new URL(args[i]);

                SyndFeedInput input = new SyndFeedInput();
                SyndFeed inFeed = input.build(new XmlReader(inputUrl));

                entries.addAll(inFeed.getEntries());
//                add the entries of the fetched "infeed" to the created feed */
        }
/*      ****************************************************************     */
```

code 16) Add aggregation to a feed creation (Java)

5.0 An Introduction into ooRexx and BSF4ooRexx

Starting with classic Rexx: “Rexx is a procedural language that allows programs and algorithms to be writen in a clear and structured way” [CoSh90]. The classic Rexx language covers different application areas like personal programming, tailoring user commands, macros and prototype development that traditionally have been served by different types of programming languages. Within a few years Rexx expanded its functionalities in massive ways. Due to a huge user community much extension modules are available and can be used to specialise, modify and expand Rexx in different ways and to use it in many business areas. There are extensions for interfacing with databases, graphical user interfaces, web programming interfaces, interfaces for XML processing and many more. The mentioned examples are very interesting in a business related context e.g. we can use “modrexx” which is the rexx interface for Apache web server to create our own Rexx Server Pages which means to use Rexx for web server programming. An other possibility is the interfacing with a relational database which is enabled by the Rexx/Sql interface. We will see that with the usage of BSF4ooRexx we are able to use an other possibility to connect to a database by using the Java connector. Rexx itself is able to integrate XML processing via the interface RexxXML. It is therefore possible to use Rexx for parsing syndication feeds as syndication feeds are valid XML code but like mentioned before while talking about Java's SAX it's not the convenient way of processing syndication feeds as there is no customisation. Therefore it would be really convenient to be able to use the PR API within Rexx.

The Rexx language faced a development of its interpreters and several different versions for different customised and specified application areas have been released. The most widely used interpreters are: Regina, Rexx/imc, Brexx, Reginald …

All these different versions came with different features and possibilities. Finally there is also an object oriented version of Rexx called ooRexx (open object Rexx). In this paper we are using open object Rexx regarding the point that it is a object oriented language like Java. Furthermore the newest version of BSF4ooRexx can only be used with ooRexx 4.0 .

“Standard (Classic) Rexx is a procedural language. In contrast, Open Object Rexx is a fully object-oriented superset of standard Rexx” [Fos05]. One main advantage of ooRexx is the portability of procedural Crexx scripts that run without changes in ooRexx. Further ooRexx emphasizes a simple syntax and therefore provides an easy entry into the object oriented paradigm.

As it is impossible to cover all the different features of classic and ooRexx in this paper, we recommend to use the following references for a deeper understanding of Rexx and all its related extensions and developments. [CoSh90], [Fos05], [RexxLA09], [ooR09]

5.1 About Open Object (oo) Rexx

IBM released ooRexx under the Common Public Licence in October 2004, after that the first version was delivered in February 2005 [Prem06]. As a result of the first independent Rexx symposium in 1990 the Rexx Language Association was founded. The main goal of the foundation is to promote Rexx and all its variants.

5.1.1 Classic Rexx features (also valid with ooRexx)

As ooRexx is fully compatible with Classic Rexx i.e. Classic Rexx scripts can also run under an ooRexx environment. We take a short focus on these basic concepts and functions and then proceed with the description of ooRexx.
The development of a programming language is a difficult thing and can be affected in many different ways and sometimes lead to unwanted results and ambiguities. According to [CoSh90] several concepts and guidelines were used in the development process of Rexx. These concepts include Readability, Natural data typing, Emphasis on symbolic manipulation, Dynamic scoping, Nothing to declare, System independence, Limited span syntactic units, Dealing with reality, Documentation before implementation and The language user is usually right.

5.1.1.1 Basic Syntax

The next pages include some basic syntax descriptions and examples about classic Rexx but are full compatible with ooRexx. The aim is to provide a short but useful introduction to Rexx and further to ooRexx.
A program in Rexx is created out of a series of clauses composed by different elements. Zero, blanks, a sequence of tokens, the delimiter ";" (may be implied by line end), certain keywords, the colon ":", … . Clauses in Rexx are classified as Null clauses, Labels, Instructions, Assignments, Keyword Instructions and Commands.
In Rexx each Sentence is scanned from left to right before execution and the tokens composing it are identified. After reading and identification the instruction keywords get recognised, comments are removed and multiple blanks (except within literal strings) are reduced to single blanks. [CoSh90]

Comments

Comments are ignored lines of characters. There are two ways to include comments into a Rexx script. The first is the Java like way of adding a comment. The additional information is includend within two "/*" signs. The second way is to indicate the comment with two dashes. Everything in the same line, behind these dashes, is ignored by the interpreter. This way is convenient for short comments.

```
/*   This is a valid comment   */
--   This is also a valid one
```

code 17) Example comment in Rexx

Rexx programs should, by recommendation, start with a comment describing the content and purpose of the program.

Strings, operators, and special characters

The essential parts or components of the clauses in a Rexx program are called tokens. Tokens are Strings, Symbols, Operator characters, Special characters. Strings can be either literals, hexadecimals of binary Strings. Strings in Rexx are delimited by either a single quote character or the double-quote character (“). A literal String is a constant and its content will never be modified. Literals may also be defined in hexadecimal or binary if required. To do so the literal String has to be followed by a "X" or "x" in the case of a hexadecimal and a "B" or "b" in the case of a binary.

Literals:

```
'This is the same'
“as this one”              /* there is no difference */
```

code 18) Example String in Rexx [CoSh90]

Hexadecimal:

```
'ABCD'x
“1d ec f8”X
'123 45'x                  /* same as '01 23 45'x */
```

code 19) Example Hexadecimal String in Rexx [CoSh90]

Binary:

```
'11110000'b                /* == 'f0'x  */
“101 1101”B                /* == '5d'x  */
'1'b                       /* == '00000001'b and '01'x */
```

code 20) Example Binary String in Rexx [CoSh90]

Operator characters:

To indicate operations the following characters are used in Rexx. They might also be used in combination. The equal sign is also used to indicate assignments.

```
+   -   *   /   %   |   &   ?   \   >   <
```

code 21) operator characters in Rexx [CoSh90]

The "¬" not Symbol is the same as the backslash "\". The symbols may be used interchangeably.

Special characters

The following characters have a special significance when used outside of a Literal String. E.g. ('Repeat' B + 3;) is treated like ('Repeat' B+3) due to the ";" operator.

```
, ; : ) (
```

code 22) special characters in Rexx [CoSh90]

Operators

Expressions in Rexx consist of terms and operators to connect them with each other. Expressions are included within clauses and refer to a mechanism of combining one or more pieces of data to produce a result. The form of the result is usually different from the original data.

Term:

Literal Strings	constants
Symbols	Group of any characters (e.g. tom or Dan.Yr.Ogof)
Function calls	symbol([expression] [, [expression]] …) string[expression] [, [expression]] …)
Sub expressions	Any expression within parenthesis

Figure 19) Possible Terms in Rexx [CoSh90]

Operators:

Operators in Rexx can be divided into four groups. The table below shows the valid operators in Rexx divided in these four Groups.

Concatenation	(blank) || (abuttal)
Arithmetic (require involved Terms to be numbers)	+ - * / % // **
Comparative	=, [?=, \= (not equal)], >, <, [<>, >< (same as not equal)] <=,>=,?<,?>, Strict comparatives: == (identical), ?==, >>,<<,>>=,<<=,?>>,?<<,
Logical (Boolean)	& And (returns 1 if both are true) | Inclusive or (returns 1 if either term is true) && Exlusive or (returns 1 if either (but not both)term is true) Prefix ?, \ Logical not (negates; 1 becomes 0 and vice versa)

Figure 20) Possible Operators in Rexx [CoSh90]

Operators relay to a hierarchical system were its precedence is determined. Expressions which include operators with higher precedence are executed fist. In the table you can see the precedence of operators: (top to bottom hierarchy)

Prefix operators:	`+ - ? \`
Power operators:	`**`
Multiplication and division	`* / % //`
Addition and subtraction	`+ -`
Concatenation	`(blank) || (abuttal)`
Comparative operators	`= == > < >= ?> << \>> etc.`
AND	`&`
Or, exclusive or	`| &&`

Figure 21) Precedence of Operators in Rexx [CoSh90]

Variables

A variable is an object whose value might be changed during the execution of the program. The value of a variable is assigned to it. Values can be retrieved e.g. vie the PULL, PARSE, ARG statements or simply by using the assignment statement. `symbol=expression;`

The result of the expression becomes the new value of the variable named by the symbol. The next two lines show a simple assignment process.

```
fred="Frederic" /* Frederic is the new value of the Symbol fred */
fred=freya      /* fred gets now the value freya                */
```

code 23) Example for Variables in Rexx

Symbols can be divided into Constant symbols, Simple Symbols, Compound symbols and stems. Constant symbols cannot be assigned a new value. Simple Symbols are used for variables where the name corresponds to a single value. Compound once and stems are used for more complex variable collections. The table shows the different symbol types with possible symbols in the right column.

Constant symbols	`77 827.54`
Simple Symbols	`FRED ?12`
Compound symbols	`fred.3 Array.I.J`
stems	`fred. A.`

Figure 22) Possible Symbols in Rexx [CoSh90]

```
/*          Example for a Stem Symbol                          */
hole.   = "empty"
hole.19 = "full"
say     hole.1 hole.mouse hole.19    /* Says "empty empty full" */
```

code 24) An Example for Stems in Rexx [CoSh90]

Keyword Instructions

Consist of one or more clauses indicated by the first word in the chain. Some of the Keywords affect the flow of control in a program. Such instructions can include nested instructions like in the "do" instruction. Here is a list of Keyword Instructions for Rexx:

```
ADRESS, ARG, CALL, DO, DROP, EXIT, INTERPRET, ITERATE, LEAVE, NOP,
NUMERIC, OPTIONS, PARSE, PROCEDURE, PULL, PUSH, QUEUE, RETURN,
SAY, SELECT, SIGNAL, TRACE,
```

Figure 23) Keyword Instructions for Rexx [CoSh90]

Build in functions

There are several built-in function defined in the Rexx language. In addition there are also external functions available tailored to the operating system under which the Rexx processor runs. The built-in functions include character manipulation, conversion, and information functions. Here is a list of several available Rexx functions. (the arguments in the squared brackets are optional)

```
ABBREV(information,info[,length]),     ABS(number),     ADDRESS(),
ARG([n[.option]]),              BITAND(string1[,[string2][,pad]]),
BITOR(string1[,[string2][,pad]]),        BITXOR(string1[,[string2]
[,pad]]),     B2X(binary-string),     CENTER(string,length[,pad]),
CHARIN([name][,[start][,length]]),        CHAROUT([name][,[string]
[,start]]),     CHARS([name]),     COMPARE(string1,string2[,pad]),
CONDITION([option]),       COPIES(string,n),       C2D(string[,n]),
C2X(string),        DATATYPE(string[,type]),        DATE([option]),
DELSTR(string,n[,length]),  DELWORD(string,n[,length]),  DIGITS(),
D2C(whole-number[,n]),     D2X(whole-number[,n]),     ERRORTEXT(n),
FORM(),    FORMAT(number,[,[before][,[after]]]),    FORMAT(number[,
[before][,[after][,[aexpp][,expt]]]]), FUZZ(), INSERT(new,target[,
[n][,[length][,pad]]]),          LASTPOS(needle,haystack[,start]),
LEFT(string,length,[,pad]),  LENGTH(string),  LINEIN([name][,[line]
[,count]]),    LINEOUT([name][,[string][.line]]),    LINES([name]),
MAX(number[,number]...),                 MIN(number[,number]...),
OVERLAY(new,target[,[n][,[length][,pad]]]),
POS(needle,haystack[,start]),        RANDOM(max),        QUEUED(),
REVERSE(string),     RIGHT(string,length[,pad]),     SIGN(number),
SOURCELINE([n]),                         SPACE(string[,[n][,pad]]),
STREAM(name[,operation[,streamcommand]]),   STRIP(string[,[option]
[,char]]),                     SUBSTR(string,n[,[length][,pad]]),
SUBWORD(string,n[,length]),     SYMBOL(name),     TIME([option]),
TRACE([setting]),   TRANSLATE(string[,[tableo][,[tablei][,pad]]]),
```

```
TRUNC(number[,n]),              VALUE(name[,[newvalue]][,selector]),
VERIFY(string,reference[,[option][,start]]),       WORD(string,n),
WORDINDEX(string,n),                              WORDLENGTH(sting,n),
WORDPOS(phrase,string[,start]),    WORDS(string),    XRANGE([start]
[,end]), X2B(hex-string),  X2C(hex-string), X2D(hex-string[,n])
```

Figure 24) Built-in functions of Rexx [CoSh90]

In- and Output

The Rexx model for in and output consists of three different types the character input stream, the character output stream and the external data queue. Here is a summary of in and output instructions and functions:

```
CHARIN,  CHAROUT,  CHARS,  LINEIN,  LINEOUT,  LINES,  PARSE  LINEIN,
PARSE PULL, PULL, PUSH, QUEUE, QUEUED, SAY
```

Figure 25) Summary of in and output functions in Rexx [CoSh90]

The above mentioned functions, instructions, … provide a short overview and introduction to the Rexx language. For a specific understanding of all or one of the mentioned examples studie the following examples of Rexx and further ooRexx and in addition research the following resources [CoSh90], [Fos05], [RexxLA09], [ooR09].

5.1.1.2 Basic Examples

The next pages include some basic examples about using Rexx. The first one is an example of how to define a function in Rexx.

```
/* using the ARG instruction                                          */
/* function is invoked by    FRED('Ogof X',1.5)                       */
FRED:   arg string, num1, num2
/*    String contains 'Ogof X'                                        */
/*    num1 contains '1'                                               */
/*    num2 contains '5'                                               */
```

code 25) A Function in Rexx [CoSh90]

This example presents how to pull a string and present this string to the system:

```
/* PULL something and SAY it                                          */
SAY 'Please enter your name:'
PULL name
SAY 'Hello there,' name'!';
EXIT 0
--  The End
```

code 26) Pull a Name with Rexx [Prem06]

This example demonstrates the usage of variables in Rexx:

```
/* Rexx and variables                                                    */
a = 5; b = 6; c = a + b
d = 'String01'; e = 'String02'; f = d || e
SAY a b c; SAY d e f
EXIT 0
```

code 27) Variable manipulation with Rexx [Prem06]

An example of a simple “do” instruction in Rexx.

```
do i=1 to 4
    if i=2 then iterate
     say i
     end                        /* would display the numbers: 1, 3, 4  */
```

code 28) Do Instruction in Rexx [CoSh90]

This example presents the usage of the “select” instruction in Rexx.

```
Testfile myfile
select
        when rc=0 then do
                Erase myfile
                say 'File' myfile 'existed, now erased'
                end
        when rc=28 | rc=36 then say myfile 'does not exist'
        otherwise
                say 'Unexpected return code “'rc'” from TESTFILE'
                exit rc
        end /* Select */
```

code 29) Select instuction in Rexx [CoSh90]

The next one explains how to call a function in Rexx. The statement “call” simply invoke the function “Factorial” and passes the symbol x whose variable is defined by the argument.

```
/* Example of recursive subroutine execution                            */
arg x
call factorial x
say x'! =' result
exit

Factorial: procedure
        arg n                   /* recursive invocation                 */
        if n=0 then return 1
        call factorial n-1
        return result*n
```

code 30) Function call in Rexx [CoSh90]

The next two examples show how to use a routine in Rexx. The routine multiply is called by the first program and passes the values 12 and 2 to the routine (code 32).

```
/* calling.rex                                                          */
result=multiply(12,2)
SAY result
::REQUIRES routine.rex
```

code 31) Using a routine in Rexx [Prem06]

```
/* routine.rex                                                    */
::ROUTINE multiply PUBLIC
result = 0
USE ARG x, b
DO b
result = x + result;
END
RETURN result
```

code 32) Using a routine in Rexx_2 [Prem06]

The presented examples and prior described syntax provides just an overview of the classic Rexx features for a deeper understanding and further research we recommend to take a look at the following [CoSh90], [Fos05], [RexxLA09]

5.1.2 Why Open Object (oo) Rexx

Back in the mid-1990 IBM developed a product called object Rexx. With several distributions for Linux, Windows, OS/2, … . In 2004 oRexx was transferred by ooRexx which means Open Object Rexx. The original IBM product became an open source project managed by the Rexx Language Association. At the date of transition the two versions (oRexx, ooRexx) were identical.

OoRexx added new features to the classical Rexx approach as shown in the table below. Beside this new features there are several reasons why open object Rexx is a powerful language. According to [Fos05] ooRexx extents classic Rexx in three ways of benefits. The first are benefits out of the object oriented paradigm. The second is related to the extension of classic Rexx with the object oriented paradigm and the third one is related to shell scripting with ooRexx. One further advantage is its easy to learn and use paradigm. [Flat05_b]

Open Object Rexx extensions to classic Rexx	Complete Object Orientation
	Classes and Methods
	Built-in Objects, Special Variables, many other features
	New Operators
	More Functions
	New Instructions
	...

Figure 26) ooRexx extensions to classic Rexx [Fos05]

5.1.2.1 Features

The following table includes a few features related to the topics named in the first column.

Object orientation	Code reuse, lower error rate (proven components), existing objects, modelling, Rapid prototyping
Extension of classic Rexx	Objects, Classes, Subclasses, Superclasses, … , class libraries, public and private methods, inheritance, multiple inheritance, encapsulation, polymorphism, method chaining, many interfaces
Shell scripting	Easy use, high productivity, portability, extra utilities, ...

Figure 27) ooRexx features [Fos05]

Further features of ooRexx are: [Prem06], [Burger05]

- English like statements
- Fewer rules
- Interpreted not compiled
- Type less variables
- String handling
- Decimal Arithmetic
- Clear error messages and powerful debugging

Object Rexx runs on almost any available platform. Windows, Unix, Linux, BSD, Mac OS, IBM iSeries, OS/2, DOS, handheld operating systems, … [Aha05], [RexxLA09]

5.1.2.2 Basics of ooRexx

OoRexx fully supports the object oriented paradigm and is therefore providing objects to use. Objects contain methods and variables, more accurately called attributes. Methods are invoked by sending a message to the object. Rexx is using the tilde operator "~" to send the message to the object and run its method.

This is the classical approach of Rexx, by using a check_in function.

```
feedback = check_in(book)              /* run user-defined CHECK_IN function */
```

code 33) Classical function call [Fos05]

The new approach of ooRexx sends a message to the object book to invoke its check_in method.

```
feedback = book~check_in                    /* run user-defined CHECK_IN method */
```

code 34) Calling a method with ooRexx [Fos05]

Instances of objects are created by calling the “new” method.

```
new_book = .book~new                        /* new method creates a new object */
```

code 35) Creating a new instance [Fos05]

If a new object is created ooRexx automatically invokes the init method (if defined) to initialize the passed variables.

```
new_book = book~new('Harry Potter'.'Publisher',1998)
```

code 36) Creating a new instance and pass arguments [Fos05]

To define objects ooRexx uses directives. Directives are used to set up classes, methods, and routines. Directives start with a double colon “::”. The most common directives in ooRexx are

```
                ::class     ::method    ::routine   ::requires
```

code 37) Directives in ooRexx [Fos05]

In the next example the usage of directives is shown. A new object is defined by creating the book class and its methods as shown. This class can now be instantiated. The term private locks in the replacement_cost method, i.e. the method can only be invoked by other methods of the class whereas the other methods are publicly available. Like classical Rexx we can use the use arg statement to initialise variables to the class. The expose instruction make the variables available to the other methods.

```
::class book

::method init
        expose title   book_id   patron_id
        use arg title, publisher, pub_date
::method check_in
        ...
::method check_out
::method replacement_cost private
        expose pages cost_per_page replacement
        replacement_amount = pages * cost_per_page
        return
```

code 38) ooRexx class example [Fos05]

In the prior examples we saw a few new characters. OoRexx provides new operators to its users. The table below shows a list of new operators in ooRexx.

New operators (ooRexx):	
~	send a message to invoke a method and return its result

~~	send a message to invoke a method, return the object or the message
[]	add or retrieve objects to/from a collection

Figure 28) New operators in ooRexx [Fos05]

OoRexx is not only providing new operators. There are also new special variables in ooRexx. Special Variables are: self and super Referring to the current executing method and the superclass method respectively.

Further there are also new keyword instructions in ooRexx. E.g. we used the expose instruction in the class example above to make attributes available to a method that can change them. The new instructions are shown in the table beneath.

Instructions	Use
Expose	Permits the access to a variable
forward	Forwards the message that caused the active method to execute
guard	Controls concurrent execution of a method
raise	Raise an error condition or Trap explicitly
reply	Sends an early reply from method to caller
use	Retrieves arguments

Figure 29) New instructions in ooRexx [Fos05]

5.1.2.3 Class libraries

Due to the object oriented paradigm it is possible to collect classes in libraries and use them. The libraries consist of a set of built-in classes. Like in Java the time needed to create a program depends on the power of the available class libraries. The following tables include lists of the most common class libraries in ooRexx. They can be divided into different categories e.g. we divide into collection classes at first.

Array, Bag, Directory, List, Queue, Relation, Set, Table

Figure 30) List of collection class libraries in ooRexx [Fos05]

Collection classes refer to groups of objects and methods used for there manipulation. For example the Array classes refer to a sequenced collection or the List classes to a sequenced collection which allows inserts at any position. In addition there are several libraries more. Just to mention a few examples we added the following list of ooRexx classes.

Alarm, Class, Message, Method, Monitor, MutableBuffer, Object, Stem, Stream, Sting, Supplier

Figure 31) List class libraries in ooRexx[Fos05]

Every class has his own methods some provide just a few whereas others provide dozens.

5.1.2.4 Built in objects and new instructions

The built-in objects are always available for scripting with ooRexx and therefore ease the way of producing a script. For built-in functions it is not necessary to call the object separately it can be instantiated straight forward. This objects are helpful for the interaction with their environment, input and outputting, to view environmental parameters or inspect return code. Here is a list of built in objects in ooRexx.

.environment, .nil, .local, .error, .input, .output, .methods, .rs, .true, .false, .stdin, .stdout, .stderr

Figure 32) Built-in objects in ooRexx [Fos05]

As it would tremendously extend the frame of this paper there is no description of each class , or method provided by ooRexx. To achieve a deeper understanding study the following examples and take a look at the documentation of ooRexx at [ooR09].

5.1.3 Samples

The next few pages include a few examples of specific ooRexx programs. The first example starts by determining if a specific file exists or not. Either if it is existing or not is printed to the screen.

```
/* Tells if a specific file exists                                    */
Parse arg file.                                    /* get user input file */
Infile = .stream~new(file)                        /* create stream object */

/* Existence test returns either full filename or the null string     */

if infile~query('exists') = '' then                /* test if nonexistent */
        .output~lineout('File does not exist')     /* no such file exists */
else
        .output~lineout('File exists')              /* found the filename */

exit 0
```

code 39) Using classes in ooRexx [Fos05]

The second example deals with the return of a squared number. This is done by defining a class called “squared”. This class gets instantiated and then the method 'square' is used to square the number and to return the result.

```
/* Returns the Square of a number                                    */
parse arg input                          /* get the number to square from user */

value = .squared~new                    /* create an object for a squared value */
sqd = value~square(input)                /* invoke SQUARE method on INPUT value */

say 'The original value:' input 'squared is:' sqd
exit 0                                          /* cause the exit of the program */

::class squared
        ::method 'square'                /* Class SQUARED has 1 method, SQUARE */
        use arg in                                          /* get input argument */
        return ( in * in )                 /* Square it and return that value */
```

code 40) Defining a class in ooRexx [Fos05]

The last example also defines a class. Two methods are used in this class the first one is the init method which prompts for the shell name and afterwards exposes the shell variable to the other method. The init method is directly called while instantiating the class object with .operating_system~new. The second method determines which operating system refers to the variable shell and returns a result.

```
/* Tells which operating system you use depending on the command shell        */
os = .operating_system~new                                  /* create a new object */

os~write_command_shell                    /* invoke the method to do the work */

exit 0

::class operating_systems                                /* class with 2 methods */

        ::method init                              /* method prompts for shell name */
        expose shell                                 /* EXPOSE the shared attribute */
        say 'Enter the shell name:'
        parse pull shell
        return

        ::method write_command_shell                    /* method determines the OS */
        expose shell
        select                                  /* determines the OS for this shell */
                when shell = 'CMD'              then string = 'DOS or Windows 9x'
                when shell = 'COMMAND'          then string = 'Windows 2k/2003/XP'
                when shell = 'ksh'              then string = 'Korn Shell'
                when shell = 'csh'              then string = 'C Shell'
                otherwise string = 'unknown'
        end
        say 'OS is:' string
        return 0
```

code 41) Class example with ooRexx [Fos05]

5.2 BSF, Bean Scripting Framework

Before we talk about the ooRexx extension BSF4ooRexx we take a look at the BSF (Bean Scripting Framework (Link 20)) itself. The reader should be enabled to gain a basic concept understanding of what a Bean Scripting Framework is providing to him.

"Bean Scripting Framework (BSF) is a set of Java classes which provides scripting languages support within Java applications, and access to Java objects and methods from scripting languages ….." [Burger05], (Link 20)

According to this BSF provides a pretty cool thing i.e. the usage of Java beans within a scripting language as it is used in BSF4ooRexx. But not only this access from within a scripting language is possible also the usage of a scripting language within Java.
BSF was developed by IBM and released in 1999. After that, BSF was subordinated to the Apache Jakarta project in 2002. Currently BSF is used in two versions (Link 20) with different APIs. The original version of BSF uses the original APIs developed by IBM. This version is represented by the 2.x versions whereas the new version is represented by the 3.x versions. The new version of BSF uses the API defined as part of JSR-223 (javax.script), which is included in Java 1.6 onwards. However, version 3.x will also run on Java 1.4+, allowing access to JSR-223 scripting for Java 1.4 and Java 1.5.

The currently supported scripting languages by version 2.x are:

Javascript, NetRexx (an extension of the IBM REXX scripting language in Java), Commons JEXL, Python (using Jython), Tcl (using Jacl), XSLT Stylesheets (as a component of Apache XML project's Xalan and Xerces) (Link 20)

Additionally there are several other languages which can be used with BSF by using a separated designed BSFEngine. These are:

Java (using BeanShell, from the BeanShell project), Groovy, Groovy Monkey, JLog (PROLOG implemented in Java), Jruby, JudoScript, ObjectScript, **ooRexx (Open Object Rexx), using BSF4Rexx/BSF4ooRexx**. (Link 20)

The basic parts of BSF are:

1) the BSFManager
2) the BSFEngine

1) The BSFManager allows the access to the Java objects and manage the scripting execution engines. A Java application can get access to the BSFManager class by creating an instance of it. [Prem06], (Link 20)
2) The BSFEngine is the language specific part and needs to be implemented for each language which is intended to be used. The BSFEngine is the abstraction of the scripting language, handling the scripting execution and object registration. One BSFManager can be used by different languages. An once registered object stays alive until the Manager is shut down. [Prem06], (Link 20)
The Graphic visualizes this concept and shows the connection between the BSFEngines and the BSFMangager.

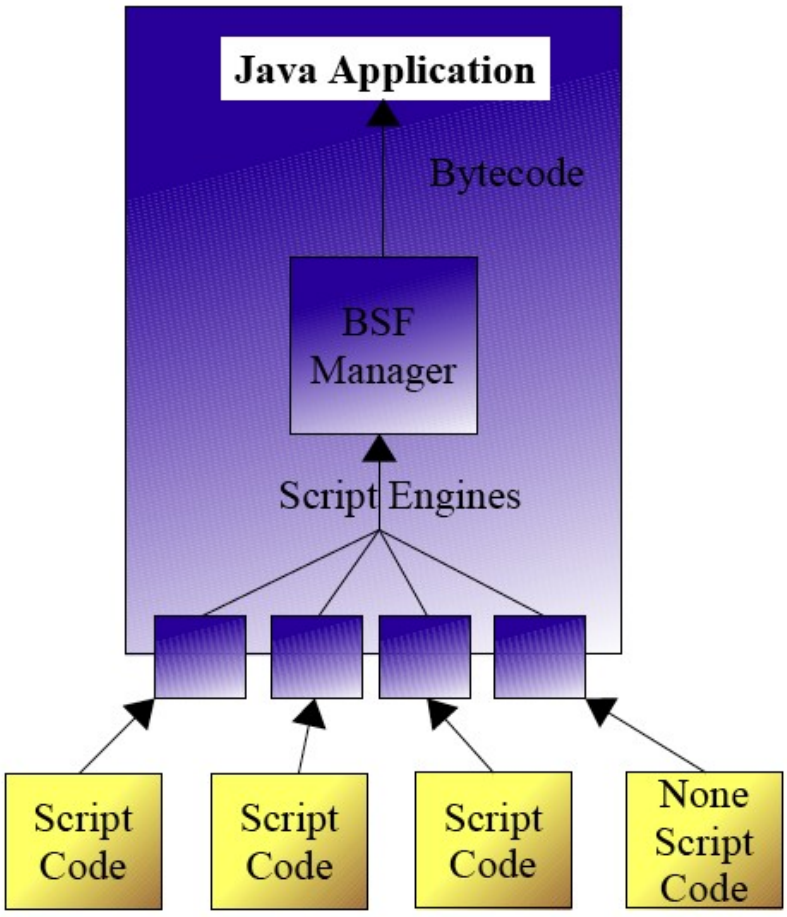

Figure 33) Concept of BSF [Aha05]

5.3 BSF4ooRexx, Bean Scripting Framework for Object Rexx [BSF4ooR]

Expanding a programming language with the ability of a Bean Scripting Framework is an interesting, useful, and - last but not least - a very cool thing.
Before we saw that there is an external support for BSF to enable ooRexx the scripting of Java and its class libraries. The basic concept is the usage of a separated BSFEngine that provides the abstraction from the scripting language. With this in our pockets we should be able to use all of the available Java classes and therefore Project Rome too.

5.3.1 Basic concepts and examples

Starting with BSF we need to get a connection to its BSFManager which allows the access to the Java objects via creating an instance of this manager. This connection is handled by the BSFEngine which is the language specific part of the Bean Scripting Framework. BSF can therefore be used by several different languages. The current version for ooRexx providing this engine and all its related features is called BSF4ooRexx and available under [BSF4ooR]. BSF4ooRexx is an open source, extension Package for ooRexx (newest version4.0). The newest version of this package is by intention called BSF4ooRexx due to the fact that it is tailored to the newest version of ooRexx (version 4.0) and therefore only usable with this. The prior version is called BSF4Rexx and is available under [BSF4R]. This paper is beside its other purposes related to the newest version BSF4ooRexx. Each created example program (Appendix) includes the information of the version of ooRexx, BSF4ooRexx, and the underlying Java program.

The basic concept of BSF4Rexx has been shown in the introduction. This concept referred to [Flat01] has been developed by the time, therefore the newest version include a few extensions to this basic concept. The basic concept deals with the connection between java and Rexx scripts and vice versa. This has been done by implementing the kernel of the shared link library of BSF4Rexx which is written in C++. This kernel provide the basic functions (external to BSF.CLS) to interface with Java. If Java invokes the script this basic functions are already registered. In the following graphic you can see some of the functions within the cloud, related to the BSF4Rexx kernel [Flat01].

By taking a look at the graphic we can recognise that there is even more related to the BSF4ooRexx Framework. We can see that the connection to the BSFEngines, either RexxEngine and RexxandJava runs through a class library called BSF.cls and that there is a so called BSF Registry between the JavaandRexx engine and the Java program. The BSF.cls is a huge and powerful class library delivering routines, classes, methods, … to ease the communication with the Java side and vice versa. Public routines are for example "bsf.loadClass(...)", "bsf.lookupBean(...)" or bsf.pollEventText(...), … . Classes provided by BSF4ooRexx are the Public Class BSF, the BSF.DIALOG class, the BSF_ARRAY_REFERENCE class and the BSF_PROXY class. Each deliverers different methods for the interfacing between the languages. E.g. bsf.sleep(...) or bsf.pollEventText(...) are methods from the Public Class BSF. An other example is the BSF.DIALOG class providing methods like messageBox(message, [title], [type]), dialogBox(message, [title], [type], [optionType], [icon], [txtButtons],[defaultTxtButton]), inputBox(message, [title], [type], [icon], [txtOptions],[defaultTxtOption]) to enable a fast use of Java message, dialog and input boxes.

Finally the BSFRegistry is created to ease the process of creating Java array objects. Therefore they are preregistered on the Java side. As with ooRexx, Java … a full description would exceed the purpose of this paper we recommend to research the following [Flat01] – [Flat09_b], [Prem06], [Aha05], [BSF4ooR] (included examples of this paper),

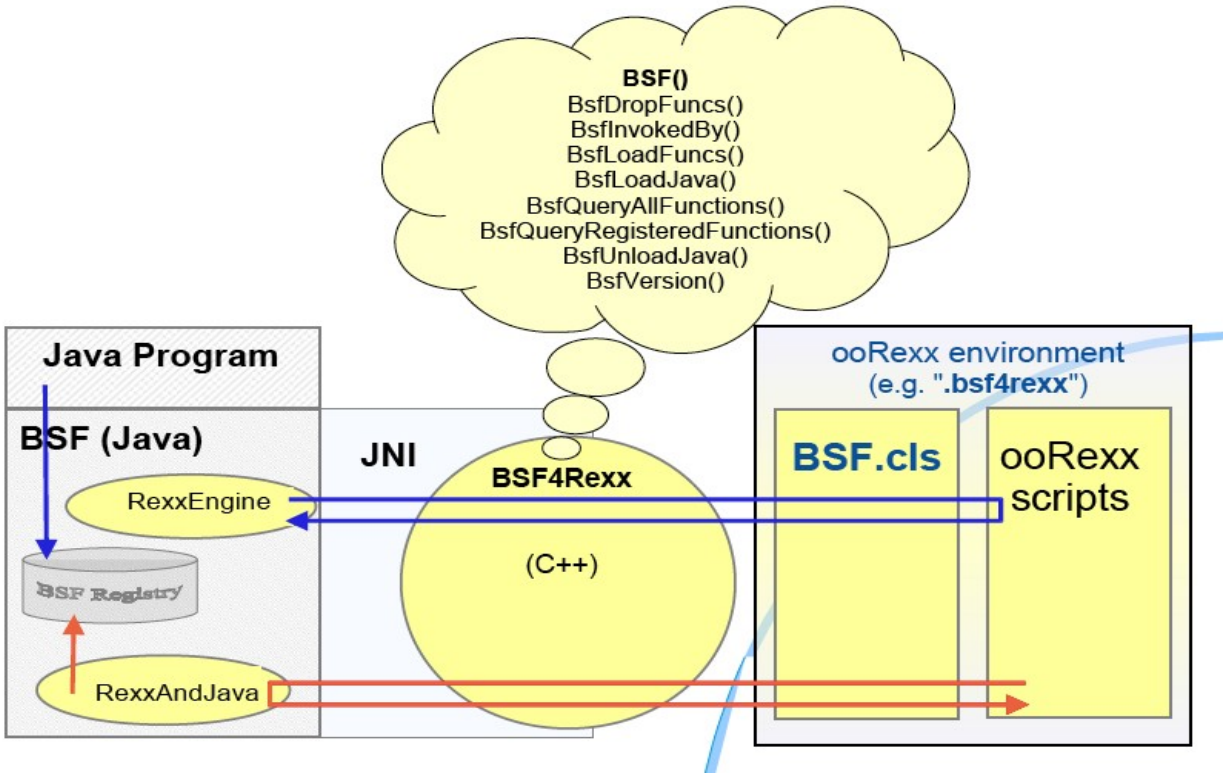

Figure 34) Architecture of BSF4Rexx [Flat06]

5.3.2 Installation, Getting the BSF support, execution

Due to a lack of installation features the “Vienna Version” of BSF4Rexx includes not only all of the files to run BSF4Rexx but also a setup routine that eases the installation of BSF4Rexx [Prem06], [Flat03]. This routine is also included in the newest version “BSF4ooRexx” which is working with the newest ooRexx version 4.0.

To install the extension package download it from [BSF4ooR] and unzip the archive "BSF4ooRexx_install.zip" into the target directory. This package include the full support for BSF and also the support for StarOffice and therefore the UNO.cls.
To proceed with the installation uninstall prior installed versions of BSF4Rexx by changing into the target directory and execute both “uninstallBSF.cmd” and “uninstallOOo.cmd” if necessary. To install BSF4ooRexx execute the Rexx script setupBSF.rex which creates the file “installBSF.cmd". After executing the “installBSF.cmd” all the necessary path information will be added to the registry and after logging off and on the BSF support will be in effect. The procedure is the same for the installation of the StarOffice support. By executing the script “setupOOO.rex” and the relating files the StarOffice support will be in effect.
The files “readmeBSF4ooRexx.txt” and “readmeOOO.txt”, delivered with BSF4ooRexx package, provide a precise description of the installation process.

There are two ways to include the BSF support into a Rexx script. This can be done either by calling “BSF.cls” before the first use of BSF. This is necessary because otherwise it would result an error.

```
/*Using call for including BSF support*                                    /
CALL BSF.CLS
.bsf.dialog~messagebox("This is a Java message box.")
SAY "Did you see it?"
EXIT 0
```

code 42) Calling BSF.CLS [Prem06], [Flat09_b]

The second way is to include the “::REQUIRES BSF.CLS” directive into the script. This allows the use of BSF whenever we want because a directive is invoked before the script gets executed. This is the reason why this way is the most preferable one. Here is a short example of how to include the directive.

```
/* Using require for including BSF support                                  */
.bsf.dialog~messagebox("This is a Java message box.")
SAY "Did you see it?"
EXIT 0
::REQUIRES BSF.CLS
```
code 43) Including ::Requires BSF.CLS [Prem06], [Flat09_b]

In either cases we are sending a message to the ".bsf.dialog" class included in the "BSF.CLS" library package. The message "~messagebox" calls the corresponding method included in the ".bsf.dialog" class and passes the string arguments.

Under 5.1.3.4 we discussed the topic of ooRexx class libraries, "BSF.CLS" is representing such a library. The library include all the classes, methods, routines, … necessary for the implementation of the BSF support. The reference cards (delivered with BSF4ooRexx) [BSF4ooR] provide a realy useful overview of the included features.

To execute a Rexx program that uses BSF, we can either call it via the "rexx" command or the "rexxj" command. This two commands refer to totally different approaches of execution. The first one calls the rexx interpreter and Java gets loaded beside the runtime of the program. The second one provides the opposite approach. First Java is called and the filename is passed to Java which executes the program.

5.3.3 The newest features of BSF4ooRexx

Due to some restrictions relating the Rexx APIs it was not possible to overcome all shortcomings of BSF4Rexx without implementing new APIs. However, 2009 a new version of ooRexx has been released including a new kernel and therefore also new features for BSF4Rexx became possible. According to [Flat09_a] the prior shortcomings include:

1. Disability of creating ooRexx proxy objects for Java
2. No real-time handling of Java events
3. No implementation of Java interface methods within Rexx methods
4. No support to implement Java abstract classes within Rexx methods
5. Rexx condition communication to Java is possible
6. No support for Java exceptions

To overcome these, with the new version of ooRexx a few new features were created like the usage and definition of Java interfaces on the Rexx side, the usage of Java abstract classes and the ability to through Java exceptions from Rexx. [Flat09_a] provides an introduction to this shortcomings and there solutions.

5.3.4 A short example using SAX

Under 4.0.4 we included an example of a SAX parser. As there are examples of SAX parsers delivered with the BSF4ooRexx package the next example is related to this examples and shows how to use the Java SAX library with ooRexx via BSF4ooRexx. Furthermore the example provides a basic introduction to the usage of Java interfaces via BSF4ooRexx. For more information about using Java interfaces or abstract classes take a look at [Flat09_a].

```
/*            This example uses the Java SAX classes via BSF4ooRexx           */
use arg xmlFileName
rexxObject=.saxHandler~new     /* creates an instance of the SaxHandler class */
rexxProxy=BsfCreateRexxProxy(rexxObject)/* creates a Rexx Proxy from r-object */
objArray=bsf.createArray(bsf.loadClass("java.lang.Object"), 1)
objArray[1]="org.xml.sax.ContentHandler"        /* Java array with one object */
javaProxy =rexxProxy~newJavaProxyInstance(objArray)
/*      allows the connection between .saxHandler and the Chandler Interface  */
rexxObject2=.dtdh~new              /* creates an instance of the .dtdh class */
rexxProxy2=BsfCreateRexxProxy(rexxObject2)/*creates a Rexx Proxy from r-object*/
objArray2=bsf.createArray(bsf.loadClass("java.lang.Object"), 1)
objArray2[1]="org.xml.sax.DTDHandler"          /* Java array with one object */
javaProxy2=rexxProxy2~newJavaProxyInstance(objArray2)
/*      allows the connection between .dtdh and the DTDHandler Interface      */
parser=bsf.loadClass("org.xml.sax.helpers.XMLReaderFactory")~createXMLReader
parser~setContentHandler(javaProxy)   /* creates a Rexx Proxy object */
parser~setDTDHandler(javaProxy2)

parser~parse(xmlFileName)

::requires BSF.CLS                                  /*   get the BSF support   */

::class "dtdh"         /*   including the interface support for DTD Handler   */

::method unparsedEntityDecl
    use arg name, publicId, systemId, notationName
    say pp(.bsf~new("java.lang.String",
                          name, publicId, systemId, notationName)~toString)

::method notationDecl
    use arg name, publicId, systemId
    say pp(.bsf~new("java.lang.String", name, publicId, systemId)~toString)

::class "saxHandler" /* including the interface support for the SAXHandler   */

::method startDocument                     /* Shows the start of the document */

say pp(this is the start)

::method endDocument                        /* Shows the end of the document */

say pp(this is the end)

::method unknown
```

code 44) Example using Java SAX [BSF4ooR]

6.0 Usage of BSF4ooRexx with the Project ROME API (ooRexx code snippets)

The Basic parts and concepts of ooRexx and its extension BSF4ooRexx were described in the previous chapter. Now we are going to take a more precise focus on the usage of the PR API which is provided in a Java class library (rome1.0.jar). The prerequisites for the execution of the examples in the appendix and therefore also for the following code snippets is the installation of Java, ooRexx and the including of the PR library into the Java resources. Finally we have to install BSF4ooRexx to get the scripting support for the Java classes. These things have been described in in detail in the prior chapters.

This chapter deals with content syndication via the Project ROME Java API and its scriptability through BSF4ooRexx. The following snippet examples should provide a short and profound introduction into how PR can be used in ooRexx. Compared with the prior discussed Java examples about the PR API this chapter provides a comparative and comprehensive introduction in the understanding of scripting Java via ooRexx.

To easy the access to the related resources we add some information in a short table above the code. The fist column is just the name / number of the code snippet related to the number of the example in the table of content and to the corresponding program number in the Appendix. The second is the name of the related file in the Appendix. Beneath the code you can find the number of the code snippet as it is named in the “table of code”.

name	*related program in the Appendix B*

6.1 ooRexx example 1 - Calling a Feed Reader Java class

Example 1 uses a predefined Java class, the "*FeedReader*" class. The class is instantiated in ooRexx via the statement .bsf~new(javaClass). After the instantiation process the Java method “.getfeed” is called via sending a message to the object by get~getfeed. This first example includes one of the main concepts for the following. As we need to instantiate most of the classes in Project Rome it is essential to know about this process.

ooRexx example 1	Read.rxj

```
javaclass = "FeedReader1"
/*                                          determine Java class to use   */
get=.bsf~new(javaClass)
/*                                       create an instance of "javaClass"   */
say get~getfeed
/*                          calls the getfeed method in the FeedReader1 class   */

::requires BSF.CLS/*              this is rewuired for all programs using BSF  */
/*      ************************************************************     */
```

code 45) Calling FeedReader (Java) class via ooRexx (BSF4ooRexx)

6.2 ooRexx example 2- Parsing a feed (using PR via BSF4ooRexx)

As we saw how to instantiate a predefined Java class in the last example this one is using the PR Java library itself. First we create a new *feedUrl* object with .bsf~new("java.net.URL",url). *Java.net.Url* is the full path to the Java class that we have to provide to the ~new method. After instantiating it is possible to use the Java method .getAuthority() by sending the message ~getAuthority() to the object. The result is a returned string with the authority part of this URL.

ooRexx example 2	rome.rxj

```
url= "http://rss.orf.at/fm4.xml"
feedUrl=.bsf~new("java.net.URL", url)  /* create an instance of java.net.URL  */
say connecting__ || feedUrl~getAuthority()
```

code 46) Using a Java class in ooRexx (BSF4ooRexx)

The next part of this example possesses the main part to parse a syndication feed. As you can see the process of parsing a feed is a rather short implementation due to this three lines of code we need to do so. First we start by instantiating the *SyndFeedInput* class from *com.sun.syndication.io* out of the PR package. This class provides a method to us that is called "build" and is therefore able to parse a news feed from different sources (W3C DOM, JDOM document, W3C SAX *InputSource*, a file or a Reader). In this case we used a *XmlReader* from *com.sun.syndication.io* . Finally the feed gets parsed by sending the "built" message to the *SyndFeedInput* object passing the *XmlReader*. The process "behind the scenes" was described in chapter 4.0.3 .

```
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
/*                                              creates a SyndFeedInput    */
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
/*     Figures out the charset encoding of the XML document within the stream */
feed= input~build(xmlr)
/*      Builds a SyndFeedImpl from an Reader , also possible with SAX or DOM  */
```

code 47) Calling FeedReader (Java) class via ooRexx (BSF4ooRexx)

Like we used SAX before to parse a Xml document we are also able to simply use its *InputSource* from *org.xml.sax* and pass it to the "built" method. As you can see there is just a small change within the code. The advantage in this case is that it is no longer necessary to provide a *java.net.Uri* object. A string which defines the identifier is enough.

```
xmlr=.bsf~new("org.xml.sax.InputSource", " /* URI String */ ")
```
code 48) Using the Sax InputSource with PR (BSF4ooRexx)

In the final snippet of this example we will now try to get some information from our parsed feed. By simply sending the message ~getEncoding() to the *SyndFeed* (feed) a string is returned and shown to the system with the "say" keyword instruction from Rexx.
The second "say" keyword instruction prints the symbolic Java object (`com.sun.syndication.feed.synd.SyndFeedImpl@8a0d5d`). Compared with 4.0.3 this is the created SyndFeed object provided by the named class.

```
say feed~getEncoding()
say feed                                          /* returns the feed object  */
```
code 49) Get some Information from a Syndfeed (BSF4ooRexx)

6.3 ooRexx example 3 - Using different methods

This example demonstrates the use of a couple of methods out of the *com.sun.syndication.feed.synd* class. First we have to create a feed parser as shown under example 2 to get a *SyndFeed*. After creating the *SyndFeed* we are able to use its methods as there are ~getTitle(), ~getURI, ~getAuthor(), … and much more.
The program also take advantage of the external Rexx function BSF() by invoking a system sleep as a function.

ooRexx example 3	rome_SyndFeed_methods.rxj

```
say "The Title of the feed is:"
say feed~getTitle()                              /* Returns the feed title */
sl = BSF("sleep",2)
say "----"
say "The URI of the feed is:"
say feed~getUri()                                /* Returns the feed URI  */
sl = BSF("sleep",2)
say "----"
say "The Author of the feed is:"
say feed~getAuthor()
```

```
/*    Returns the name of the first--feed author in the collection of authors */
sl = BSF("sleep",2)
say "----"
say "The copyright of the feed is:"
say feed~getCopyright()                          /* Returns the feed copyright */
sl = BSF("sleep",2)
say "----"
say "The description of the feed is:"
say feed~getDescription()                      /* Returns the feed description */
sl = BSF("sleep",2)
say "----"
say "The encoding of the feed is:"
say feed~getEncoding()           /* Returns the charset encoding of a the feed */
say "----"

sl = BSF("sleep",2)                         /* sleep by using a BSF subfunktion */
say just a break;
sl = BSF("sleep",2)                         /* sleep by using a BSF subfunktion */

say "----"
aut= feed~getAuthors()            /* Returns the feed authors , java.util.List */
say aut
say this is the Authors object
say "----"
array = feed~getEntries()         /* Returns the feed entries , java.util.List */
say array
say "this is the entries-array object"
```

code 50) Usage of SyndFeed methods (BSF4ooRexx)

The figure bellow presents the output from the corresponding program in the Appendix. This sample retrieved a feed from rss.orf.at with the URI http://fm4.orf.at . You can see the different returns from the used methods. The NIL object as under “encoding of the feed” denote that no information about the questioned information is available.

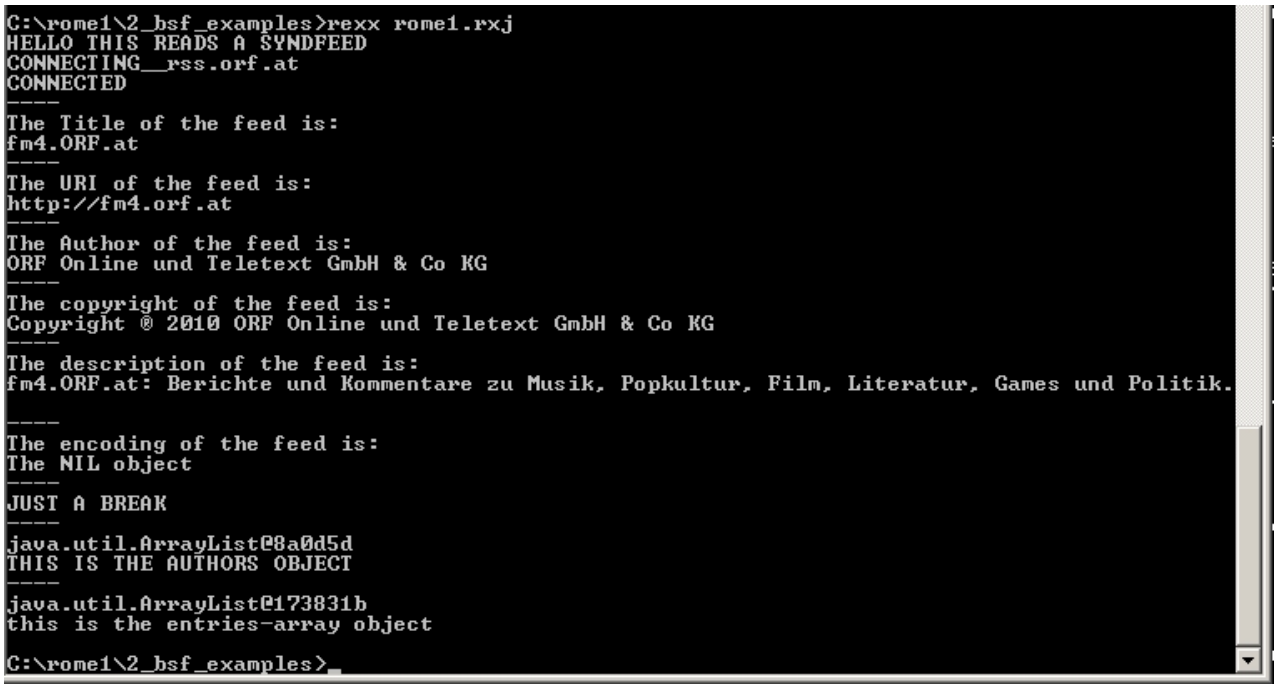

Figure 35) Example of different method outputs (ROME API) (using BSF4ooRexx)

6.4 ooRexx example 4 - Setting the feed type

As described under 4.0.3 the *SyndFeedImpl* class creates a *SyndFeed* which denotes to the independent object model. If created as a new one this model needs a *FeedType* definition to be outputted. In the case of a parsed feed the *SyndFeed* is of the same type as the original. However, it is possible to change the FeedType with the method ~setFeedType(“ /* default type */ “). To retrieve the current type we can use the ~getFeedType method.

ooRexx example 4	FeedReader_with_setType.rxj

```
feed~setFeedType("rss_1.0")   /* This line allows to set the type of the feed */
/* the default formats in ROME are: rss_0.90, rss_0.91, rss_0.92,             */
/* rss_0.93, rss_0.94, rss_1.0 rss_2.0 or atom_0.3                            */

say feed~getFeedType()               /* This line retrieves the current type */

feed~setFeedType("rss_2.0")

say feed~getFeedType()

feed~setFeedType("atom_1.0")

say feed~getFeedType()
```

code 51) Setting the FeedType of a SyndFeed (BSF4ooRexx)

Taking a look a the result of this program we can see its function. The first ~getFeedType method returns the type of the original feed retrieved from rss.orf.at. The second one shows the changed feed type of rss_2.0 as well as the third one with atom_1.0 .

Figure 36) Output of the set , get FeedType methods(using BSF4ooRexx)

6.5 ooRexx example 5 - Java Awt extension

Example 5 is not specifically dealing with PR but adds some additional features to our programs. As we are able to use the whole Java library as a big extension for ooRexx we are also able to create some graphical interfaces via Java Awt. [Flat09_b] provides some basic examples of how to use the Java AWT with BSF4ooRexx and therefore this example is related to the corresponding examples

there. First we create an instance of *java.awt.Frame* which we set to a FlowLayout by using the ~setLayout method. The function *bsf.EventLIstener* is used to recognise the closing of the created window. To keep the window open we use a "do forever" keyword instruction to constantly use the ~bsf.pollEventText method from the *.bsf* class. The used GUI includes one *java.awt.TextFild* and one *java.awt.Button* which are added to the prior created frame via the ~add method. Finally the Frame gets created by using the ~~pack~~show~~toFront methods.

ooRexx example 5	FeedReader_awt_extension.rxj

```
/*            graphical extension                                          */
frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.FlowLayout") )

tf=.bsf~new("java.awt.TextField", "", 50)
frame~add(tf)

but=.bsf~new('java.awt.Button', 'retrieve feed')
frame~add(but)
but~bsf.addEventListener('action', '', ' call text tf ')

frame~~pack~~show~~toFront

url= tf~getText

do forever
INTERPRET .bsf~bsf.pollEventText
end
```

code 52) Use of Java Awt with PR (BSF4ooRexx)

The resulting Java window is shown in the graphic bellow. Now it is quit easy to insert every feed address someone can imagine.

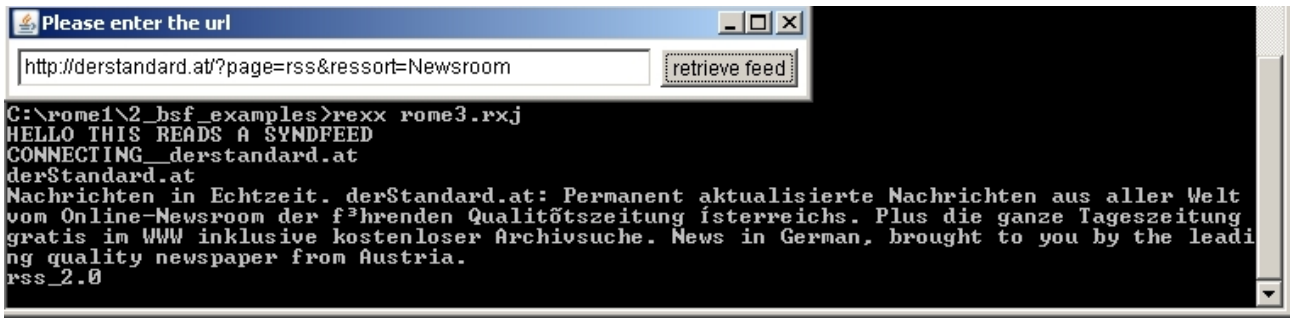

Figure 37) Example of PR Java Awt extension (using BSF4ooRexx)

6.6 ooRexx example 6 - Outputting a feed

This demonstrates how to get the output support from the ROME API. This example is independent from the prior process it is using either a created or a parsed *SyndFeed*. First the class *SyndFeedOutput* is instantiated and in a second step the method ~outputString prints the feed.

ooRexx example 6	FeedReader_FeedOutput.rxj

```
/*                      outputs from the SyndFeed, (independend object model) */
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
/*  Builds a SyndFeedOutput from a SyndFeed                                    */
say output~outputString(feed)

say output~outputString(feed,1)       /* usage of pritty print(0 yes or 1 no) */
```

code 53) Outputting a SyndFeed (BSF4ooRexx)

Because it is also possible to use the *WireFeedOutput* the second example shows the usage of this one. It is very similar to the *SyndFeedOutput* and also uses the same method for outputting but the passes feed must be in the *WireFeed* model.

```
/*                      outputs from the WireFeed,    (abstract object model) */
output=.bsf~new("com.sun.syndication.io.WireFeedOutput")
/*  Builds a WireFeedOutput from a WireFeed                                    */
say output~outputString(feed)
/* WireFeed as well as SyndFeed include the method outputString               */
say output~outputString(feed,1)       /* usage of pritty print(0 yes or 1 no) */
```

code 54) Outputting a WireFeed (BSF4ooRexx)

6.7 ooRexx example 7 - SyndFeed and WireFeed

As there is the possibility to output a *WireFeed* there is also one to create a *WireFeed*. Example 7 demonstrates a combined usage of a syndication feed parser. We used a *SyndFeedImpl* class to parse a feed in the previous examples. This class is returning a *SyndFeed* and therefore we use the method ~createWireFeed to convert to the corresponding model Type. Finished with this we are able to print both the *SyndFeed* and the *WireFeed*. (Program 14 in the Appendix B demonstrates the stand alone creation of a *WireFeed*)

One addition to the creation process of a *SyndFeed* (as described in 4.0.3) is the usage of the input healer method which is used before the *SyndFeed* gets build.

ooRexx example 7	SyndFeed_WireFeed.rxj

```
input~setXmlHealerOn(0)                                  /* set healer on off */
say input~getXmlHealerOn()                             /* checks for xml healer */
```

```
feed=input~build(xmlr)              /* Creates a SyndFeed out ouf the SFInput */
wfeed = feed~createWireFeed("rss_1.0")  /*  creates a WireFeed (abstract DM)  */
feed~setFeedType(typet)              /*   sets the feedtype of the SyndFeed  */

/*                      outputs from the SyndFeed, (independend object model) */
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
say output~outputString(feed,1)       /* usage of pritty print(0 yes or 1 no) */

say feed~getFeedType()

/*                           outputs from the WireFeed, (abstract data model)  */
output2=.bsf~new("com.sun.syndication.io.WireFeedOutput")
say output2~outputString(wfeed)
```

code 55) Creation of a WireFeed(BSF4ooRexx)

6.8 ooRexx example 8 – Writing a feed to a file

A *FeedWriter* is used to write the created or parsed feed to a file in a directory. The following example presents the usage of a so called *FeedWriter*. So called because it uses a *java.io.FileWriter* to carry out the writing process. Both classes *SyndFeedOutput* and *WireFeedOutput* include a method called output which can be used with *java.io.File* or *java.io.Writer*. There is also a method called ~outputJDom(SyndFeed feed) to create a JDom document out of the given *SyndFeedImpl*.

ooRexx example 8	FeedReader_Writer_combination.rxj;

```
/*                 writer part: saves the feed file as XML code to the system */
/*                 this writes the original feed to a file                    */
writer=.bsf~new("java.io.FileWriter",fileName)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

say output~outputString(feed,0)       /* usage of pretty print(0 yes or 1 no) */
```

code 56) Storing a feed to a file (BSF4ooRexx)

6.9 ooRexx example 9 - Aggregation

The fist part of this example deals with the creation of an Rexx array. The program pulls the URL address of a syndication feed and adds it to the array. This process is repeated until the user types in the word “end”. This is used as key to tell the program to end with the insertion into the array. All the inserted addresses will later be aggregated into a single feed regardless of which type they are.

ooRexx example 9	FeedAggregator.rxj

```
/*          this creates an array of feed urls which will later be aggregated */
tmpColl = .array ~new
i = 0
do until url = "end"
PARSE PULL url
if url = "end"
then
say over
else
do
i = i + 1
say i
tmpColl[i] = url
end
end

say "your urls are :"
SAY tmpColl~string || ":"
DO item OVER tmpColl
SAY "[" || item || "]"
END
```

code 57) Creating a ooRexx array with URL entries

The next part sets up a new *SyndFeedImpl* i.e. the independent object model from PR. First we instantiate the *com.sun.syndication.feed.synd.SyndFeedImpl* class to create the new object model. Then we proceed with adding some basic information to the feed. This is not necessary regarding the aggregation but enables to provide further information. There are several methods available to set up the feed individually. In this example just a few are used, like ~setTitle, ~setAuthor, … . The last two lines are essential for the aggregation process. In the first one we create a *java.util.ArrayList* which can be added to the feed as an entry via ~setEntries(entries). The important step here is that the aggregated information is added to this Java array. This process is shown in code58.

```
feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(typet)

feed~setTitle("Aggregated Feed")
feed~setDescription("Anonymous Aggregated Feed")
feed~setAuthor("Martin Stoppacher")
feed~setLink("http://martinstoppacher.com")
/*             create some entries for the feed by using the SyndFeed methods */

/*                        creates a Java array list for the entries of the feed*/
entries =.bsf~new("java.util.ArrayList")

/*                                  sets the Java array as entries of the feed */
feed~setEntries(entries)
```

code 58) Creating a SyndFeedImpl and use its methods (BSF4ooRexx)

Finally we use the prior mentioned *SyndFeedInput* process in a loop to parse all the before added feeds from our Rexx array. The loop runs until the Rexx array is empty. In each round one inserted address is parsed to a *SyndFeed* and converted via the ~getEntries() method. This method returns all entries of the parsed feed. Finally the entries symbolized through infeed get added to the prior mentioned Java array symbolized through entries.

```
/* this uses the URL array to retrieve the entries fro the feeds              */
/* and add them to the entries array                                          */
do y=i to 1 by -1
feedUrl=.bsf~new("java.net.URL",tmpColl~at(i))
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
infeed=input~build(xmlr)
entries~addAll(infeed~getEntries())
end
```

code 59) Parse URLs from an array and get there entries (BSF4ooRexx)

6.10 ooRexx example 10 - Creating a feed

Until now we used parsed syndication feeds in most cases. In contrast we are creating a new feed in this example. Before we start with PR we have to add some other features. We start with the creation of a *java.text.SimpleDateFormat* because we will need it later for the creation process. Further we create something like a converter to use the Rexx function date().

ooRexx example 10	FeedCreator.rxj

```
/*        create a java  SimpleDateFormat instance for entry~setPublishedDate */
DATE_PARSER = .bsf~new("java.text.SimpleDateFormat", "yyyy-MMM-dd")

datum = date()
parse var datum day mounth year
if mounth="Dec" then mounth=12
if mounth="Jan" then mounth=01
if mounth="Feb" then mounth=01              /*  simply add the necessary once */
datum_new = year"-"mounth"-"day
say "Todays date: " datum_new
```

code 60) Create a Java SimpleDateFormat (BSF4ooRexx)

The next step would be the creation of a *SyndFeedImpl* like in example 9. This will not be repeated here. After that the *java.util.ArrayList* object gets created which is used as a container for the feeds entries. One entry is created by using the *SyndEntryImpl* class and its information can be set by using the corresponding methods. Moreover the *SyndEntryImpl* itself allows to add objects to it. E.g. the *SyndContentImpl* or the *SyndCategoryImpl* like it is shown in the examples below.

```
/*                        creates a Java array list for the entries of the feed*/
entries =.bsf~new("java.util.ArrayList")
entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")


/*                     create a independent description content implemetation */
description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue("ROME Web Page")
entry~setDescription(description)
/*                               add the description object to the entry object */
```

code 61) Creating an entry object and add a description object to it (BSF4ooRexx)

```
/* create a new category; by adding the SyndCategoryImpl to a Java array      */
/* object and  this array object to the entry object                          */
cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName("Java")
category~setTaxonomyUri("https://rome.dev.java.net/")
cat~add(category)
entry~setCategories(cat)
```

code 62) Create a category object and add it to an entry(BSF4ooRexx)

```
im=.bsf~new("com.sun.syndication.feed.synd.SyndImageImpl")
im~setTitle("A new image")
im~setLink("http://www.freefoto.com/images/14/26/14_26_4---Trevi-Fountain--
Rome--Italy_web.jpg")
im~setDescription("Trevi Fountain Rome")
im~setUrl("http://www.freefoto.com/images/14/26/14_26_4---Trevi-Fountain--Rome--
Italy_web.jpg")

feed~setImage(im)
```

code 63) Create an image object and add it to a Feed(BSF4ooRexx)

```
auth =.bsf~new("java.util.ArrayList")    /* a Java array to do setAuthors     */
/*                                  Using the SyndPersonImpl to add a person     */
pers=.bsf~new("com.sun.syndication.feed.synd.SyndPersonImpl")
pers~setName("Martin Stoppacher")
pers~setUri("martinstoppacher.com")
pers~setEmail("office@artinstoppacher.com")
auth~add(pers)

pers2=.bsf~new("com.sun.syndication.feed.synd.SyndPersonImpl")
pers2~setName("A second person")
pers2~setUri("the Uri of the person")
pers2~setEmail("the persons email")
auth~add(pers2)

feed~setAuthors(auth)
```

code 64) Create a Java array and add person objects to it, add the array to the Feed

Finally we add the whole entry to our Java array List using the ~add(entry) method and this List array to the prior created *SyndFeedImpl*. Obviously this demonstration deals only with one entry but nevertheless there can be more entries added to the Java array List.

```
/*                               add the entry to the Java entries array object */
entries~add(entry)

feed~setEntries(entries)
/*                                  sets the Java array as entries of the feed */
```

code 65) Add an entry to a Java array and this to the feed(BSF4ooRexx)

6.11 ooRexx example 11 - Graphical feed creator

The last example is a Feed Creator using the Java Swing classes in combination with the PR classes to create syndication feeds in various types. The independent (*SyndFeedImpl*) object model is used in this example the set up a type independent model of a feed. This allows us to create syndication feeds in all the available variants. The program allows to set up the main information of the feed (as it is in Rss " the channel ") and further to add entries to this information (e.g. represented as "items" in Rss). In addition it is possible to aggregate further information by adding a URL of a feed and add the entries from that feed to the new one.

The graphic demonstrates the resulting GUI and an example output in the windows shell. The full code of this program can be found in the Appendix B.

ooRexx example 11	Graphical_FeedCreator.rxj

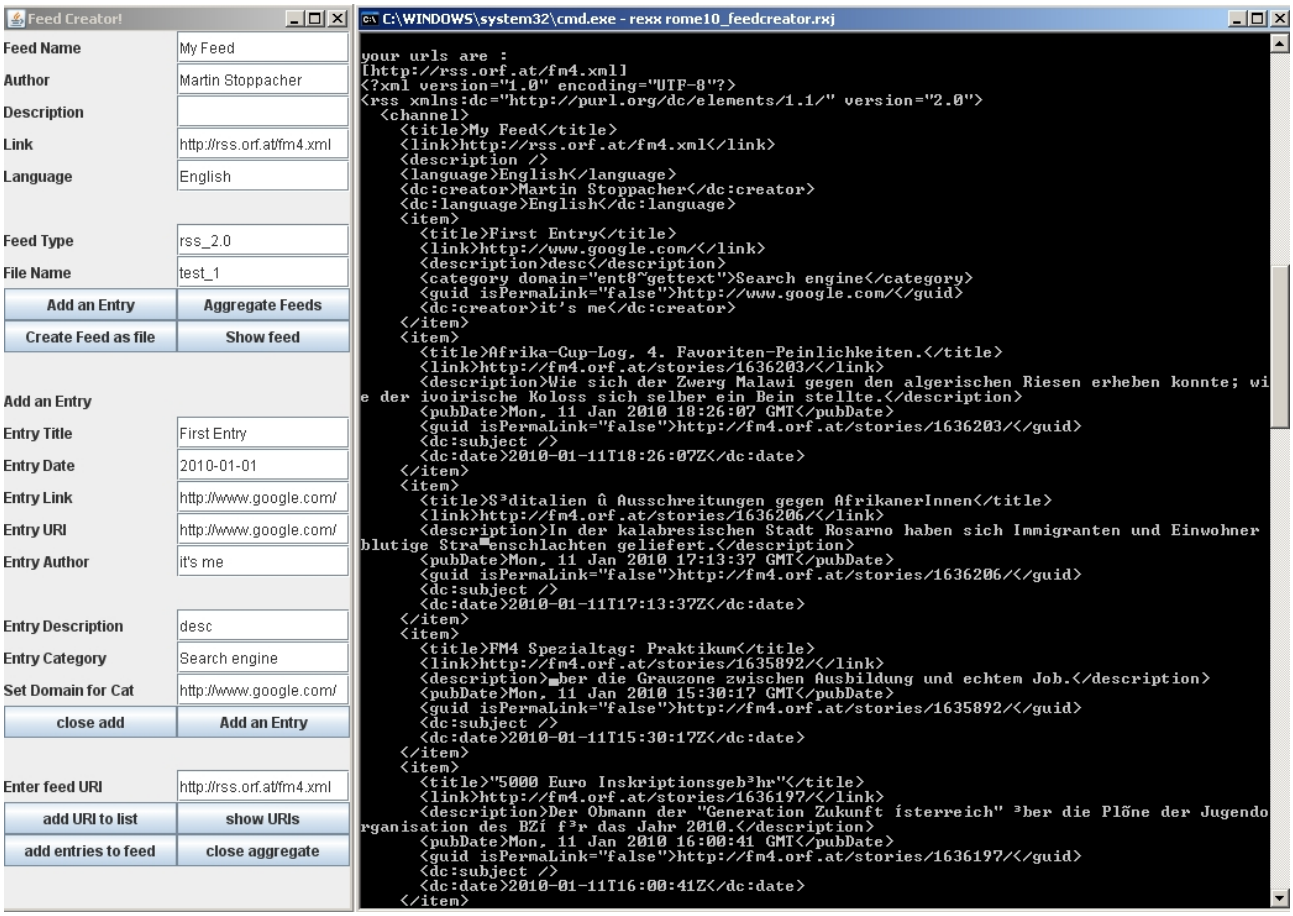

Figure 38) Graphical Feed Creator (using BSF4ooRexx)

This chapter depicted an overview of the most important features of Project Rome. The full program code of all the examples is available in the Appendix B.

7.0 Conclusion and Roundup

Finally, to make a short roundup, this chapter provides a short conclusion and discussion about the covered topics.

The paper began with the description of content syndication and its processing with the Project Rome Java library. As content syndication is a highly interesting topic in the actual media driven world, its processing get more and more important. The processing took place with the PR Java library and was extended with the usage of ooRexx and its extension BSF4ooRexx to provide the usage of Java features (classes) to ooRexx.
Beside the still mentioned advantage of universal execution, ... Rexx according to [Prech_00] needs just half as much code as programming in C++, ... Furthermore it is only half as much time consuming. Of course it is not the fastest language due to its interpreted paradigm but the actual speed of computer hardware leverages this deficit.
With BSF and its Rexx specific implementation BSF4ooRexx we are able to add a high rate of power to the Rexx language. By using the Java library with ooRexx we occupy the hole power of an other languages programming library. The previous mentioned survey [Prech_00] compared seven programming language with each other. Rexx was compared in its classical form. Regarding this paper it would be interesting to make a comparison between Java and ooRexx with its BSF extension.

Project Rome is a powerful Java library for processing syndication feeds in all available standards. PR provides an easy to use data model and helps developers to fasten up the creation process of there programs. Many companies and developers switched from there home made parsers to PR to employ its functionalities.
Currently the library is used within several open-source programs, commercial programs and websites. The range of its usage is very high. Starting with simple news feed reader programs PR can also be used to create simple reports in the XML format. Regarding the supported types PR gets permanently extended in the way of using its modules. E.g. there is also a support for OPML (1) in PR.

---

1) OPML – Outline Processor Markup Language – A XML format for outlines

The following is a list of possible areas where PR can be used to set up or improve facilities.

- News distribution
- Reports
- Wikis
- Blogs
- Trading software
- Search engines
- News collection

Much more are possible and there is no constraint in creativity. Like PR is expanding the features of developers we expanded two two areas in this paper. First we expanded the application area of PR due to its usage with ooRexx and second we widened a programmers possibilities because they are no longer forced to Java to implement the PR API within there projects.

8.0 Outlook

The interaction between the described programming languages and projects and all its including and related topic create an interesting and multi-faced field of research. The paper denoted the basic concepts of its related topics to provide an introduction in the field of content syndication processing. One interesting point mentioned is that we are no longer forced to a specific language like Java. Via Bean Scripting we are able to use the full Java library and with the ongoing research process of BSF4ooRexx more and more shortcomings of the framework get solved and more and more extensions are available.

As we saw in the previous chapter there is a wide range of application areas and projects already using the Project Rome library for content syndication. The movement towards ooRexx and its extension BSF4ooRexx opens new doors in the way of dealing with contend syndication. Now we have the power of one of the biggest application libraries in the hand of a scripting language and therefore the creation of code and its distribution gets easy. Some critics might say that interpreting is not fast enough but don't underestimate the speed of current hardware. Today we face machines with an incredible performance compared with computers from a few decades ago.

Will everything be interpreted in the future? This is a reasonable question because due to the ongoing performance development of hardware we are no longer forced to compiled software. Therefore Java's credo “Write once run everywhere” might get a small extension like “Code once run everywhere” as it is with ooRexx. We don't need to compile byte-code anymore and can take immediate advantage of the programs code and - depending on some rights - change whatever we want.

Keeping this in mind we are looking forward to an interlacing future where common borders start blurring more and more.

Appendix A Java coded Programs

The Appendix includes all created sample programs with its full code. The description includes three informations. First, a short description of the program functionality. Second, a file name which is also shown in the snippet description within the main part of the paper. Third, the number of the corresponding code snippet within the paper denoted by the code number out of the table of codes.

prog. nb.:	*description*
program 1	A simple Feed Reader that parses the information; FeadReader.java; code(8)

```
/*      ************************************************************      */
/* Example of a syndication feed reader using the Project Rome API        */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;    */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar,    you can find this at          https://jdom.org/           */
/* Parts of this example are taken from the PRome Web Page tutorials      */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur       */
/*                                                                        */
/* This class retrieves a syndfeed from the web and prints its            */
/* pure content to the system          used java version: 1.4.2; 1.6;     */
/* created by Martin Stoppacher        date:  26.12.2009                  */
/* license:      LGPL 3.0                                                 */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/*      ************************************************************      */
import java.net.URL;
/* Class URL represents a Uniform Resource Locator, a pointer to a "resource" */
/* on the World Wide Web                                                  */
import java.io.InputStreamReader;
/* An InputStreamReader is a bridge from byte streams to character streams    */
import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                     */
import com.sun.syndication.io.SyndFeedInput;
/*Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C */
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).            */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within    */
/* the stream.                                                            */
public class FeedReader {

        public static void main(String[] args) {
                boolean ok = false;
                if (args.length==1) {
                try {
                URL feedUrl = new URL(args[0]);/*creates a string with the URL*/

                SyndFeedInput input = new SyndFeedInput(); /* new input object*/
                SyndFeed feed = input.build(new XmlReader(feedUrl));
/* reads feed from URL and puts it into feed                              */

                System.out.println(feed);   /*  outputs file to system        */
/* this is just a pure output of the xml data                             */
                ok = true;
        }
        catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
```

```
                }
        }
        if (!ok) {
                System.out.println();
                System.out.println("FeedReader reads and prints any RSS/Atom
feed type.");
                System.out.println("The first parameter must be the URL of”
                                        +” the feed to read.");
                System.out.println();
        }
    }
}
```

program 2	A Feed Reader with the functionality to sava the feed in a file; FeedReader_with_writer.java; code(9)

```
/*      ************************************************************    */
/* Example of a syndication feed reader using the Project Rome API      */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;  */
/* You need to implement this API plus the JDOM API                     */
/* jdom.jar,    you can find this at           https://jdom.org/        */
/* Parts of this example are taken from the PRome Web Page tutorials    */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur     */
/*                                                                      */
/* This class prints a given syndfeed document to a file on your HD     */
/*                                    used java version: 1.4.2; 1.6;    */
/* created by Martin Stoppacher       date:  26.12.2009                 */
/* license:      LGPL 3.0                                               */
/*              (Lesser Gnu Public License version 3.0),                */
/*              cf. <http://www.gnu.org/licenses/lgpl.html>             */
/*      ************************************************************    */
import java.net.URL;
/*  Class URL represents a Uniform Resource Locator,                    */
/*  a pointer to a "resource" on the World Wide Web                     */
import java.io.FileWriter;
/*  class for writing character files                                   */
import java.io.InputStreamReader;
/*  An InputStreamReader is a bridge from byte streams to character streams   */
import java.io.Writer;
/* Abstract class for writing to character streams                      */
import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                   */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).          */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,        */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl             */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within  */
/* the stream.                                                          */

public class FeedReader2 {

    public static void main(String[] args) {
        String feedType = "http://rss.orf.at/fm4.xml";
        //String feedType = args[1];  /* e.g. you can also use the args array */
          String outputType = "rss_2.0";
        //String outputType = args[0]; /*e.g. you can also use the args array */
          String fileName = "test";
        boolean ok = false;
            try {
                URL feedUrl = new URL(feedType);

                SyndFeedInput input = new SyndFeedInput();
                SyndFeed feed = input.build(new XmlReader(feedUrl));
                //feed.setFeedType(outputType);    /*  optional converter     */
```

```
                // System.out.println(feed);       /* optional system out     */

                Writer writer = new FileWriter(fileName);
                /*     a new writer Object with specified file name           */
                SyndFeedOutput output = new SyndFeedOutput();
                /*     new output Object                                      */
                output.output(feed,writer);   /* outputs the feed to the file */
                writer.close();

                System.out.println("The feed has been written"
                                   +" to the file ["+fileName+"]");

                ok = true;
            }
            catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
            }
        if (!ok) {
            System.out.println();
            System.out.println("FeedReader reads and prints
                               +" any RSS/Atom feed type.");
            System.out.println("The first parameter must be
                               +" the URL of the feed to read.");
            System.out.println();
        }
    }
}
```

program 3	Feed Reader that outputs the feed to the system; FeedReader_with_output.java; code(10)

```
/*      ************************************************************    */
/* Example of a syndication feed reader using the Project Rome API      */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;  */
/* You need to implement this API plus the JDOM API                     */
/* jdom.jar,    you can find this at           https://jdom.org/        */
/* Parts of this example are taken from the PRome Web Page tutorials    */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur     */
/*                                                                      */
/* This class prints a retrieved feed with its tree structure to the system  */
/*                                    used java version: 1.4.2; 1.6;    */
/* created by Martin Stoppacher          date:  26.12.2009              */
/* license:      LGPL 3.0                                               */
/*              (Lesser Gnu Public License version 3.0),                */
/*              cf. <http://www.gnu.org/licenses/lgpl.html>             */
/*      ************************************************************    */
import java.net.URL;
/*  Class URL represents a Uniform Resource Locator,                    */
/*  a pointer to a "resource" on the World Wide Web                     */
import java.io.InputStreamReader;
/*  An InputStreamReader is a bridge from byte streams to character streams  */
import java.io.Writer;
/* Abstract class for writing to character streams                      */
import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                   */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).          */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,        */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl             */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within  */
/* the stream.                                                          */
```

```
public class FeedReader3 {

    public static void main(String[] args) {
        String feedType = "http://rss.orf.at/fm4.xml";
        //String feedType = args[1];  /* e.g. you can also use the args array */
        String outputType = "rss_2.0";
        //String outputType = args[0];/* e.g. you can also use the args array */
        boolean ok = false;
            try {
                URL feedUrl = new URL(feedType);

                SyndFeedInput input = new SyndFeedInput();
                SyndFeed feed = input.build(new XmlReader(feedUrl));
                feed.setFeedType(outputType);
                /*    feed converter, set the type to the chosen outputType   */

                SyndFeedOutput output = new SyndFeedOutput();
                /*    Tree Output of the feed                                 */
                output.output(feed,new PrintWriter(System.out));

                //System.out.println(feed);

                ok = true;
            }
            catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
            }
        if (!ok) {
            System.out.println();
            System.out.println("FeedReader reads and prints"
                                                +"any RSS/Atom feed type.");
            System.out.println();
        }
    }
}
```

program 4	Program supplying different output with different types; FeedReader_combination.java; code(11)

```
/*      ****************************************************************    */
/* Example of a syndication feed reader using the Project Rome API          */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;      */
/* You need to implement this API plus the JDOM API                         */
/* jdom.jar,    you can find this at           https://jdom.org/            */
/* Parts of this example are taken from the PRome Web Page tutorials        */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur         */
/*                                                                          */
/* Prints a retrieved feed in its pure and tree format to the system and    */
/* saves to files, one in the retrieved format and one in the converted format*/
/*                                    used java version: 1.4.2; 1.6;        */
/* created by Martin Stoppacher          date:  26.12.2009                  */
/* license:      LGPL 3.0                                                   */
/*               (Lesser Gnu Public License version 3.0),                   */
/*               cf. <http://www.gnu.org/licenses/lgpl.html>                */
/*      ****************************************************************    */
import java.net.URL;
/*  Class URL represents a Uniform Resource Locator,                        */
/*  a pointer to a "resource" on the World Wide Web                         */
import java.io.FileWriter;
/*class for writing character files*/
import java.io.InputStreamReader;
/*  An InputStreamReader is a bridge from byte streams to character streams */
import java.io.PrintWriter;
/* Print formatted representations of objects to a text-output stream       */
import java.io.Writer;
/* Abstract class for writing to character streams                          */
```

```
import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                    */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).           */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,         */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl              */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within   */
/* the stream.                                                           */

public class FeedReader4 {

    public static void main(String[] args) {
        boolean ok = false;
            try {
                String feedType = "http://rss.orf.at/fm4.xml";
                  String outputType = "rss_2.0";
                  String fileName = "test2_file_with_specified_output";
                  String fileName2 = "test3_original_source_file";
                  //String feedType = args[0]; /*creates a string with the URL*/
                //String fileName = args[1];
                /*                creates a string with the filename to write */
                URL feedUrl = new URL(feedType);

                SyndFeedInput input = new SyndFeedInput();/* new input object */
                SyndFeed feed = input.build(new XmlReader(feedUrl));
                /*                    reads feed from URL and puts it into feed*/
                feed.setFeedType(outputType);

                Writer writer = new FileWriter(fileName); /* new writer object*/
                SyndFeedOutput output = new SyndFeedOutput();
                /*                                          new output object*/
                output.output(feed,writer);
                /*                             outout method, writes feed to file*/
                writer.close();

                SyndFeedInput input2 = new SyndFeedInput(); /*new input object*/
                SyndFeed feed2 = input2.build(new XmlReader(feedUrl));
                /*                    reads feed from URL and puts it into feed*/


                Writer writer2 = new FileWriter(fileName2);
                /*                                        second writer object*/
                SyndFeedOutput output2 = new SyndFeedOutput();
                /*                                        second output object*/
                output2.output(feed2,writer2);
                /*                    second outout method, writes feed to file*/
                writer.close();

                System.out.println("The feed output with the
                                   +" Print Writer Class");
                System.out.println("___________");
                output.output(feed,new PrintWriter(System.out));

                System.out.println("The feed outputed by the system");
                System.out.println("___________");
                System.out.println(feed);              /*outputs file to system*/
                System.out.println("___________");
                System.out.println("The feed has been written
                                  +" to the file ["+fileName+"]");
                System.out.println("The feed has been written
                                  +" to the file ["+fileName2+"]");
                ok = true;
            }
            catch (Exception ex) {
```

```
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
            }
        if (!ok) {
            System.out.println();
            System.out.println("FeedReader reads and prints any
                               +" RSS/Atom feed type.");
            System.out.println("The first parameter must be the
                               +" URL of the feed to read.");
            System.out.println();
        }
    }
}
```

program 5	Program using the independent data model and allows to define the type of the feed; FeedConverter.java; code(12)

```
/*      ************************************************************    */
/* Example of a syndication feed reader using the Project Rome API      */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;  */
/* You need to implement this API plus the JDOM API                     */
/* jdom.jar,    you can find this at           https://jdom.org/        */
/* Parts of this example are taken from the PRome Web Page tutorials    */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur     */
/*                                                                      */
/* This program converts any syndication feed into a selected type      */
/* of feed, prints it to the screen and stores it to a file.            */
/*                                    used java version: 1.4.2; 1.6;    */
/* created by Martin Stoppacher       date:  26.12.2009                 */
/* license:    LGPL 3.0                                                 */
/*             (Lesser Gnu Public License version 3.0),                 */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>              */
/* ******************************************************************** */
import java.net.URL;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.Writer;

import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                   */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).          */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,        */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl             */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within  */
/* the stream.                                                          */

public class FeedConverter {

    public static void main(String[] args) {
        boolean ok = false;
        if (args.length==1) {
            try {
                String outputType = args[0];
                String fileName = outputType;
                URL feedUrl = new URL("http://rss.orf.at/fm4.xml");

                SyndFeedInput input = new SyndFeedInput();
                SyndFeed feed = input.build(new XmlReader(feedUrl));

                feed.setFeedType(outputType);               /*         this
line converts the feed into a specific format */
```

```
                SyndFeedOutput output = new SyndFeedOutput();
                output.output(feed,new PrintWriter(System.out));
                /*                                      // optional writer part
                Writer writer = new FileWriter(fileName);
                SyndFeedOutput output1 = new SyndFeedOutput();
                output1.output(feed,writer);
                writer.close();
                */
                ok = true;
            }
            catch (Exception ex) {
                System.out.println("ERROR: "+ex.getMessage());
            }
        }
        if (!ok) {
            System.out.println();
            System.out.println("FeedConverter converts between syndication"
                               +" feeds types.");
            System.out.println("The first parameter must be the feed type to"
                               +" convert to.");
            System.out.println(" [valid values are: rss_0.9, rss_0.91,"
                               +" rss_0.92, rss_0.93, ]");
            System.out.println(" [                  rss_0.94, rss_1.0,"
                               +" rss_2.0 & atom_0.3  ]");
            System.out.println();
        }
    }
}
```

program 6	This aggregates several feeds to a single one; FeedAggregator.java; code(13)

```
/*      ***************************************************************     */
/* Example of a syndication feed reader using the Project Rome API           */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;       */
/* You need to implement this API plus the JDOM API                          */
/* jdom.jar,    you can find this at           https://jdom.org/             */
/* Parts of this example are taken from the PRome Web Page tutorials         */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur          */
/*                                                                           */
/* This program aggregates different syndication feeds from                  */
/* different web resources and joins them into one file.                     */
/*                                    used java version: 1.4.2; 1.6;         */
/* created by Martin Stoppacher       date:  26.12.2009                      */
/* license:    LGPL 3.0                                                      */
/*             (Lesser Gnu Public License version 3.0),                      */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                   */
/* ***************************************************************************/
import java.net.URL;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.Writer;
import java.util.List;
import java.util.ArrayList;

import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                        */
import com.sun.syndication.feed.synd.SyndFeedImpl;
/* This is a Bean for all types of feeds                                     */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,             */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl                  */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).               */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary */
/* Voodo to figure out the charset encoding of the XML document within       */
/* the stream.                                                               */
```

```
public class FeedAggregator {

    public static void main(String[] args) {
        boolean ok = false;
        if (args.length>=2) {                   /* one or more feed adresses */
            try {
                String outputType = args[0];
                /*                          fist argument must be the feed type */
                String fileName = "test4_"+outputType;

                SyndFeed feed = new SyndFeedImpl();
                /*                              creates a new syndfeed object */
                feed.setFeedType(outputType); /* specifies which type of feed */

                feed.setTitle("Aggregated Feed"); /*specification of the feed */
                feed.setDescription("Anonymous Aggregated Feed");  /*  *****  */
                feed.setAuthor("Martin Stoppacher");               /*  *****  */
                feed.setLink("http://martinstoppacher.com");       /*  *****  */

                List entries = new ArrayList(); /* Creates an enties Object   */
                feed.setEntries(entries);
                /*                   connects the entries Object to the feed  */

                for (int i=1;i<args.length;i++) { /* loop to insert all feeds */
                    URL inputUrl = new URL(args[i]);

                    SyndFeedInput input = new SyndFeedInput();
                    SyndFeed inFeed = input.build(new XmlReader(inputUrl));

                    entries.addAll(inFeed.getEntries());
                    /* add the entries of the fetched "infeed"                */
                    /* to the created feed                                    */
                }
                SyndFeedOutput output = new SyndFeedOutput();
                output.output(feed,new PrintWriter(System.out));

                /*                                          // optional writer part
                Writer writer = new FileWriter(fileName);
                SyndFeedOutput output1 = new SyndFeedOutput();
                output1.output(feed,writer);
                writer.close();
                */
                ok = true;
            }
            catch (Exception ex) {
                System.out.println("ERROR: "+ex.getMessage());
            }
        }
        if (!ok) {
            System.out.println();
            System.out.println("FeedAggregator aggregates different feeds"
                               +" into a single one.");
            System.out.println("The first parameter must be the feed type"
                               +" for the aggregated feed.");
            System.out.println(" [valid values are: rss_0.9, rss_0.91U,"
                               +" rss_0.91N, rss_0.92, rss_0.93, ]");
            System.out.println(" [                  rss_0.94, rss_1.0,"
                               +" rss_2.0 & atom_0.3  ]");
            System.out.println("The second to last parameters are the"
                               +" URLs of feeds to aggregate.");
            System.out.println();
        }
    }
}
```

program 7	This adds a few features to program 6; FeedAggregator2.java; code(14)

```
/*      ***************************************************************     */
/* Example of a syndication feed reader using the Project Rome API          */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;      */
/* You need to implement this API plus the JDOM API                         */
/* jdom.jar,    you can find this at           https://jdom.org/            */
/* Parts of this example are taken from the PRome Web Page tutorials        */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur         */
/*                                                                          */
/* This program aggregates different syndication feeds from                 */
/* different web resources and joins them into one file.                    */
/* The retrieved resources are shown in the description part                */
/* of the created feed                                                      */
/*                                    used java version: 1.4.2; 1.6;        */
/* created by Martin Stoppacher       date:  26.12.2009                     */
/* license:    LGPL 3.0                                                     */
/*             (Lesser Gnu Public License version 3.0),                     */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                  */
/* ************************************************************************ */
import java.net.URL;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.Writer;
import java.util.List;
import java.util.ArrayList;

import com.sun.syndication.feed.synd.SyndFeed;
/* This is the Bean interface for all types of feeds.                       */
import com.sun.syndication.feed.synd.SyndFeedImpl;
/* This is a Bean for all types of feeds                                    */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,            */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl                 */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).              */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within      */
/* the stream.                                                              */
public class FeedAggregator2 {

    public static void main(String[] args) {
        boolean ok = false;
        if (args.length>=2) {                   /* one or more feed adresses */
            try {
                String outputType = args[0];
                /*                          fist argument must be the feed type */
                String fileName = "test4_"+outputType;
                String sources = args[1];  /* Converts the Array to a String  */
                String sep = "_";                                    /*  ***  */
                for (int i=2;i<args.length;i++) {                    /*  ***  */
                    sources = sources + sep + args[i];               /*  ***  */
                }
                SyndFeed feed = new SyndFeedImpl();
                /*                                creates a new syndfeed object */
                feed.setFeedType(outputType); /* specifies which type of feed */

                feed.setTitle("Aggregated_Feed_with the type_"+outputType);
                /*                                    specification of the feed */
                feed.setDescription(sources);                        /*  ***  */
                feed.setAuthor("Martin Stoppacher");                 /*  ***  */
                feed.setLink("http://martinstoppacher.com");         /*  ***  */

                List entries = new ArrayList();   /* Creates an enties Object */
```

```
                feed.setEntries(entries);
                /*                      connects the entries Object to the feed */

                for (int i=1;i<args.length;i++) { /* loop to insert all feeds */
                    URL inputUrl = new URL(args[i]);

                    SyndFeedInput input = new SyndFeedInput();
                    SyndFeed inFeed = input.build(new XmlReader(inputUrl));

                    entries.addAll(inFeed.getEntries());
                /*add the entries of the fetched "infeed" to the created feed */
                }
                SyndFeedOutput output = new SyndFeedOutput();
                output.output(feed,new PrintWriter(System.out));

                /*                                          // optional writer part
                Writer writer = new FileWriter(fileName);
                SyndFeedOutput output1 = new SyndFeedOutput();
                output1.output(feed,writer);
                writer.close();
                          */
                    ok = true;
            }
            catch (Exception ex) {
                System.out.println("ERROR: "+ex.getMessage());
            }
        }
        if (!ok) {
            System.out.println();
            System.out.println("FeedAggregator aggregates different feeds"
                               +" into a single one.");
            System.out.println("The first parameter must be the feed type"
                               +" for the aggregated feed.");
            System.out.println(" [valid values are: rss_0.9, rss_0.91U,"
                               +" rss_0.91N, rss_0.92, rss_0.93, ]");
            System.out.println(" [                  rss_0.94, rss_1.0,"
                               +" rss_2.0 & atom_0.3  ]");
            System.out.println("The second to last parameters are the"
                               +" URLs of feeds to aggregate.");
            System.out.println();
        }
    }
}
```

program 8	This creates a syndication Feed and prints it to the screen and includes a additional writer part; FeedCreator.java; code(15)

```
/*      ****************************************************************    */
/* Example of a syndication feed reader using the Project Rome API          */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;      */
/* You need to implement this API plus the JDOM API                         */
/* jdom.jar,    you can find this at           https://jdom.org/            */
/* Parts of this example are taken from the PRome Web Page tutorials        */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur         */
/*                                                                          */
/* This code is used to create feeds in different formats.                  */
/* The created feed will be printed to the system.                          */
/*                                    used java version: 1.4.2; 1.6;        */
/* created by Martin Stoppacher       date:  26.12.2009                     */
/* license:    LGPL 3.0                                                     */
/*             (Lesser Gnu Public License version 3.0),                     */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                  */
/* ************************************************************************ */
import com.sun.syndication.feed.synd.*;
/* imports the whole package:(interfaces: Converter,SyndCategory,SyndContent  */
/* SyndEnclosure,SyndEntry,SyndFeed,SyndImage SyndLink,SyndPerson           */
/* Classes: SyndCategoryImpl, SyndContentImpl,
SyndEnclosureImpl                    */
```

```
/* SyndEntryImpl, SyndFeedImpl, SyndImageImpl, SyndLinkImpl SyndPersonImpl    */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,              */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl                   */

import java.io.FileWriter;
import java.io.Writer;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.io.PrintWriter;

public class FeedWriter {

    private static final DateFormat
                DATE_PARSER = new SimpleDateFormat("yyyy-MM-dd");

    public static void main(String[] args) {
        boolean ok = false;
        if (args.length==1) {
            try {
                String feedType = args[0];
                /*              creates a string with the filetype Information*/
                //String fileName = args[1];
                /*               creates a string with the filename to write */

                SyndFeed feed = new SyndFeedImpl();          /*new feed object*/
                feed.setFeedType(feedType);      /*define the type of the feed*/

                feed.setTitle("This is a simple Sample");
                feed.setLink("martinstoppacher.com");
                feed.setDescription("My favorite webpages; format:_"+feedType);

                List entries = new ArrayList();       /* entries array object */
                SyndEntry entry;
                SyndContent description;

                entry = new SyndEntryImpl();                 /* entry opbject  */
                entry.setTitle("The Project Rome Web Page");
                /*                                  shows feed type in the feed*/
                entry.setLink("https://rome.dev.java.net/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl(); /*    description object */
                description.setType("text/plain");
                description.setValue("ROME Web Page");
                entry.setDescription(description);
                /*             add the description object to the entry object */
                entries.add(entry);
                /*               adds the entry object to the entries array  */

                entry = new SyndEntryImpl();       /* an other entry opbject  */
                entry.setTitle("Java 1.4 Documentation");
                entry.setLink("http://java.sun.com/j2se/1.4.2/docs/api/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl();
                description.setType("text/plain");
                description.setValue("Documentation of the Java 1.4 API");
                entry.setDescription(description);
                entries.add(entry);

                entry = new SyndEntryImpl();        /* an other entry opbject */
                entry.setTitle("The ooRexx Web Page");
                entry.setLink("http://www.oorexx.org/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl();
                description.setType("text/html");
                description.setValue("<p>Open Object Rexx version 4.0.0 is now"
```

```
                                    +" available. Visit the announcement page"
                                    +" for More information.</p><p>For details"
                                    +" check the <a href=\"http://www.oo"
                                    +"rexx.org/\">ooRexx Web Page</a></p>");

                entry.setDescription(description);
                entries.add(entry);

                feed.setEntries(entries);
                /*                          add the entries array to the  feed */

                SyndFeedOutput output = new SyndFeedOutput();  /* putput part */
                output.output(feed,new PrintWriter(System.out));
                /*                                          // optional writer part
                Writer writer = new FileWriter(fileName);
                SyndFeedOutput output = new SyndFeedOutput();
                output.output(feed,writer);
                writer.close();

                System.out.println("The feed has been"
                                   +" written to the file ["+fileName+"]");
                */

                ok = true;
            }
            catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
            }
        }
        if (!ok) {
            System.out.println();
            System.out.println("FeedWriter creates a RSS/Atom feed and"
                               +" writes it to a file.");
            System.out.println("The first parameter must be the"
                               +" syndication format for the feed");
            System.out.println("  (rss_0.90, rss_0.91, rss_0.92, rss_0.93,"
                               +" rss_0.94, rss_1.0 rss_2.0 or atom_0.3)");
            System.out.println();
        }
    }

}
```

program 9	This adds the aggregation function to a Feed Creator; FeedWriter_with_Aggregator.java; code(16)

```
/*      ************************************************************    */
/* Example of a syndication feed reader using the Project Rome API      */
/* current version of Rome: rome1.0.jar    https://rome.dev.java.net/;  */
/* You need to implement this API plus the JDOM API                     */
/* jdom.jar,    you can find this at           https://jdom.org/        */
/* Parts of this example are taken from the PRome Web Page tutorials    */
/* https://rome.dev.java.net/; thanks to author  Alejandro Abdelnur     */
/*                                                                      */
/* This code is used to create feeds in different formats and           */
/* to add some other feeds from the web in this one                     */
/*                                    used java version: 1.4.2; 1.6;    */
/* created by Martin Stoppacher       date:  26.12.2009                 */
/* license:    LGPL 3.0                                                 */
/*             (Lesser Gnu Public License version 3.0),                 */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>              */
/* ******************************************************************** */
import com.sun.syndication.feed.synd.*;
/* imports the whole package:(interfaces: Converter,SyndCategory,SyndContent  */
/* SyndEnclosure,SyndEntry,SyndFeed,SyndImage SyndLink,SyndPerson       */
/* Classes: SyndCategoryImpl, SyndContentImpl,
SyndEnclosureImpl                    */
```

```
/* SyndEntryImpl, SyndFeedImpl, SyndImageImpl, SyndLinkImpl SyndPersonImpl    */
import com.sun.syndication.io.SyndFeedOutput;
/* Generates an XML document(String, File, OutputStream, Writer,              */
/* W3C DOM document or JDOM document)out of an SyndFeedImpl                   */
import com.sun.syndication.io.SyndFeedInput;
/* Parses an XML document (File, InputStream, Reader, W3C SAX InputSource, W3C*/
/* DOM Document or JDom DOcument) into an WireFeed (RSS/Atom).                */
import com.sun.syndication.io.XmlReader;
/* Character stream that handles (or at least attemtps to) all the necessary  */
/* Voodo to figure out the charset encoding of the XML document within        */
/* the stream.                                                                */

import java.io.PrintWriter;
import java.net.URL;
import java.io.FileWriter;
import java.io.Writer;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.io.PrintWriter;

public class FeedWriter2 {

    private static final DateFormat
                DATE_PARSER = new SimpleDateFormat("yyyy-MM-dd");

    public static void main(String[] args) {
        boolean ok = false;
        if (args.length>=1) {
            try {
                String feedType = args[0];
                /*              creates a string with the filetype Information*/
                //String fileName = args[1];
                /*                creates a string with the filename to write */

                SyndFeed feed = new SyndFeedImpl();          /*new feed object*/
                feed.setFeedType(feedType);      /*define the type of the feed*/

                feed.setTitle("This is a simple Sample");
                feed.setLink("martinstoppacher.com");
                feed.setDescription("My favorite webpages; format:_"+feedType);

                List entries = new ArrayList();       /* entries array object */
                SyndEntry entry;
                SyndContent description;

                entry = new SyndEntryImpl();                /* entry opbject  */
                entry.setTitle("The Project Rome Web Page");
                /*                                  shows feed type in the feed*/
                entry.setLink("https://rome.dev.java.net/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl(); /*    description object */
                description.setType("text/plain");
                description.setValue("ROME Web Page");
                entry.setDescription(description);
                /*              add the description object to the entry object */
                entries.add(entry);
                /*                adds the entry object to the entries array  */

                entry = new SyndEntryImpl();       /* an other entry opbject  */
                entry.setTitle("Java 1.4 Documentation");
                entry.setLink("http://java.sun.com/j2se/1.4.2/docs/api/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl();
                description.setType("text/plain");
                description.setValue("Documentation of the Java 1.4 API");
                entry.setDescription(description);
```

```
                entries.add(entry);

                entry = new SyndEntryImpl();        /* an other entry opbject */
                entry.setTitle("The ooRexx Web Page");
                entry.setLink("http://www.oorexx.org/");
                entry.setPublishedDate(DATE_PARSER.parse("2009-12-26"));
                description = new SyndContentImpl();
                description.setType("text/html");
                description.setValue("<p>Open Object Rexx version 4.0.0 is now"
                                     +" available. Visit the announcement page"
                                     +" for More information.</p><p>For details"
                                     +" check the <a href=\"http://www.oo"
                                     +"rexx.org/\">ooRexx Web Page</a></p>");

                entry.setDescription(description);
                entries.add(entry);

                feed.setEntries(entries);
                /*                          add the entries array to the  feed */

                for (int i=1;i<args.length;i++) { /* loop to insert all feeds */
                    URL inputUrl = new URL(args[i]);

                    SyndFeedInput input = new SyndFeedInput();
                    SyndFeed inFeed = input.build(new XmlReader(inputUrl));

                    entries.addAll(inFeed.getEntries());
                /*add the entries of the fetched "infeed" to the created feed */

                }

                SyndFeedOutput output = new SyndFeedOutput();
                output.output(feed,new PrintWriter(System.out));

                Writer writer = new FileWriter(fileName);      /* writer part */
                SyndFeedOutput output2 = new SyndFeedOutput();
                output2.output(feed,writer);
                writer.close();

                System.out.println("The feed has been"
                                   +" written to the file ["+fileName+"]");

                ok = true;
            }
            catch (Exception ex) {
                ex.printStackTrace();
                System.out.println("ERROR: "+ex.getMessage());
            }
        }
        if (!ok) {
            System.out.println();
            System.out.println("FeedWriter creates a RSS/Atom feed and"
                               +" writes it to a file.");
            System.out.println("The first parameter must be the"
                               +" syndication format for the feed");
            System.out.println("  (rss_0.90, rss_0.91, rss_0.92, rss_0.93,"
                               +" rss_0.94, rss_1.0 rss_2.0 or atom_0.3)");
            System.out.println();
        }
    }
}
```

Appendix B ooRexx coded Programs using BSF4ooRexx

The Appendix includes all created sample programs with its full code. The description includes three informations. First, a short description of the program functionality. Second, a file name which is also shown in the snippet description within the main part of the paper. Third, the number of the corresponding code snippet within the paper denoted by the code number out of the table of codes.

program 1	Using a precreated Java class with ooRexx; Read.rxj; code(45)

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This class retrieves a syndfeed from the web by using a precreated     */
/* Java class via BSF4ooRexx                                              */
/* "com.sun.syndication.io.SyndFeedInput" methods                         */
/* created by Martin Stoppacher       date:  26.12.2009                   */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*            (Lesser Gnu Public License version 3.0),                    */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

javaclass = "FeedReader1"              /* determine Java class to use     */
get=.bsf~new(javaClass)                /* create an instance of "javaClass"  */
say get~getfeed         /* calls the getfeed method in the FeedReader1 class  */

::requires BSF.CLS                     /* get the BSF support             */
```

program 2	Setting up a feed parser; rome.rxj; code(46), (47), (48), (49)

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This class retrieves a syndfeed from the web and prints some           */
/* information about the feed using                                       */
/* "com.sun.syndication.io.SyndFeedInput" methods                         */
/* created by Martin Stoppacher     date:   26.12.2009                    */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

say hello this reads a syndfeed
--say please type in the url     /* possible input statement for the feed URL */
--pull url

--parse arg url   /*   optional input via arguments                       */

/* The url String is predefined in this example, to change the feed url   */
/* replace to String in this file or use one of the above methods         */

url= "http://rss.orf.at/fm4.xml"
```

```
feedUrl=.bsf~new("java.net.URL", url)   /* create an instance of java.net.URL */
say connecting__ || feedUrl~getAuthority()

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
/*  creates a SyndFeedInput                                                   */
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
/*     Figures out the charset encoding of the XML document within the stream */
feed= input~build(xmlr)
/*  Builds a SyndFeedImpl from an Reader , also possible with SAX or DOM      */

say bb~getDefaultEncoding()
say feed                              /* returns the feed object             */

::requires BSF.cls                    /* get the BSF support                 */
```

program 3	Using different methods from the Project Rome library to get information about a parsed feed; Rome_SyndFeed_methods.rxj; code(50)

```
/* ************************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with       */
/* BSF4ooRexx                                                                 */
/* current version of Rome: rome1.0.jar            https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                           */
/* jdom.jar  ,   you can find this at              https://jdom.org/_______; */
/*                                                                            */
/* This class retrieves a syndfeed from the web and prints some               */
/* information about the feed using                                           */
/* "com.sun.syndication.io.SyndFeedInput" methods                             */
/* created by Martin Stoppacher     date:   26.12.2009                        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                       */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                    */
/* ************************************************************************** */

say hello this reads a syndfeed
--say please type in the url
--pull url
/* The url String is predefined in this example, to change the feed url       */
/* replace to String in this file or use one of the above methods             */
url="http://rss.orf.at/fm4.xml"
feedUrl=.bsf~new("java.net.URL", url)   /* create an instance of java.net.URL
*/
say connecting__ || feedUrl~getAuthority()

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
/*                          Parses an XML document into an SyndFeedImpl */
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
/*     Figures out the charset encoding of the XML document within the stream */
feed= input~build(xmlr)
/*      Builds a SyndFeedImpl from an Reader , also possible with SAX or DOM  */
call syssleep(2)                                          /* sleep by Rexx */
say connected
say "----"
say "The Title of the feed is:"
say feed~getTitle()                              /* Returns the feed title */
sl = BSF("sleep",2)
say "----"
say "The URI of the feed is:"
say feed~getUri()                                 /* Returns the feed URI  */
sl = BSF("sleep",2)
say "----"
say "The Author of the feed is:"
say feed~getAuthor()
/*    Returns the name of the first--feed author in the collection of authors */
sl = BSF("sleep",2)
```

```
say "----"
say "The copyright of the feed is:"
say feed~getCopyright()                          /* Returns the feed copyright */
sl = BSF("sleep",2)
say "----"
say "The description of the feed is:"
say feed~getDescription()                      /* Returns the feed description */
sl = BSF("sleep",2)
say "----"
say "The encoding of the feed is:"
say feed~getEncoding()          /* Returns the charset encoding of a the feed */
say "----"

sl = BSF("sleep",2)                       /* sleep by using a BSF subfunktion */
say just a break;
sl = BSF("sleep",2)                       /* sleep by using a BSF subfunktion */

say "----"
aut= feed~getAuthors()           /* Returns the feed authors , java.util.List */
say aut
say this is the Authors object
say "----"
array = feed~getEntries()        /* Returns the feed entries , java.util.List */
say array
say "this is the entries-array object"

::requires BSF.cls                     /* get the BSF support                  */
```

program 4	Using the setFeedType method; FeedReader_with_setType.rxj; code(51)

```
/* ************************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with       */
/* BSF4ooRexx                                                                 */
/* current version of Rome: rome1.0.jar            https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                           */
/* jdom.jar  ,   you can find this at              https://jdom.org/_______; */
/*                                                                            */
/* This example shows how to set the type of a feed                           */
/* created by Martin Stoppacher     date:   26.12.2009                        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                       */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                    */
/* ************************************************************************** */

say hello this reads a syndfeed
say please type in the url
url="http://rss.orf.at/fm4.xml"

feedUrl=.bsf~new("java.net.URL", url)
say connecting__ || feedUrl~getAuthority()

--call Syssleep 1
sl = BSF("sleep",3)          /* BSF Subfunction referencing to the Java Bean */

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
feed=input~build(xmlr)

feed~setFeedType("rss_1.0")   /* This line allows to set the type of the feed */
/* the default formats in ROME are: rss_0.90, rss_0.91, rss_0.92,             */
/* rss_0.93, rss_0.94, rss_1.0 rss_2.0 or atom_0.3                            */

say feed~getFeedType()               /* This line retrieves the current type */
```

```
feed~setFeedType("rss_2.0")

say feed~getFeedType()

feed~setFeedType("atom_1.0")

say feed~getFeedType()

::requires BSF.cls                      /* get the BSF support                */
```

program 5	Using the parser with a graphical extension (Java AWT); FeedReader_awt_extension.rxj; code(52)

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This example extents the function via BSF4ooRexx by using Java AWT     */
/* to create a graphical user interface for inputting the feed URL        */
/* created by Martin Stoppacher     date:   26.12.2009                    */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */
/* graphical part of the eaxample                                         */
frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.FlowLayout") )

tf=.bsf~new("java.awt.TextField", "", 50)
frame~add(tf)

but=.bsf~new('java.awt.Button', 'retrieve feed')
frame~add(but)
but~bsf.addEventListener('action', '', ' call text tf ')

frame~~pack~~show~~toFront

do forever
INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave
end
exit

text: --procedure
use arg tf
say hello this reads a syndfeed

url= tf~getText                 /* you can use:  "http://rss.orf.at/fm4.xml"  */

/*   this part is using the ROME API                                       */

feedUrl=.bsf~new("java.net.URL", url)
say connecting__ || feedUrl~getAuthority()

call Syssleep 1

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
feed=input~build(xmlr)

--feed~setFeedType("rss_2.0")                  /* optional feed type setting */
```

```
say feed~getTitle()
say feed~getDescription()
say feed~getFeedType()

do forever
INTERPRET .bsf~bsf.pollEventText
end

::requires BSF.cls                      /* get the BSF support                */
```

program 6	Using the io classes of PR to output a parsed feed; FeedReader_FeedOutput.rxj; code(53)

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This class retrieves a Syndfeed from the web and converts it to a chosen   */
/* rss type. Finally the feed is printed to the system  (in combination with  */
/* a graphical interface)                                                 */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */
/* graphical part of the eaxample                                         */

frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.FlowLayout") )
tf=.bsf~new("java.awt.TextField", "http://rss.orf.at/fm4.xml", 50)
type=.bsf~new("java.awt.TextField", "rss_2.0", 50)
but=.bsf~new('java.awt.Button', 'retrieve feed')
but~bsf.addEventListener('action', '', ' call text tf,type ')
lable1=.bsf~new('java.awt.Label', "Feed URL:")
lable2=.bsf~new('java.awt.Label', "Feed Type:")

frame~add(lable1)
frame~add(tf)
frame~add(lable2)
frame~add(type)
frame~add(but)

frame~~pack~~show~~toFront
do forever
INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave
end
exit

text: --procedure
use arg tf,type
say hello this reads a syndfeed
--say please type in the url

url= tf~getText                 /* you can use:  "http://rss.orf.at/fm4.xml"  */
typet= type~getText

/*   this part is using the ROME API                                       */
```

```
feedUrl=.bsf~new("java.net.URL", url)
say connecting__ || feedUrl~getAuthority()

call Syssleep 1

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
feed=input~build(xmlr)

feed~setFeedType(typet)              /* This line retrieves the current type */
/* the default formats in ROME are: rss_0.90, rss_0.91, rss_0.92,            */
/* rss_0.93, rss_0.94, rss_1.0 rss_2.0 or atom_0.3                           */

output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
say output~outputString(feed)

say feed~getFeedType()
say "------"
say done

do forever
INTERPRET .bsf~bsf.pollEventText
end

::requires BSF.cls
```

program 7	Demonstrates the usage of Synd and WireFeed; SyndFeed_WireFeed.rxj; code(55);

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This class retrieves a syndfeed from the web and prints it to the system */
/* using the syndfeed Outout and the wirefeed Output                      */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

/* grafical part                                                          */
frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.FlowLayout") )

tf=.bsf~new("java.awt.TextField", "http://rss.orf.at/fm4.xml", 50)
type=.bsf~new("java.awt.TextField", "rss_2.0", 50)
but=.bsf~new('java.awt.Button', 'retrieve feed')
but~bsf.addEventListener('action', '', ' call text tf,type ')
lable1=.bsf~new('java.awt.Label', "Feed URL:")
lable2=.bsf~new('java.awt.Label', "Feed Type:")

frame~add(lable1)
frame~add(tf)
frame~add(lable2)
frame~add(type)
frame~add(but)

frame~~pack~~show~~toFront

do forever
```

```
INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave
end
exit

text: --procedure
use arg tf,type

say hello this reads a syndfeed
-- say please type in the url
-- pull url                                          /* optional url input */
--url="http://rss.orf.at/fm4.xml"
url = tf~getText

typet= type~getText                                /* optional rss type parts */
--pull typet

typet="rss_2.0"

feedUrl=.bsf~new("java.net.URL", url)               /* creating a URL object  */
say connecting__ || feedUrl~getAuthority()

call Syssleep 1

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
/*  creates a SyndFeedInput , an independent object model  instance           */
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
/*     Figures out the charset encoding of the XML document within the stream */

input~setXmlHealerOn(0)                                 /* set healer on off */
say input~getXmlHealerOn()                            /* checks for xml healer */

feed=input~build(xmlr)              /* Creates a SyndFeed out ouf the SFInput */
wfeed = feed~createWireFeed("rss_1.0")  /*  creates a WireFeed (abstract DM)  */
feed~setFeedType(typet)                /*  sets the feedtype of the SyndFeed  */

/*                      outputs from the SyndFeed, (independend object model) */
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
say output~outputString(feed,1)       /* usage of pritty print(0 yes or 1 no) */

say feed~getFeedType()
say "-----"

/*                           outputs from the WireFeed, (abstract data model)  */
output2=.bsf~new("com.sun.syndication.io.WireFeedOutput")
say output2~outputString(wfeed)

say wfeed~getFeedType()
say "------"
say done

/* grafical part                                                              */

do forever
INTERPRET .bsf~bsf.pollEventText
end

::requires BSF.cls
```

program 8	This program parses a feed and converts it in different ways using type conversion and different data models; FeedReader_Writer_combination.rxj; code(56)

```
/* ************************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with       */
```

```
/* BSF4ooRexx                                                          */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                    */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                     */
/* This class retrieves a syndfeed from the web and prints it to the system   */
/* using the syndfeed Outout and the wirefeed Output                   */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009     */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>             */
/* ******************************************************************* */

/* grafical part                                                       */

frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.FlowLayout") )

tf=.bsf~new("java.awt.TextField", "http://rss.orf.at/fm4.xml", 50)
type=.bsf~new("java.awt.TextField", "rss_2.0", 50)
but=.bsf~new('java.awt.Button', 'retrieve feed')
but~bsf.addEventListener('action', '', ' call text tf,type ')
lable1=.bsf~new('java.awt.Label', "Feed URL:")
lable2=.bsf~new('java.awt.Label', "Feed Type:")

frame~add(lable1)
frame~add(tf)
frame~add(lable2)
frame~add(type)
frame~add(but)

frame~~pack~~show~~toFront

do forever
INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave
end
exit

text: --procedure
use arg tf,type

say hello this reads a syndfeed
--say please type in the url
-- pull url                                        /* optional url input */
--url="http://rss.orf.at/fm4.xml"
url = tf~getText

typet= type~getText                               /* optional rss tape parts */
--pull typet
--typet="rss_2.0"

fileName2 = "test2_printed_from_syndfeed" typet
fileName3 = "test3_printed_from_wirefeed"

feedUrl=.bsf~new("java.net.URL", url)             /* ocreating a URL object  */
say connecting__ || feedUrl~getAuthority()

call Syssleep 1

input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
/*  creates a SyndFeedInput , an independent object model  instance          */
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
/*     Figures out the charset encoding of the XML document within the stream */
```

```
input~setXmlHealerOn(0)                                  /* set healer on off */
say input~getXmlHealerOn()                             /* checks for xml healer */

feed=input~build(xmlr)
fileName1 = "test1_original_syndfeed" feed~getFeedType()

/*                writer part: saves the feed file as XML code to the system */
/*                this writes the original feed to a file                    */
writer=.bsf~new("java.io.FileWriter",fileName1)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

say output~outputString(feed,0)       /* usage of pretty print(0 yes or 1 no) */

say feed~getFeedType()
say "-----"

feed~setFeedType(typet)

writer2=.bsf~new("java.io.FileWriter",fileName2)
output2=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output2~output(feed,writer2)
/*  output method of the SyndFeedOutput writes feed with the java file writer */
writer2~close()

say output2~outputString(feed,0) -- usage of pritty print(0 yes or 1 no)

say feed~getFeedType()
say "-----"

wfeed = feed~createWireFeed("rss_1.0")

writer3=.bsf~new("java.io.FileWriter",fileName3)
output3=.bsf~new("com.sun.syndication.io.WireFeedOutput")
output3~output(wfeed,writer3)     -- also possible with pretty print
writer3~close()

say output3~outputString(wfeed,0)  -- usage of pretty print(0 yes or 1 no)

say wfeed~getFeedType()
say "-----"
say done

/* Optional grafical part                                                    */

do forever
INTERPRET .bsf~bsf.pollEventText
end


::requires BSF.cls
```

program 9	This program allows to collect URIs of feeds and afterwards aggregates them to a single one; FeedAggregator.rxj; code(57)

```
/* ************************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with       */
/* BSF4ooRexx                                                                 */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                           */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                            */
/* This example retrieves several feeds from a URI and aggregates them        */
```

```
/* to a single one                                                          */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009          */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                     */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                  */
/* ************************************************************************ */

say hello this aggregates syndfeeds
say type in the urls _ to end type in _ end _
/*          this creates an array of feed urls which will later be aggregated */
tmpColl = .array ~new
i = 0
do until url = "end"
PARSE PULL url
if url = "end"
then
say over
else
do
i = i + 1
say i
tmpColl[i] = url
end
end

say "your urls are :"
SAY tmpColl~string || ":"
DO item OVER tmpColl
SAY "[" || item || "]"
END

/*                                                    optional type input */
--pull typet
typet="rss_2.0"
fileName = "test4_aggregated_feed" typet

call Syssleep 1
say 'Your crated feed starts hear :'

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(typet)
/*                                              sets the type of this feed */

feed~setTitle("Aggregated Feed")
feed~setDescription("Anonymous Aggregated Feed")
feed~setAuthor("Martin Stoppacher")
feed~setLink("http://martinstoppacher.com")
/*             create some entries for the feed by using the SyndFeed methods */

/*                       creates a Java array list for the entries of the feed*/
entries =.bsf~new("java.util.ArrayList")
/*                                  sets the Java array as entries of the feed */
feed~setEntries(entries)

/* this uses the URL array to retrieve the entries fro the feeds             */
/* and add them to the entries array                                         */
do y=i to 1 by -1
feedUrl=.bsf~new("java.net.URL",tmpColl~at(i))
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
infeed=input~build(xmlr)
entries~addAll(infeed~getEntries())
end
```

```
/*                                              output of the aggregated feeds */
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
say output~outputString(feed,1)     /* usage of pretty print(0 yes or 1 no)   */

/*                                                                  writer part */
writer=.bsf~new("java.io.FileWriter",fileName)
output~output(feed,writer)
writer~close()

PP : RETURN "[" || ARG(1)|| "]"
::requires BSF.cls
```

program 10	This example demonstrates the creation process of a syndication feed with Project ROME; FeedCreator.rxj; code(60)

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This is a simple example of how to create a syndication feed with ooRexx   */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

say "hello this creates a syndfeeds"
--pull typet  /* optional type input */
typet = "atom_0.3"
fileName = "test6_new_feed" typet

/*                             converting the rexx date() to an other format */
datum = date()
parse var datum day mounth year
if mounth="Dec" then mounth=12
if mounth="Jan" then mounth=01
if mounth="Feb" then mounth=01              /*  simply add the necessary once */
datum_new = year"-"mounth"-"day
say "Todays date: " datum_new

call Syssleep 1

say 'Your crated feed starts hear :'

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(typet)
/*                                              sets the type of this feed */

desc = "My favorite webpages; format:_" typet

feed~setTitle("This is a simple Sample")
feed~setDescription(desc)
feed~setAuthor("Martin Stoppacher")
feed~setLink("http://martinstoppacher.com")
feed~setLanguage("English")

/*        create a java  SimpleDateFormat instance for entry~setPublishedDate */
DATE_PARSER = .bsf~new("java.text.SimpleDateFormat", "yyyy-MM-dd")

/*                      creates a Java array list for the entries of the feed*/
```

```
entries =.bsf~new("java.util.ArrayList")
entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")

/*                                      add some entries to an entriy object */
entry~setTitle("The Project Rome Web Page")
entry~setPublishedDate(DATE_PARSER~parse(datum_new))
entry~setLink("https://rome.dev.java.net/")
entry~setUri("https://rome.dev.java.net/")
entry~setAuthor("dev.java.net")

/*                     create a independent description content implemetation */
description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue("ROME Web Page")
entry~setDescription(description)
/*                              add the description object to the entry object */

/* create a new category; by adding the SyndCategoryImpl to a Java array      */
/* object and  this array object to the entry object                          */
cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName("Java")
category~setTaxonomyUri("https://rome.dev.java.net/")
cat~add(category)
entry~setCategories(cat)

/*                              add the entry to the Java entries array object */
entries~add(entry)

/*                                                          a secound entry */
entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")

entry~setTitle("Java 1.4 Documentation")
entry~setPublishedDate(DATE_PARSER~parse(datum_new))
entry~setLink("http://java.sun.com/j2se/1.4.2/docs/api/")
entry~setUri("http://java.sun.com/")
entry~setAuthor("java.sun.com/")

description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue("Documentation of the Java 1.4 API")
entry~setDescription(description)

cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName("Documentation")
category~        setTaxonomyUri("http://java.sun.com/j2se/1.4.2/docs/api/")
cat~add(category)
entry~setCategories(cat)

entries~add(entry)

feed~setEntries(entries)
/*                                 sets the Java array as entries of the feed */

/*                                                              writer part */
writer=.bsf~new("java.io.FileWriter",fileName)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

say output~outputString(feed,1) -- usage of pretty print(0 yes or 1 no)

PP : RETURN "[" || ARG(1)|| "]"
```

```
::requires BSF.cls
```

program 11	This program allows to create syndication feeds with a GUI using the Java swing classe; Graphical_FeedCreator.rxj;

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This is an example of a graphical feed creator (using PR and Java Swing */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

/*                             converting the rexx date() to an other format */

datum = date()
parse var datum day mounth year
if mounth="Dec" then mounth=12
if mounth="Jan" then mounth=01             /*  simply add the necessary once */
datum_new = year"-"mounth"-"day

/*        create a java  SimpleDateFormat instance for entry~setPublishedDate */
DATE_PARSER = .bsf~new("java.text.SimpleDateFormat", "yyyy-MM-dd")

entries =.bsf~new("java.util.ArrayList")
i = 1

f=.bsf~new("javax.swing.JFrame", "Feed Creator!")
f~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
f~setLayout( .bsf~new("java.awt.GridLayout",0,2))

input1=.bsf~new("javax.swing.JTextField", "My Feed", 10)
input2=.bsf~new("javax.swing.JTextField", "Martin Stoppacher", 10)
input3=.bsf~new("javax.swing.JTextField", "", 10)
input4=.bsf~new("javax.swing.JTextField", "http://rss.orf.at/fm4.xml", 10)
input5=.bsf~new("javax.swing.JTextField", "English", 10)

inputtype=.bsf~new("javax.swing.JTextField", "rss_2.0", 10)
inputname=.bsf~new("javax.swing.JTextField", "test_1", 10)

label1=.bsf~new("javax.swing.JLabel", "Feed Name")
label2=.bsf~new("javax.swing.JLabel", "Author")
label3=.bsf~new("javax.swing.JLabel", "Description")
label4=.bsf~new("javax.swing.JLabel", "Link")
label5=.bsf~new("javax.swing.JLabel", "Language")

labeltype=.bsf~new("javax.swing.JLabel", "Feed Type")
labelname=.bsf~new("javax.swing.JLabel", "File Name")

label20=.bsf~new("javax.swing.JLabel", "")
label21=.bsf~new("javax.swing.JLabel", "")
label22=.bsf~new("javax.swing.JLabel", "")

add=.bsf~new('javax.swing.JButton', 'Add an Entry')
add~bsf.addEventListener( 'action', '', 'call add entries')
aggregate=.bsf~new('javax.swing.JButton', 'Aggregate Feeds')
aggregate~bsf.addEventListener( 'action', '', 'call aggregate entries, tmpColl')
create=.bsf~new('javax.swing.JButton', 'Create Feed as file')
```

```
create~bsf.addEventListener( 'action', '', 'call create input1, input2, input3,
                             input4, input5, inputtype, entries, inputname')
show=.bsf~new('javax.swing.JButton', 'Show feed')

show~bsf.addEventListener( 'action', '', 'call show input1, input2, input3,
input4, input5, inputtype, entries')

f~add(label1)
f~add(input1)
f~add(label2)
f~add(input2)
f~add(label3)
f~add(input3)
f~add(label4)
f~add(input4)
f~add(label5)
f~add(input5)
f~add(label21)
f~add(label22)
f~add(labeltype)
f~add(inputtype)
f~add(labelname)
f~add(inputname)
f~add(add)
f~add(aggregate)
f~add(create)
f~add(show)
f~add(label20)

f ~~pack ~~show ~~toFront

do forever
        INTERPRET .bsf~bsf.pollEventText
        if result="SHUTDOWN, REXX !" then leave
end
exit

show:
use arg input1, input2, input3, input4, label5, labeltype, entries

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(inputtype~gettext)
/*                                              sets the type of this feed */

feed~setTitle(input1~gettext)
feed~setAuthor(input2~gettext)
feed~setDescription(input3~gettext)
feed~setLink(input4~gettext)
feed~setLanguage(input5~gettext)

feed~setEntries(entries)

output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")

say output~outputString(feed,1)

return

add:
use arg entries, ent1, ent2, ent3, ent4, ent5, ent6, ent7, ent8

label24=.bsf~new("javax.swing.JLabel", "")
```

```
label25=.bsf~new("javax.swing.JLabel", "Add an Entry")
label26=.bsf~new("javax.swing.JLabel", "")
label27=.bsf~new("javax.swing.JLabel", "")
label28=.bsf~new("javax.swing.JLabel", "")
label29=.bsf~new("javax.swing.JLabel", "")

entlab1=.bsf~new("javax.swing.JLabel", "Entry Title")
entlab2=.bsf~new("javax.swing.JLabel", "Entry Date")
entlab3=.bsf~new("javax.swing.JLabel", "Entry Link")
entlab4=.bsf~new("javax.swing.JLabel", "Entry URI")
entlab5=.bsf~new("javax.swing.JLabel", "Entry Author")
entlab6=.bsf~new("javax.swing.JLabel", "Entry Description")
entlab7=.bsf~new("javax.swing.JLabel", "Entry Category")
entlab8=.bsf~new("javax.swing.JLabel", "Set Domain for Cat")

ent1=.bsf~new("javax.swing.JTextField", "First Entry", 10)
ent2=.bsf~new("javax.swing.JTextField", "2010-01-01", 10)
ent3=.bsf~new("javax.swing.JTextField", "http://www.google.com/", 10)
ent4=.bsf~new("javax.swing.JTextField", "http://www.google.com/", 10)
ent5=.bsf~new("javax.swing.JTextField", "it's me", 10)
ent6=.bsf~new("javax.swing.JTextField", "desc", 10)
ent7=.bsf~new("javax.swing.JTextField", "Search engine", 10)
ent8=.bsf~new("javax.swing.JTextField", "http://www.google.com/", 10)

addit=.bsf~new('javax.swing.JButton', 'Add an Entry')
addit~bsf.addEventListener( 'action', '', 'call addit ent1, ent2, ent3, ent4,
                                          ent5, ent6, ent7, ent8, entries')

close=.bsf~new('javax.swing.JButton', 'close add')
close~bsf.addEventListener( 'action', '', 'call closeadd')

f~add(label24)
f~add(label25)
f~add(label26)
f~add(entlab1)
f~add(ent1)
f~add(entlab2)
f~add(ent2)
f~add(entlab3)
f~add(ent3)
f~add(entlab4)
f~add(ent4)
f~add(entlab5)
f~add(ent5)
f~add(label27)
f~add(label28)
f~add(entlab6)
f~add(ent6)
f~add(entlab7)
f~add(ent7)
f~add(entlab8)
f~add(ent8)
f~add(close)
f~add(addit)
f~add(label29)

f ~~pack ~~show ~~toFront

do forever
     INTERPRET .bsf~bsf.pollEventText
     if result="SHUTDOWN, REXX !" then leave
end
```

```
exit

addit:
use arg ent1, ent2, ent3, ent4, ent5, ent6, ent7, ent8, entries

entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")
entry~setTitle(ent1~gettext)
--entry~setPublishedDate(DATE_PARSER~parse(ent2~gettext))   -- datum_new
entry~setLink(ent3~gettext)
entry~setUri(ent4~gettext)
entry~setAuthor(ent5~gettext)

description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue(ent6~gettext)
entry~setDescription(description)

cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName(ent7~gettext)
category~       setTaxonomyUri("ent8~gettext")
cat~add(category)
entry~setCategories(cat)

entries~add(entry)

label29~setText("Done!")
call syssleep 1
label29~setText("")

return

closeadd:

f~remove(label24)
f~remove(label25)
f~remove(label26)
f~remove(entlab1)
f~remove(ent1)
f~remove(entlab2)
f~remove(ent2)
f~remove(entlab3)
f~remove(ent3)
f~remove(entlab4)
f~remove(ent4)
f~remove(entlab5)
f~remove(ent5)
f~remove(label27)
f~remove(label28)
f~remove(entlab6)
f~remove(ent6)
f~remove(entlab7)
f~remove(ent7)
f~remove(entlab8)
f~remove(ent8)
f~remove(close)
f~remove(addit)
f~remove(label29)

f ~~pack ~~show ~~toFront

do forever
```

```
     INTERPRET .bsf~bsf.pollEventText
     if result="SHUTDOWN, REXX !" then leave
end
exit
return

create:
use arg input1, input2, input3, input4, label5, labeltype, entries, inputname

fileName = inputname~gettext

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(inputtype~gettext)
/*                                              sets the type of this feed */

feed~setTitle(input1~gettext)
feed~setAuthor(input2~gettext)
feed~setDescription(input3~gettext)
feed~setLink(input4~gettext)
feed~setLanguage(input5~gettext)
feed~setEntries(entries)

writer=.bsf~new("java.io.FileWriter",fileName)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

textname = "File has been written"
textname2 = "to file: " || fileName

label20~setText(textname)
label24~setText(textname2)

call syssleep 2

label20~setText("")
label24~setText("")

return

aggregate:
use arg entries, ent9, tmpColl

tmpColl = .array ~new

label30=.bsf~new("javax.swing.JLabel", "")
label31=.bsf~new("javax.swing.JLabel", "")
entlab9=.bsf~new("javax.swing.JLabel", "Enter feed URI")
ent9=.bsf~new("javax.swing.JTextField", "http://rss.orf.at/fm4.xml", 10)

addagg=.bsf~new('javax.swing.JButton', 'add URI to list')
addagg~bsf.addEventListener( 'action', '', 'i=i+1')
addagg~bsf.addEventListener( 'action', '', 'tmpColl~put(ent9~gettext, i)')
--addagg~bsf.addEventListener( 'action', '', 'call aggadd ent9, tmpColl')
aggshow=.bsf~new('javax.swing.JButton', 'show URIs')
aggshow~bsf.addEventListener( 'action', '', 'call aggshow')
--aggshow~bsf.addEventListener( 'action', '', 'say tmpColl~at(i)')
agg=.bsf~new('javax.swing.JButton', 'add entries to feed')
agg~bsf.addEventListener( 'action', '', 'call aggit ent9, entries, tmpColl')
closeagg=.bsf~new('javax.swing.JButton', 'close aggregate')
closeagg~bsf.addEventListener( 'action', '', 'call closeagg')
```

```
f~add(label30)
f~add(entlab9)
f~add(ent9)
f~add(addagg)
f~add(aggshow)
f~add(agg)
f~add(closeagg)
f~add(label31)

f ~~pack ~~show ~~toFront

do forever
     INTERPRET .bsf~bsf.pollEventText
     if result="SHUTDOWN, REXX !" then leave
end
exit

aggshow:

say "your urls are :"
DO item OVER tmpColl
SAY "[" || item || "]"
END
return

aggit:
use arg ent9, entries, tmpColl

do y=i-1 to 1 by -1
feedUrl=.bsf~new("java.net.URL",tmpColl~at(y))
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
infeed=input~build(xmlr)
entries~addAll(infeed~getEntries())
end

label31~setText("Done!")
call syssleep 1
label31~setText("")

return

closeagg:

f~remove(label30)
f~remove(entlab9)
f~remove(ent9)
f~remove(addagg)
f~remove(aggshow)
f~remove(agg)
f~remove(closeagg)
f~remove(label31)
f ~~pack ~~show ~~toFront

do forever
     INTERPRET .bsf~bsf.pollEventText
     if result="SHUTDOWN, REXX !" then leave
end
exit
return

::requires BSF.cls
```

program 12	This example is an extension to program 9, it builds a graphical interface to aggregate feeds; FeedAggregator2.rxj

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This example retrieves several feeds from a URI and aggregates them    */
/* to a single one    (in combination with  a graphical interface)        */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

say hello this aggregates a syndfeeds

/*          this creates an array of feed urls which will later be aggregated */

tmpColl = .array ~new
insert = .urlinsert~new
i = 0

frame=.bsf~new("java.awt.Frame", "Please enter the url")
frame~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame~setLayout( .bsf~new("java.awt.GridLayout",3,1) )
tf=.bsf~new("java.awt.TextField", "http://rss.orf.at/fm4.xml", 50)
but=.bsf~new('java.awt.Button', 'insert url')
but~bsf.addEventListener('action', '', 'i=i+1')
but~bsf.addEventListener('action', '', 'insert~insert(tmpColl, tf~getText, i) ')
but_end=.bsf~new('java.awt.Button', 'end insert')
but_end~bsf.addEventListener('action', '', 'call create')
frame~add(tf)
frame~add(but)
frame~add(but_end)

frame~~pack~~show~~toFront

do  forever

INTERPRET .bsf~bsf.pollEventText

if result="SHUTDOWN, REXX !" then leave

end

/*                            this retrieves the type of the aggregated feed */

create:

frame~toBack

inserttype = .typeinsert~new
typet = inserttype~getframe

fileName2 = "test5_aggregated_feed" typet

call Syssleep 1

inserttype~closeframe
frame~~toFront
```

```
say 'Your crated feed starts hear :'

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */

feed~setFeedType(typet)
/*                                                  sets the type of this feed */

/*                                 this aggregates the URI names to one String */
do x=i to 1 by -1
desc =  "Anonymous Aggregated Feed from _ "
desc = desc || tmpColl~at(x)
end

feed~setTitle("Aggregated Feed")
feed~setDescription(desc)
feed~setAuthor("Martin Stoppacher")
feed~setLink("http://martinstoppacher.com")

/*             create some entries for the feed by using the SyndFeed methods */
/*                        creates a Java array list for the entries of the feed*/
entries =.bsf~new("java.util.ArrayList")

/* this uses the URL array to retrieve the entries fro the feeds              */
/* and add them to the entries array                                          */

do y=i to 1 by -1
feedUrl=.bsf~new("java.net.URL",tmpColl~at(y))
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
infeed=input~build(xmlr)
entries~addAll(infeed~getEntries())
end

/*                                              output of the aggregated feeds */
feed~setEntries(entries)

writer=.bsf~new("java.io.FileWriter",fileName2)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

say output~outputString(feed,1) /*     usage of pretty print(0 yes or 1 no)   */

--call BSF "exit"         /* optional exit point */

PP : RETURN "[" || ARG(1)|| "]"

::requires BSF.cls

::class urlinsert

::method insert
use arg tmpColl, url, i

tmpColl[i] = url

say "your urls are :"
SAY tmpColl~string || ":"
DO item OVER tmpColl
SAY "[" || item || "]"
END
```

```
::class typeinsert

:: method getframe
expose frame2

frame2=.bsf~new("java.awt.Frame", "Please enter the FeedType")
/* ss_0.90,rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0 rss_2.0 or atom_0.3*/

frame2~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame2~setLayout( .bsf~new("java.awt.GridLayout",2,1) )
type=.bsf~new("java.awt.TextField", "rss_1.0", 50)
frame2~add(type)
but2=.bsf~new('java.awt.Button', 'insert FeedType')
frame2~add(but2)
but2~bsf.addEventListener('action', '', 'return type~getText ')
frame2~~pack~~show~~toFront

INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave

:: method closeframe
expose frame2

frame2~~toBack
drop frame2
```

program 13	A Feed Creator with additional functions. FeedCreator2.rxj

```
/* ********************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with   */
/* BSF4ooRexx                                                             */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/ */
/* You need to implement this API plus the JDOM API                       */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______; */
/*                                                                        */
/* This is a simple example of how to create a syndication feed with ooRexx */
/* With a GUI extension and the possibility to aggregate files too        */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009        */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                   */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                */
/* ********************************************************************** */

say "hello this creates a syndfeeds"

tmpColl = .array ~new
insert = .urlinsert~new
i=0

/* grafical extensin                                                      */

frame2=.bsf~new("java.awt.Frame", "Please enter the FeedType")
frame2~bsf.addEventListener( 'window', 'windowClosing', 'call BSF "exit"')
frame2~setLayout(.bsf~new("java.awt.GridLayout",4,1))

type=.bsf~new("java.awt.TextField", "rss_2.0", 50)
but2=.bsf~new('java.awt.Button', 'Create Feed with Type')
but2~bsf.addEventListener('action', '', 'call create, type')
tf=.bsf~new("java.awt.TextField", "http://rss.orf.at/fm4.xml", 50)
but=.bsf~new('java.awt.Button', 'insert url')
but~bsf.addEventListener('action', '', 'i=i+1')
but~bsf.addEventListener('action', '', 'insert~insert(tmpColl, tf~getText, i)')

frame2~add(type)
```

```
frame2~add(but2)
frame2~add(tf)
frame2~add(but)

frame2~~pack~~show~~toFront

do forever
INTERPRET .bsf~bsf.pollEventText
if result="SHUTDOWN, REXX !" then leave
end

create:

typet = type~getText
fileName = "test6_new_feed" typet                          /* set file name */
/*                              converting the rexx date() to an other format */
datum = date()
parse var datum day mounth year
if mounth="Dec" then mounth=12
if mounth="Jan" then mounth=01              /*  simply add the necessary once */
datum_new = year"-"mounth"-"day
say "Todays date: " datum_new

call Syssleep 1

say 'Your crated feed starts hear :'

feed=.bsf~new("com.sun.syndication.feed.synd.SyndFeedImpl")
/*      using syndfeedImpl to creat a feed using the independent object model */
feed~setFeedType(typet)
/*                                          sets the type of this feed */
desc = "My favorite webpages; format:_" typet

feed~setTitle("This is a simple Sample")
feed~setDescription(desc)
feed~setAuthor("Martin Stoppacher")
feed~setLink("http://martinstoppacher.com")
feed~setLanguage("English")
/*                               Using the SyndImageImpl to add a person     */
im=.bsf~new("com.sun.syndication.feed.synd.SyndImageImpl")
im~setTitle("A new image")
im~setLink("http://www.freefoto.com/images/14/26/14_26_4---Trevi-Fountain--
Rome--Italy_web.jpg")
im~setDescription("Trevi Fountain Rome")
im~setUrl("http://www.freefoto.com/images/14/26/14_26_4---Trevi-Fountain--Rome--
Italy_web.jpg")

feed~setImage(im)

auth =.bsf~new("java.util.ArrayList")    /* a Java array to do setAuthors     */
/*                                 Using the SyndPersonImpl to add a person     */
pers=.bsf~new("com.sun.syndication.feed.synd.SyndPersonImpl")
pers~setName("Martin Stoppacher")
pers~setUri("martinstoppacher.com")
pers~setEmail("office@artinstoppacher.com")
auth~add(pers)

pers2=.bsf~new("com.sun.syndication.feed.synd.SyndPersonImpl")
pers2~setName("A second person")
pers2~setUri("the Uri of the person")
pers2~setEmail("the persons email")
auth~add(pers2)
```

```
feed~setAuthors(auth)

--say feed~toString                        /* string representation of the feed */

/*        create a java  SimpleDateFormat instance for entry~setPublishedDate */
DATE_PARSER = .bsf~new("java.text.SimpleDateFormat", "yyyy-MM-dd")

/*                        creates a Java array list for the entries of the feed*/
entries =.bsf~new("java.util.ArrayList")

entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")
/*                                      add some entries to an entriy object */
entry~setTitle("The Project Rome Web Page")
entry~setPublishedDate(DATE_PARSER~parse(datum_new))
entry~setLink("https://rome.dev.java.net/")
entry~setUri("https://rome.dev.java.net/")
entry~setAuthor("dev.java.net")

/*                     create a independent description content implemetation */

description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue("ROME Web Page")
entry~setDescription(description)

/*                              add the description object to the entry object */
/* create a new category; by adding the SyndCategoryImpl to a Java array      */
/* object and  this array object to the entry object                          */

cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName("Java")
category~       setTaxonomyUri("https://rome.dev.java.net/")
cat~add(category)
entry~setCategories(cat)

/*                              add the entry to the Java entries array object */
entries~add(entry)

/*                                                              a secound entry */
entry =.bsf~new("com.sun.syndication.feed.synd.SyndEntryImpl")

entry~setTitle("Java 1.4 Documentation")
entry~setPublishedDate(DATE_PARSER~parse(datum_new))
entry~setLink("http://java.sun.com/j2se/1.4.2/docs/api/")
entry~setUri("http://java.sun.com/")
entry~setAuthor("java.sun.com/")

description=.bsf~new("com.sun.syndication.feed.synd.SyndContentImpl")
description~setType("text/plain")
description~setValue("Documentation of the Java 1.4 API")
entry~setDescription(description)

cat =.bsf~new("java.util.ArrayList")
category=.bsf~new("com.sun.syndication.feed.synd.SyndCategoryImpl")
category~setName("Documentation")
category~       setTaxonomyUri("http://java.sun.com/j2se/1.4.2/docs/api/")
cat~add(category)
entry~setCategories(cat)

entries~add(entry)

feed~setEntries(entries)
/*                                  sets the Java array as entries of the feed */
```

```
-- -------------

/* this part adds the possibilitie of adding some aggregated antries to the   */

/* feed.   Take a look at example "rome8_a and b"                              */

do y=i to 1 by -1

feedUrl=.bsf~new("java.net.URL",tmpColl~at(y))
input=.bsf~new("com.sun.syndication.io.SyndFeedInput")
xmlr=.bsf~new("com.sun.syndication.io.XmlReader", feedUrl)
infeed=input~build(xmlr)
entries~addAll(infeed~getEntries())
end

-- ---------------
/*                                                                writer part */
writer=.bsf~new("java.io.FileWriter",fileName)
output=.bsf~new("com.sun.syndication.io.SyndFeedOutput")
output~output(feed,writer)
writer~close()

say output~outputString(feed,1) -- usage of pretty print(0 yes or 1 no)

PP : RETURN "[" || ARG(1)|| "]"

::requires BSF.cls

::class urlinsert

::method insert
use arg tmpColl, url, i

tmpColl[i] = url

say "your urls are :"
SAY tmpColl~string || ":"
DO item OVER tmpColl
SAY "[" || item || "]"
END
```

program 14	This uses the abstract data model (WireFeed) from Project ROME to create a web feed; FeedCreator_WireFeed.rxj

```
/* ************************************************************************** */
/* Example of a syndication feed reader using the Project Rome API with       */
/* BSF4ooRexx                                                                 */
/* current version of Rome: rome1.0.jar           https://rome.dev.java.net/  */
/* You need to implement this API plus the JDOM API                           */
/* jdom.jar  ,   you can find this at             https://jdom.org/_______;   */
/*                                                                            */
/* This is a simple example of how to create a syndication feed with ooRexx   */
/* using the abstract data model from PR.     WireFeed                        */
/* created by Martin Stoppacher     date:   26.12.2009    (c) 2009            */
/* license:    LGPL 3.0       used versions: Java 1.6, ooRexx 4.0, Bsf4ooRexx */
/*             (Lesser Gnu Public License version 3.0),                       */
/*             cf. <http://www.gnu.org/licenses/lgpl.html>                    */
/* ************************************************************************** */

say "hello this creates a syndfeeds"

typet = "atom_0.3"
--typet = "rss_1.0"
fileName = "test6_new_feed" typet
```

```
DATE_PARSER = .bsf~new("java.text.SimpleDateFormat", "yyyy-MM-dd")

say 'Your crated feed starts hear :'

feed=.bsf~new("com.sun.syndication.feed.atom.Feed",typet)
/*           using atom.Feed to create a feed using the abstract object model */
--feed~setFeedType(typet)                              /* only Atom types possible */
/*                              optionally this sets the type of the feed */
/*                          using a few methods to set some feed information   */
feed~setTitle("This is a simple Sample")
feed~setRights("using WireFeed")
feed~setUpdated(DATE_PARSER~parse("2010-01-01"))

/*                                                                  writer part */
--writer=.bsf~new("java.io.FileWriter",fileName)
output=.bsf~new("com.sun.syndication.io.WireFeedOutput")
--output~output(feed,writer)
--writer~close()

say output~outputString(feed,1) -- usage of pretty print(0 yes or 1 no)

PP : RETURN "[" || ARG(1)|| "]"

::requires BSF.cls
```

Appendix C different examples

The Appendix C includes a few additional programs and diagrams useful for the understanding of the covered topics.

program 1	Connecting to a MySql Database via Java; Mysql.java; code(5); (Link 14)

```
/*      ****************************************************************    */
/* This is just a short example of implementing Java MySql DB connector       */
/* using Java version 1.6                                                     */
/* created by Martin Stoppacher         26.12.2009                            */
/*      ****************************************************************    */

import java.sql.*;

public class mysql {

    public static void main(String[] args) {

    String uid = "root";
    String url = "jdbc:mysql://localhost/test";

    try {
      DriverManager.registerDriver(new com.mysql.jdbc.Driver());
      Connection conn = DriverManager.getConnection(url, "root", "aron1212");
      Statement stmt = conn.createStatement ();
         ResultSet rset =
         stmt.executeQuery("select NR, NAME from mytest");
               System.out.println("NR_____NAME");
         while (rset.next()) {
         System.out.print(rset.getString("nr"));
         System.out.print("_____");
         System.out.println(rset.getString("NAME"));
         }
        System.out.println("END");

    } catch (SQLException e) {
      System.out.println(e);
    }
  }
}
```

program 2	Using some of the “java.net.URL” features; getinfo.rxj;

```
/*      ****************************************************************    */
/* This is just a short example of implementing the  "java,net.URL" classes   */
/* using BSF4ooRexx   (ooRexx 4.0, BSF4ooRexx, Java 1.6)                      */
/* created by Martin Stoppacher         26.12.2009                            */
/*      ****************************************************************    */
say hello this reads a syndfeed
say please type in the url
url= "http://rss.orf.at/fm4.xml"

f=.bsf~new("java.net.URL", url)
/*                         creating a java url object with the above url     */

say f~getAuthority()        /*  Gets the authority part of this URL          */
say f~getDefaultPort()      /*  Gets the default port number of the protocol */
                            /*  associated with this URL                     */
say f~getPort()             /*  Gets the port number of this URL             */
say f~getFile()             /*  Gets the file name of this URL               */
say f~getHost()             /*  Gets the host name of this URL, if applicable */
```

```
say f~getProtocol()          /*  Gets the protocol name of this URL          */
say f~getQuery()             /*  Gets the query part of this URL             */
say f~hashCode()             /*  Creates an integer suitable for hash        */
                             /*     table indexing                           */

::requires BSF.cls
```

program 3	Connecting to a MySql Database using BSF4ooRexx; Mysql.rxj;

```
/*      ****************************************************************    */
/* This is just a short example of implementing Java MySql DB connector      */
/* using BSF4ooRexx   (ooRexx 4.0, BSF4ooRexx, Java 1.6)                     */
/* created by Martin Stoppacher         26.12.2009                           */
/*      ****************************************************************    */
say hello access to a my sql database via java using bsf4rexx
uid = "root";
--say uid
url = "jdbc:mysql://localhost/test";
--say url

mydrive=.bsf~new('com.mysql.jdbc.Driver')
man=bsf.loadClass("java.sql.DriverManager")~registerDriver(mydrive)
conn=bsf.loadClass("java.sql.DriverManager")~getConnection(url, uid, "pw")
stmt=conn~createStatement
rset=stmt~executeQuery("select NR, NAME from mytest")

say "NR_____NAME"
do while rset~next
say rset~getString("nr")  || _____ || rset~getString("NAME")
end

say end

::requires BSF.CLS
```

Diagram 1	UML Class-diagram; Shows the independent data model with a class-diagram. Within the rectangle you can see the available methods. [BMcL02]

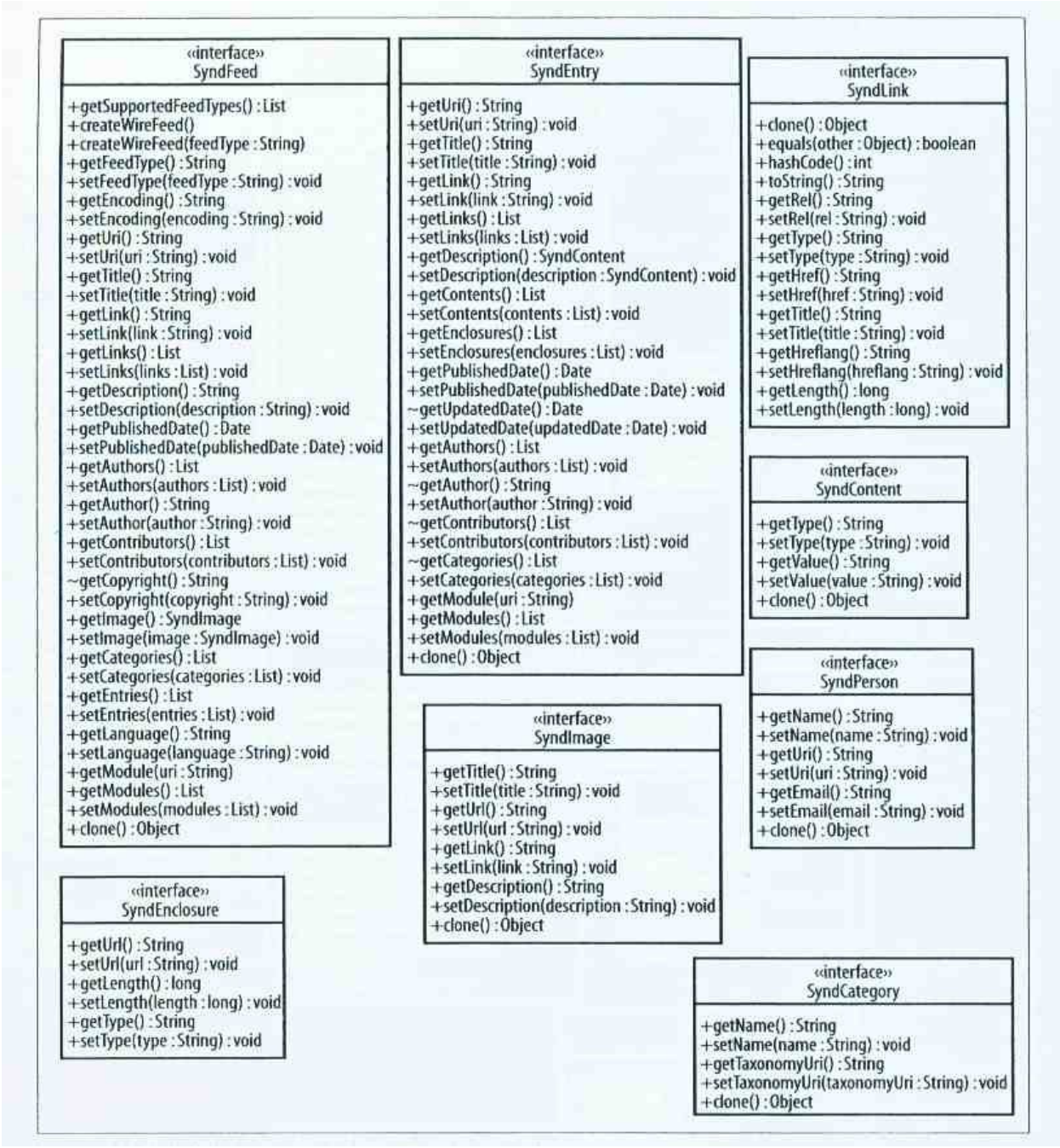

Diagram 2	UML Class-diagram; Shows the abstract RSS data model with a class-diagram. Within the rectangle you can see the available methods. [BMcL02]

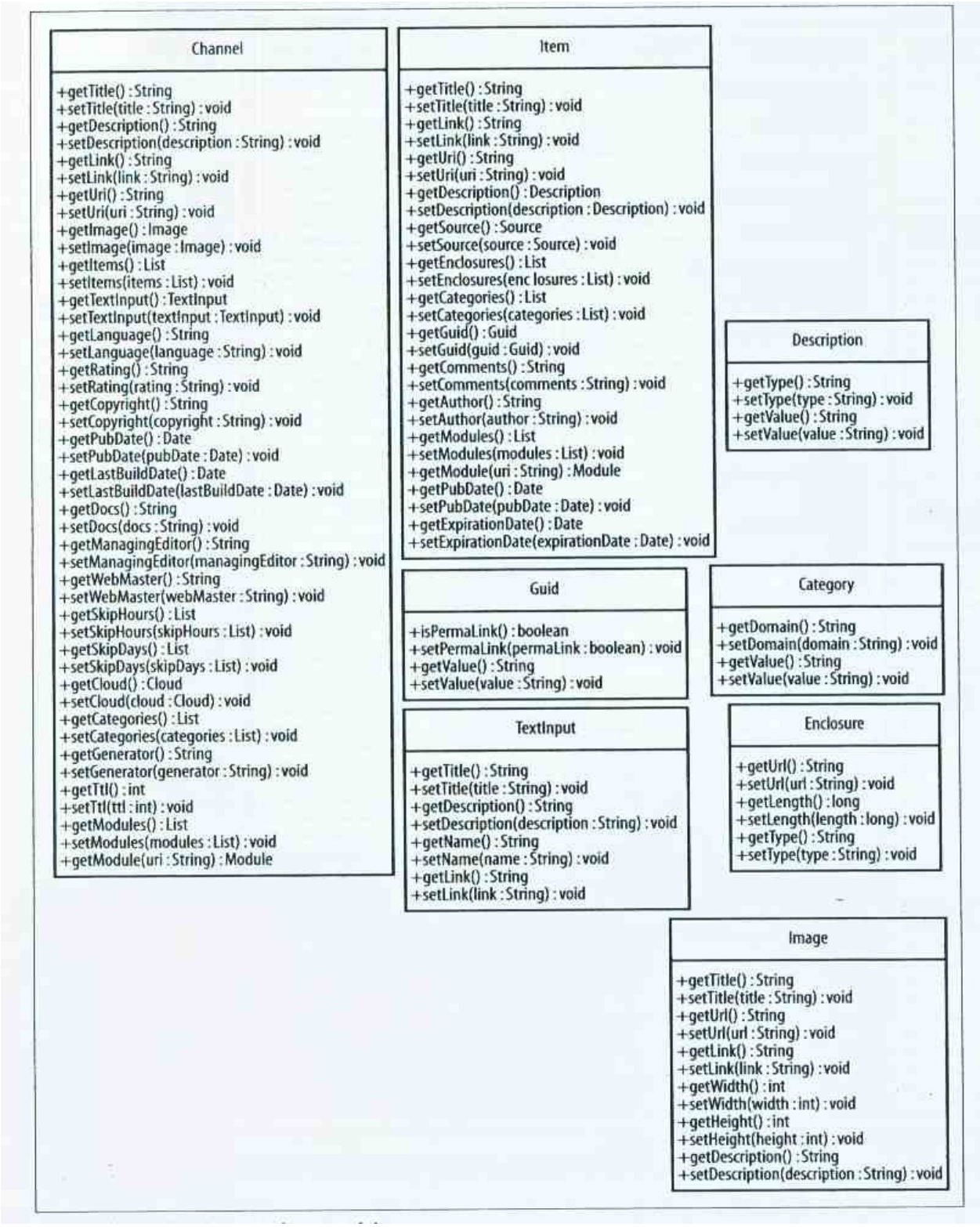

Diagram 3	UML Class-diagram; Shows the abstract ATOM data model with a class-diagram. Within the rectangle you can see the available methods. [BMcL02]

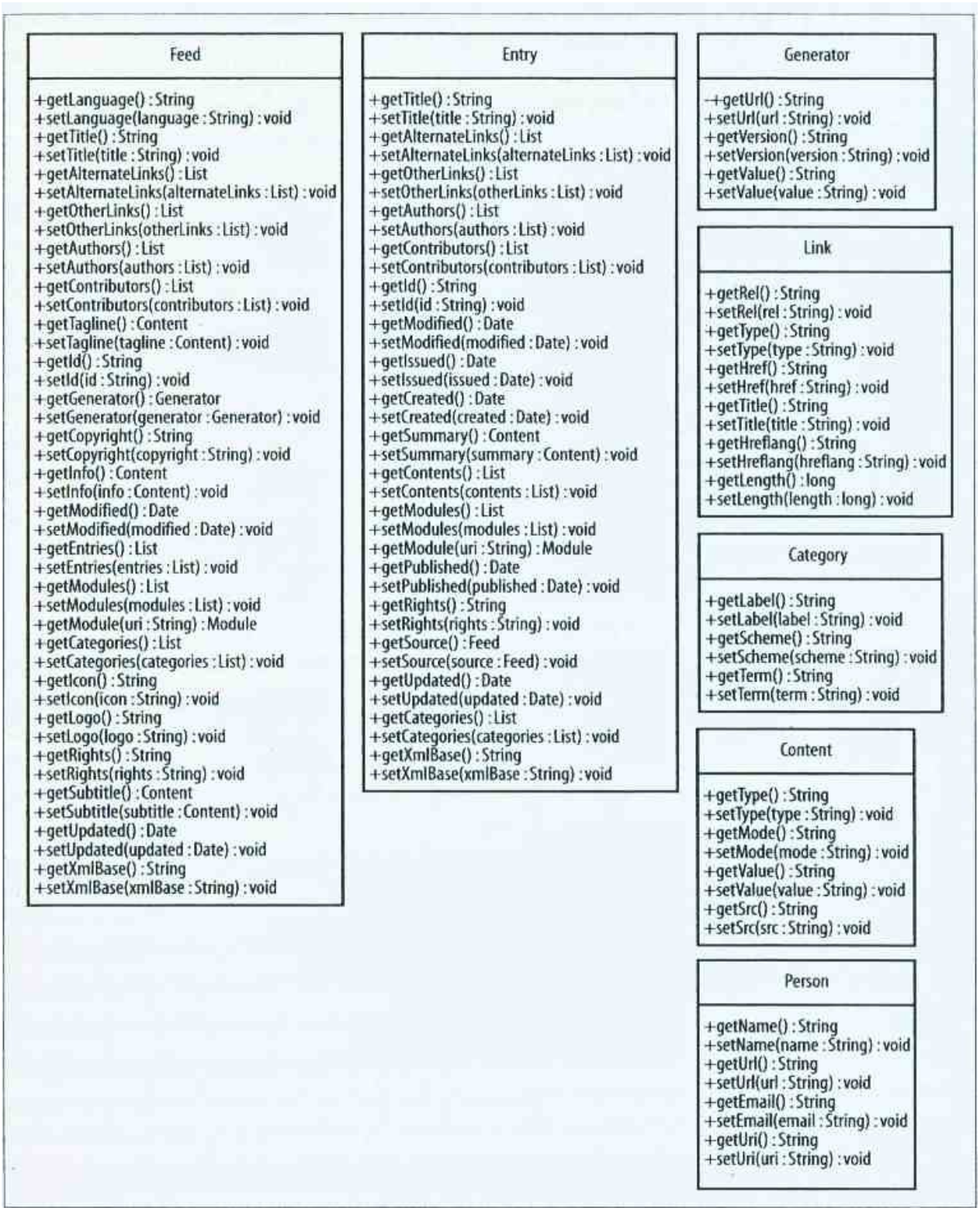

Table of links

number	*description*
Link 1	Wikipedia article about XML (retrieved on: 2009-12-23) http://en.wikipedia.org/w/index.php?title=XML&oldid=334951019
Link 2	Wikipedia article about RSS feeds (retrieved on: 2009-12-23) http://en.wikipedia.org/w/index.php?title=RSS&oldid=334620412
Link 3	Wikipedia article about Atom feeds (retrieved on: 2009-12-23) http://en.wikipedia.org/w/index.php?title=Atom_(standard)&oldid=335449485
Link 4	Wikipedia article about the Rexx language (retrieved on: 2009-12-23) http://de.wikipedia.org/w/index.php?title=REXX&oldid=62771442
Link 5	Open Object Rexx Homepage (retrieved on: 2009-11-15) http://www.oorexx.org/
Link 6	Rexx Language Association Homepage (retrieved on: 2009-11-16) http://www.rexxla.org/
Link 7	Project ROME Homepage (retrieved on: 2009-10-11) https://rome.dev.java.net/
Link 8	Java 1.4 online documentation (retrieved on: 2009-11-20) http://java.sun.com/j2se/1.4.2/docs/index.html
Link 9	Wikipedia article about RDF (retrieved on: 2009-11-17) http://en.wikipedia.org/w/index.php? title=Resource_Description_Framework&oldid=333919175
Link 10	Wikipedia article about DTD (retrieved on: 2009-11-17) http://en.wikipedia.org/w/index.php? title=Document_Type_Definition&oldid=331414014
Link 11	Webpage of the Dublin Core meta data Initiative (retrieved on: 2009-11-17) http://purl.org/dc
Link 12	Information about PR modules (retrieved on: 2009-11-09) http://wiki.java.net/bin/view/Javawsxml/RomeModules? TWIKISID=a275da5164b00d7d729fc4a9235168ee
Link 13	Information about using Jar files (retrieved on: 2009-12-14) http://java.sun.com/developer/Books/javaprogramming/JAR/basics/
Link 14	Using a MySQL Database connector in Java (retrieved on: 2010-01-06) http://www.stardeveloper.com/articles/display.html?article=2003090401&page=1
Link 15	Java download Archive (retrieved on: 2009-12-12) http://java.sun.com/products/archive/
Link 16	Mysql Java Connector (retrieved on: 2009-12-02) http://dev.mysql.com/downloads/connector/j/
Link 17	Information about how to set the Java classpath (retrieved on: 2009-12-02) http://java.sun.com/j2se/1.3/docs/tooldocs/win32/classpath.html

Link 18	Homepage of the SAX project (retrieved on: 2010-12-02) http://www.saxproject.org/
Link 19	Article about Project Rome (how it works) (retrieved on: 2009-11-13) http://wiki.java.net/bin/view/Javawsxml/Rome04HowRomeWorks
Link 20	Webpage of the BSF project (retrieved on: 2010-01-16) http://jakarta.apache.org/bsf/
Link 21	Wikipedia artivle about monospaced fonts (retrieved on: 2010-01-16) http://en.wikipedia.org/w/index.php?title=Monospaced_font&oldid=334494117
Link 22	ROME in a Day: Parse and Publish Feeds in Java, by Mark Woodman (retrieved on: 2009-01-15) http://www.xml.com/pub/a/2006/02/22/rome-parse-publish-rss-atom-feeds-java.html
Link 23	Taking a Tour of ROME, Randy J. Ray (retrieved on: 2010-01-15) http://today.java.net/pub/a/today/2006/02/02/tour-of-rome.html
Link 24	Interview with Patrick Chanezon, Interviews about Syndication in the Enterprise. (retrieved on: 2009-01-15) http://inkblots.markwoodman.com/rss-diaries/patrick-chanezon
Link 25	The General Public Licence 3.0 (retrieved on: 2009-12-25) http://www.gnu.org/licenses/gpl-3.0.txt
Link 26	Webpage of the JDOM projekt (retrieved on: 2010-01-02) http://www.jdom.org/
Link 27	Project Rome online documentation (retrieved on: 2010-01-02) http://wiki.java.net/bin/view/Javawsxml/Rome
Link 28	Webpage of the OPML project (retrieved on: 2010-01-02) http://www.opml.org/
Link 29	Oxygen XML Editor 11.1 (retrieved on: 2010-01-15) http://www.heise.de/software/download/oxygen_xml_editor/43758

References

[ROME09] Project ROME web resources; 2009;
URL (2009-12-22) https://rome.dev.java.net/;

[RexxLA09] The Rexx Language Association Homepage;
URL (2010-01-06) http://www.rexxla.org;

[ooR09] About Open Object Rexx; Rexx Language Association; 2009;
URL (2009-12-22) http://www.ooRexx.org/

[BSF4ooR] BSF4ooRexx (Bean Scripting Framework for open object Rexx) 2009;
URL (2009-12-22) http://wi.wu-wien.ac.at:8002/rgf/rexx/bsf4oorexx/current/

[BSF4R] BSF4Rexx (Bean Scripting Framework for Rexx) 2009;
URL (2009-12-22) http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/

[Fos05] REXX Programmer's Reference (Programmer to Programmer);
Wrox; edition (März 2005);
Author: Howard Fosdick; ISBN 0-13-780651-5;

[CoSh90] The REXX Language: A Practical Approach to Programing;
Prentice Hall PTR; 2nd edition (March 2, 1990);
Author: M. F. Cowlishaw; ISBN-13: 978-0137806515; 1990;

[w3c09] Extensible Markup Language; W3c information about XML;
Copyright © 1996-2003 W3C® (MIT, ERCIM, Keio)
URL (2009-12-08) http://www.w3.org/XML/

[BMcL02] Java und XML (Broschiert);
O'Reilly/VVA; Edition(April 2002);
Author: Brett McLaughlin; ISBN-13: 978-3897212800;

[Flat01] Flatscher, Rony G.: Java Bean Scripting with Rexx;
"12 International Rexx Symposium", Raleigh, North Carolina;
USA, April 30th – May 2nd , 2001;
URL (2010-01-03) http://wi.wu.ac.at:8002/rgf/rexx/orx12/
JavaBeanScriptingWithRexx%5forx12%2epdf

[Flat03] Flatscher, Rony G.: The Augsburg version of BSF4Rexx;
"The 2003 International Rexx Symposium", Raleigh, North Carolina;
USA, May 4th – May 7th , 2003;
URL (2010-01-03) http://wi.wu.ac.at:8002/rgf/rexx/orx14/
orx14%5fbsf4rexx%2dav%2epdf

[Flat04] Flatscher, Rony G.: Camouflaging Java as Object REXX;
"The 2004 International Rexx Symposium", Böblingen;
Germany, May 3rd – May 6th , 2004;
URL (2010-01-03) http://wi.wu-wien.ac.at/rgf/rexx/orx15/
2004_orx15_bsf-orxlayer.Pdf

[Flat05_a] Flatscher, Rony G.: "Automating OpenOffice.org with OORexx,
Architecture, gluing to Rexx using BSF4Rexx";
"The 2005 International Rexx Symposium", Los Angeles, Californien;
USA, April 17th – April 21st , 2005;
URL (2010-01-03) http://wi.wu.ac.at:8002/rgf/rexx/orx16/
2005%5forx16%5fGluing2ooRexx%5fOOo%2epdf

[Flat05_b] Flatscher, Rony G.: Leaping from Classic to Object;
"2005 International Rexx Symposium"; Los Angeles, California;
USA, April 2005;
URL (2010-01-03) http://wi.wu.ac.at:8002/rgf/rexx/orx16/2005%5fLeaping
%5ffrom%5fClassic%5fto%5fObject%5f%2d%5fTutorial
%2epdf

[Flat06] Flatscher Rony G.: The Vienna Version of BSF4Rexx;
"2006 International Rexx Symposium", Austin, Texas;

USA, April 2006;
URL (2010-01-03) http://wi.wu.ac.at:8002/rgf/rexx/orx17/2006%5forx17%5fBSF%5fViennaEd%2epdf

[Flat09_a] Flatscher Rony G.: The 2009 Edition of BSF4Rexx;
"2009 International Rexx Symposium", Chilworth, England;
Great Britain, May 18th – May 21st 2009;
URL (2010-01-03): http://wi.wu.ac.at:8002/rgf/rexx/orx20/2009%5forx20%5fBSF4Rexx%2d20091031%2darticle%2epdf

[Flat09_b] Flatscher, Rony G; Java Automation;
Course slides (in German/English); 2009;
URL (2010-01-03) http://wi.wu-wien.ac.at:8002/rgf/wu/lehre/autojava/material/foils/

[Aha05] Ahammer Andreas; OpenOffice.org Automation;
Object Model, Scripting Languages, "Nutshell"-Examples;
(Vienna University of Economics and Business Administration), Austria, 2005;
URL (2010-01-03) http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/BakkStuff/2005/200511_OOo-Ahammer/200511_OOoAutomation.pdf

[Burger05] Burger Martin; OpenOffice.org Automation with Object Rexx;
Vienna University of Economics and Business Administration, Austria, 2005;
URL (2010-01-03) http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/BakkStuff/2006/200605_Burger/Bakk_Arbeit_Burger20060519.pdf

[Prem06] ooRexx Snippets for OpenOffice.org Writer Version 2.1: 2006-07-24
Vienna University of Economics and Business Administration, Austria, 2006;
URL (2010-01-04) http://wi.wu-wien.ac.at:8002/rgf/diplomarbeiten/BakkStuff/2006/200607_Prem/20060724_ooRexxSnippetsOOoWriter_2.1_odt.pdf

[Prech_00] An Empirical Comparison of Seven Programming Languages
combaring C, C++, Java, Perl, Python, Rexx, and Tcl.
Lutz Prechelt, University of Karlsruhe, 2000
URL

[HP05] Java Einführung; Kursunterlage (German);
University of Natural Resources and Applied Life Sciences, Vienna;
Version: July 2005 Author: Hubert Partl
URL (2009-12-22) http://www.boku.ac.at/javaeinf/jein.html

[GNU09] GNU Lesser General Public License,
URL (2006-06-10) http://www.gnu.org/
http://www.gnu.org/licenses/gpl-3.0.txt

[wiki09_a] Wikipedia article about XML
URL (2009-11-05) http://en.wikipedia.org/w/index.php?
title=XML&oldid=334951019

[wiki09_b] Wikipedia article about RSS Feeds
URL (2009-11-05) http://en.wikipedia.org/w/index.php?
title=RSS&oldid=334620412

[wiki09_c] Wikipedia article about ATOM Feeds
URL (2009-11-05) http://en.wikipedia.org/w/index.php?
title=Atom_(standard)&oldid=335449485

[wiki09_d] Wikipedia article about Web_Syndication
URL (2009-11-05) http://en.wikipedia.org/w/index.php?
title=Web_syndication&oldid=332308170

Index