# Seminar paper

# BSF4ooRexx:
# JSP with javax.script Languages

Author: Nora Lengyel

Matriculation no: 1552636

Class Title: Projektseminar aus Wirtschaftsinformatik (Schiseminar)

Instructor: ao.Univ.Prof. Mag. Dr. Rony G. Flatscher

Term: Winter Term 2019/2020

Vienna University of Economics and Business

## Content

**List of Figures**

# 1. Introduction

This seminar paper provides information about the Apache Tomcat servers and their operation (chapter 2. Tomcat), explains how cookies work and are created (chapter 3. Cookie). Moreover, it helps to understand the operation of Java Servlet Pages (chapter 4. JSP (Java Servlet Pages)) based on Taglibs (chapter 5 Taglib). Tomcat is a web server, which contains web sites, and supports the communication with the clients through HTTP protocol. All programming examples are executed on the Tomcat Server. Examples based on the BSF Taglib are created, one that uses Jython (chapter 6. Jython Cookie in JSP) and one that lets Open Object Rexx (chapter 7. Rexx Cookie in JSP with BSF Taglib) written programming codes on Tomcat execute. An example with JSR-223 Taglib with Rexx is also created (chapter 8. Rexx Cookie in JSP with JSR-223 Taglib). As examples, the creation, modification, deleting and listing of cookies have been applied. Cookies are stored on the user computer and store small amount of information, which makes it possible that the web server recognizes the user for the second time. JSPs allow to create applications and execute them on a web server.

The creation of the examples requires the installation of the following software:

1. Java Development Kit (JDK) version 8 or later: [1]
2. Rexx - The 5th version can be found under “files” on this link: [2]
3. BSF4ooRexx: [3]
4. Apache BSF 2.4: [4]
5. Tomcat (installation guide in chapter 2. Tomcat)
6. Jython, version 2.5.3: [5]
7. Apache Ant: [6]
8. BSF Taglib: [7]
9. JSR-223 Taglib: [8]

For the installation the version of the processor has to be taken into consideration. The use of 64-bit version software is recommended. In case the above software is installed with different versions, errors occur. Both, the 32-bit and the 64-bit version software can be executed on the computer (if the computer has 64-bit processor). In this example the software above must interact with each other and in case of different bit versions this interaction might not be possible.

# 2. Tomcat

“A server is a computer program or a device that provides functionality for other programs or devices, called clients.” [9]

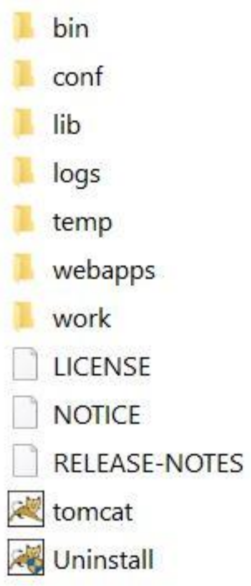
Tomcat is a web server, fully written in java language. This makes Tomcat available with every operation system that supports java.

## 2.1 Introduction to Tomcat

The actual version of the Tomcat is Tomcat 9.0. There have been many changes compared to the old versions, therefore it is recommended to search tutorials for the current version of Tomcat.

**Directory structure of Tomcat**

Tomcat has the following directory structure:



```
bin
conf
lib
logs
temp
webapps
work
LICENSE
NOTICE
RELEASE-NOTES
tomcat
Uninstall
```

*Figure 1 - Tomcat directory structure*

The “`bin`” directory contains batch files to start or stop the server. Batch files are script files to be executed by the command line. Self-written batch files can also be put in this directory.

The “`conf`” directory contains configuration files.

The “`lib`” directory contains java files that are necessary for Tomcat

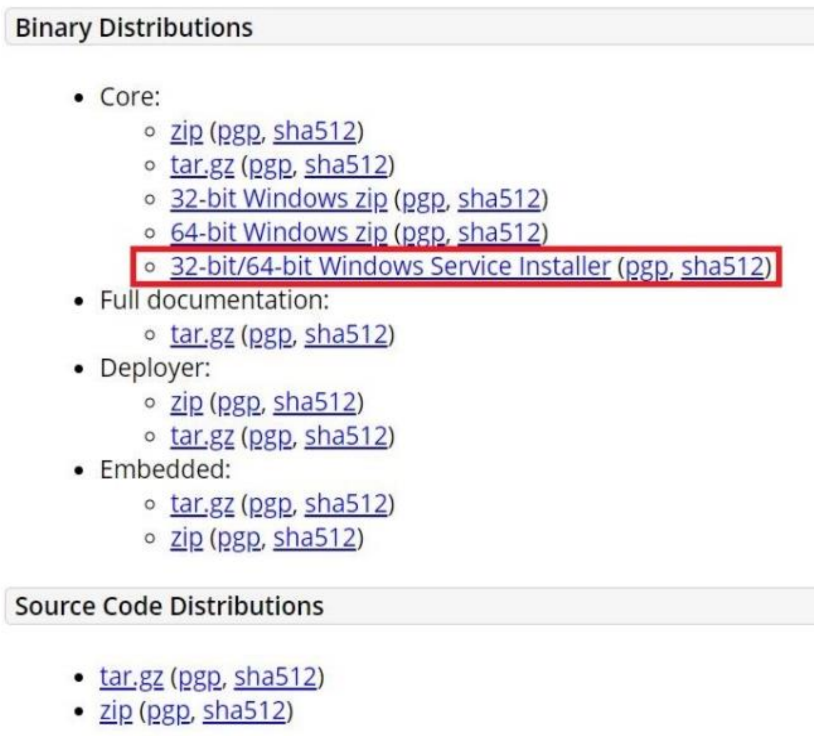
The “`logs`” directory contains the log files.

The “`webapps`” directory is where the applications, the folders that include JSPs, are placed.

The “`work`” contains files, which allow the JSPs to work.

## 2.2 The Installation of Tomcat

The Tomcat can be downloaded via the following link: [10]



Binary Distributions

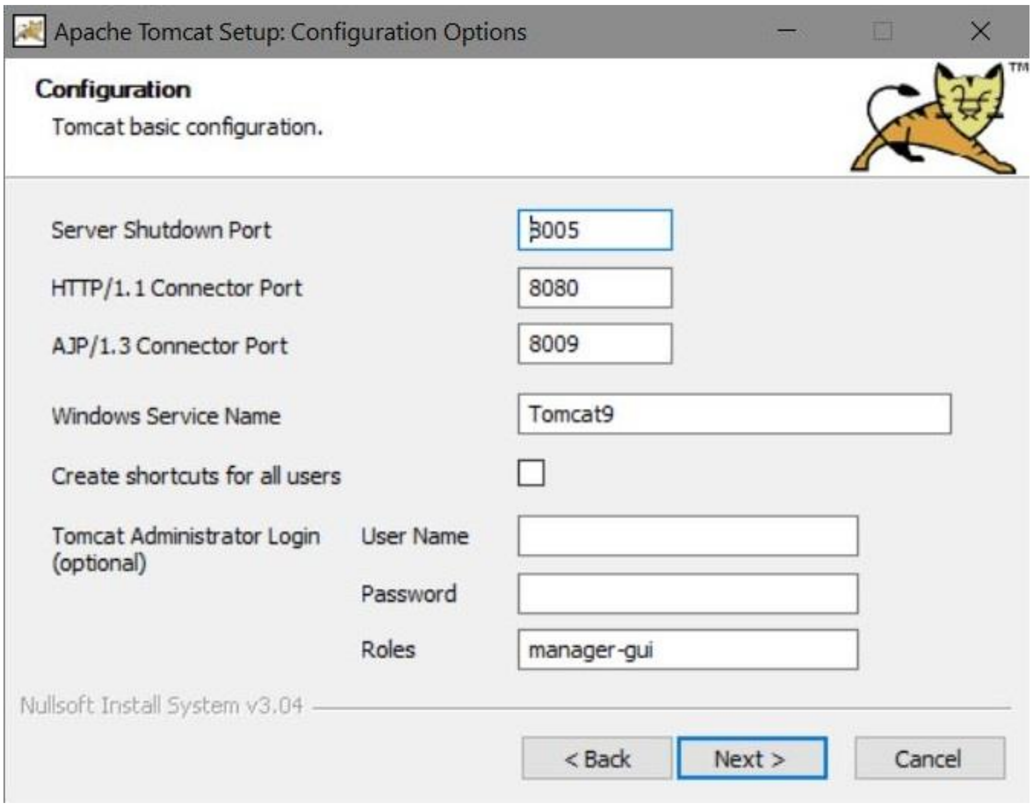
- Core:
  - zip (pgp, sha512)
  - tar.gz (pgp, sha512)
  - 32-bit Windows zip (pgp, sha512)
  - 64-bit Windows zip (pgp, sha512)
  - 32-bit/64-bit Windows Service Installer (pgp, sha512)
- Full documentation:
  - tar.gz (pgp, sha512)
- Deployer:
  - zip (pgp, sha512)
  - tar.gz (pgp, sha512)
- Embedded:
  - tar.gz (pgp, sha512)
  - zip (pgp, sha512)

Source Code Distributions

- tar.gz (pgp, sha512)
- zip (pgp, sha512)

*Figure 2 - Versions of Tomcat installer*

Tomcat can be installed to Windows, Linux or macOS. In case of Windows operating system the 32-bit/64-bit Windows Service Installer should be chosen. The downloaded file includes the Tomcat installer, which helps to install Tomcat easily. Each default setting may be accepted.



Apache Tomcat Setup: Configuration Options

**Configuration**

Tomcat basic configuration.

| | | |
|---|---|---|
| Server Shutdown Port | | 8005 |
| HTTP/1.1 Connector Port | | 8080 |
| AJP/1.3 Connector Port | | 8009 |
| Windows Service Name | | Tomcat9 |
| Create shortcuts for all users | | ☐ |
| Tomcat Administrator Login (optional) | User Name | |
| | Password | |
| | Roles | manager-gui |

*Figure 3 - Port settings in Apache Tomcat installer*

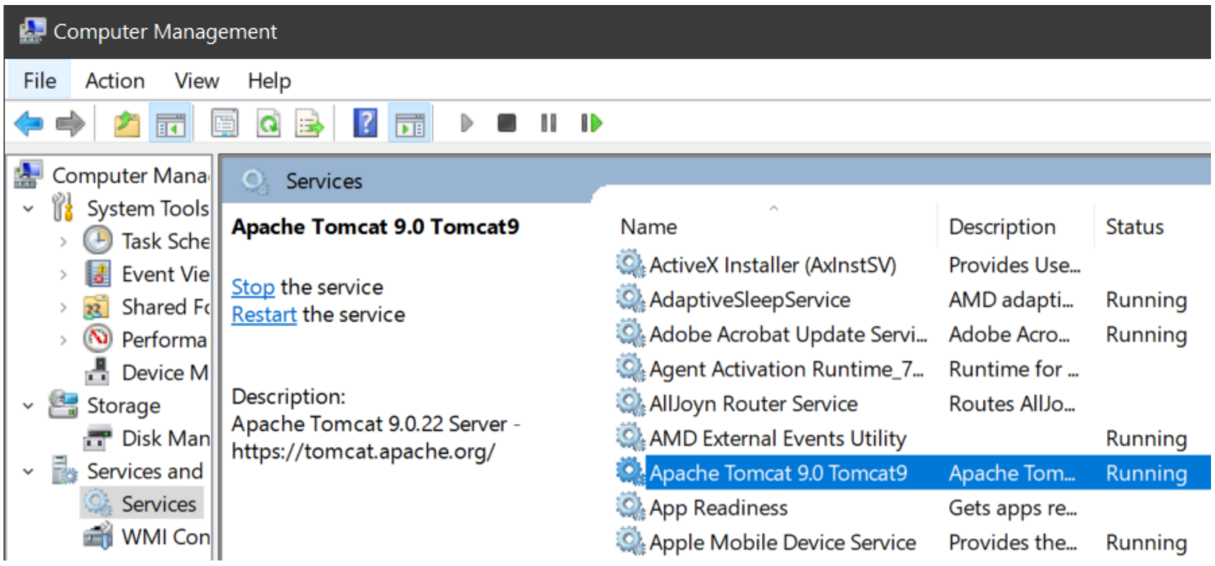
The installer offers many configuration options. The `HTTP/1.1 Connector Port` defines the port, which allows access to our applications. The default setting is 8080, which makes Tomcat available via the url: *http://localhost:8080/.*

The localhost is describing your local computer address. Instead of localhost the IP address “`127.0.0.1`” could also be used. Ports are defining the exact destination of the data communication. It is possible to run more servers on the same computer and it defines exactly where the request should be forwarded.

The Tomcat is installed in `C:\Program Files\Apache Software Foundation`.

The server can be started, re-started or stopped, in the computer management window or via batch files saved in the “`bin`” directory in Tomcat. The “`Computer Management`” can be opened by clicking with right mouse button to “`This PC`” and choosing the option “`Manage`”. With right click on “`Apache Tomcat 9.0 Tomcat9`”, the server can be started or stopped. The other possibility to start or stop Tomcat is the use of the batch files saved in the Tomcat `bin` directory. The “`startup`” starts Tomcat, the “`shutdown`” stops the server.

If the server is working, the main site of Tomcat can be accessed in browser with the following url: *http://localhost:8080/*



Apache Tomcat 9.0 Tomcat9

Stop the service
Restart the service

Description:
Apache Tomcat 9.0.22 Server - https://tomcat.apache.org/

| Name | Description | Status |
|---|---|---|
| ActiveX Installer (AxInstSV) | Provides Use... | |
| AdaptiveSleepService | AMD adapti... | Running |
| Adobe Acrobat Update Servi... | Adobe Acro... | Running |
| Agent Activation Runtime_7... | Runtime for ... | |
| AllJoyn Router Service | Routes AllJo... | |
| AMD External Events Utility | | Running |
| Apache Tomcat 9.0 Tomcat9 | Apache Tom... | Running |
| App Readiness | Gets apps re... | |
| Apple Mobile Device Service | Provides the... | Running |

*Figure 4 - Start, restart or stop the Tomcat in the Computer Management Window*

If the JSPs are not working, the reasons are possibly the `environment variables`. The `environment variables` define where different programs are installed. If they are not defined exactly, Tomcat might not be able to find other programs, on which it depends.

### 2.2.1 Environment Variables

The settings of `environment variables` can be accessed by clicking with right click to "`This PC`" and choosing "`Properties`" option.

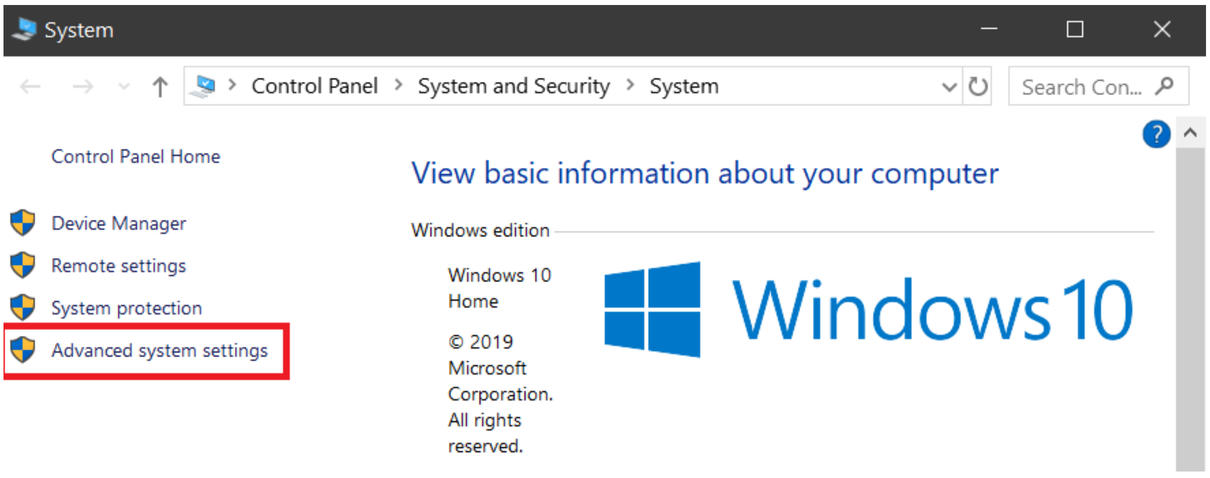
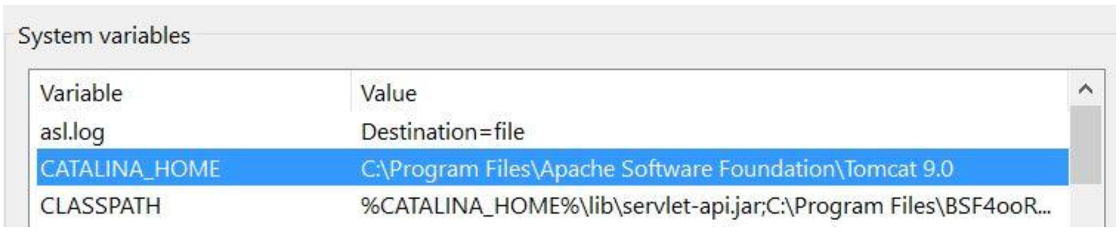


*Figure 5 - Opening advanced system settings to change the environment variables*

Here the advanced system settings should be chosen, which opens the "`System Properties`" window. Then the `environment variables` option should be chosen. For the operation of Tomcat, the correct settings of the following environment variables are necessary:

1. `CATALINA_HOME`: The `CATALINE_HOME` environment variable is where the location of Tomcat should be defined.



System variables

| Variable | Value |
|---|---|
| asl.log | Destination=file |
| CATALINA_HOME | C:\Program Files\Apache Software Foundation\Tomcat 9.0 |
| CLASSPATH | %CATALINA_HOME%\lib\servlet-api.jar;C:\Program Files\BSF4ooR... |

*Figure 6 - The CATALINA_HOME system variable*

2. `CLASSPATH:` In the CLASSPATH environment variable the location of the following jar (collection of Java classes) files should be given:
    a. The servlet-api.jar: This includes the necessary Java classes for a servlet.
    b. The `%JYTHON_HOME%\jython.jar` is the location of the Java classes of Jython. In the `JYTHON_HOME` environment variable should be defined the location of Jython.
    c. The BSF4ooRexx's location is `C:\Program Files\BSF4ooRexx\bsf4ooRexx-v641-20191126-bin.jar`.
    d. `C:\Program Files\BSF4ooRexx\jni4net.j-0.8.8.0.jar`: This a Jar file allowing the connection between Java and .NET.
    e. `C:\Program Files\BSF4ooRexx\oorexx.net.jar`
    f. `%BSF_HOME%\build\lib\bsf.jar`: The `BSF_HOME` is another environment variable, which defines the location of the Apache BSF.

By clicking "`edit`" the environment variable can be changed.

### 2.2.2 Tomcat Web Application Manager

The `Tomcat Web Application Manager` is automatically installed with Tomcat. It allows to deploy, undeploy, or restart a web application. It gives more information about the server and applications. The manager can be opened via the url: *http://localhost:8080/manager/html*.

For security reasons username and password are necessary. To define the username and password the `C:\Program Files\Apache Software Foundation\Tomcat 9.0\conf\ tomcat-users.xml` must be edited. The following lines must be included in the xml:

```
<role rolename="manager-gui"/>
<user username="user" password="1234" roles="manager-gui"/>
```

*Figure 7 - tomcat-users.xml*

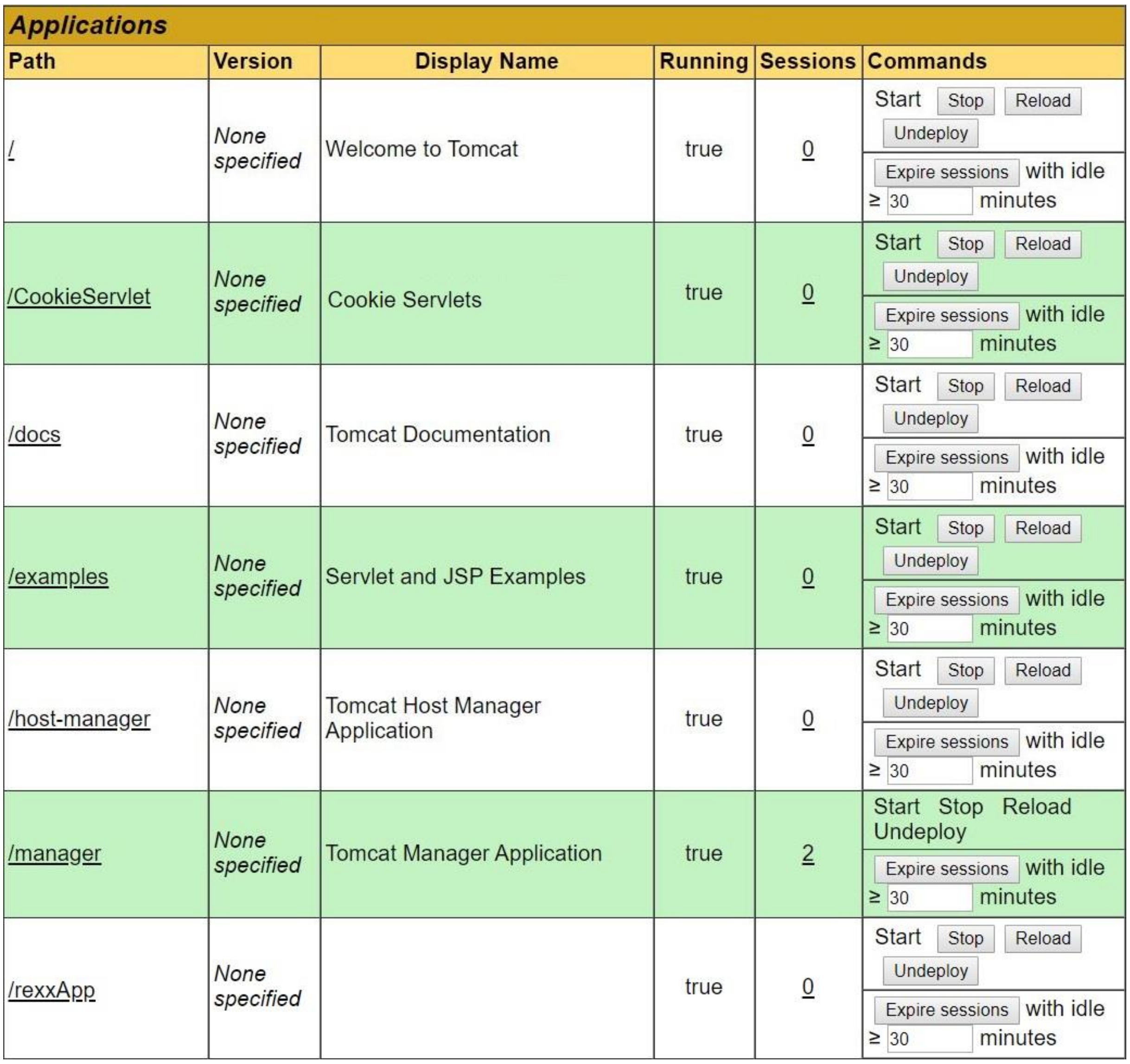
After editing the `tomcat-users xml`, the `Tomcat Web Application Manager` can be used with the `username: user` and `password: 1234`.

All web applications are listed in the `Tomcat Web Application Manager`. The path shows how to reach the applications via browser. The link starts with *http://localhost:8080/* and continues with the path defined for each web application. The `Display Name` can be defined or changed in the `web.xml` file of each project with the tag `display-name`:

```
<display-name>This is how to define a display name</display-name>.
```

*Figure 8 - Defining the Display Name*

The `Running` column shows if the application is available. The `Sessions` column show how many sessions exist. At the `Commands` column there are possibilities to start, stop, reload, undeploy a web application.



| Applications | | | | | |
|---|---|---|---|---|---|
| Path | Version | Display Name | Running | Sessions | Commands |
| / | None specified | Welcome to Tomcat | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /CookieServlet | None specified | Cookie Servlets | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /docs | None specified | Tomcat Documentation | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /examples | None specified | Servlet and JSP Examples | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /host-manager | None specified | Tomcat Host Manager Application | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /manager | None specified | Tomcat Manager Application | true | 2 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |
| /rexxApp | None specified | | true | 0 | Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes |

*Figure 9 - Tomcat Web Application Manager*

# 3. Cookie

Cookies can store small amount of data. They help the server to recognize the client.

## 3.1 Introduction to Cookies

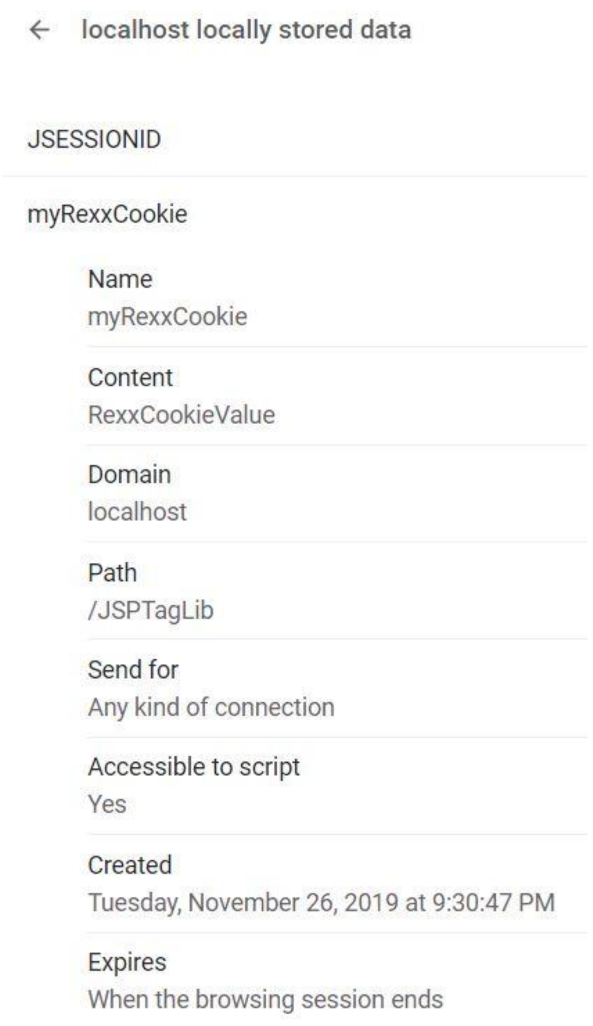
“Cookies are small files, which are stored on a user's computer. They are designed to hold a modest amount of data specific to a particular client and website, and can be accessed either by the web server or the client computer. This allows the server to deliver a page tailored to a particular user, or the page itself can contain some script which is aware of the data in the cookie and so is able to carry information from one visit to the website (or related site) to the next.” [11]

“There are two different types of cookies - session cookies and persistent cookies. If a cookie does not contain an expiration date, it is considered a session cookie. Session cookies are stored in memory and never written to disk. When the browser closes, the cookie is permanently lost from this point on. If the cookie contains an expiration date, it is considered a persistent cookie. On the date specified in the expiration, the cookie will be removed from the disk.” [12]

The cookies are saved on the client’s computer. The cookies in Google Chrome can be seen in the settings. At the settings the advanced settings must be open. After clicking “privacy and security”, “site settings”, “Cookies and site data”, “See all cookies and site data” should be chosen. In the search field on the right upper corner, the cookies can be filtered and by typing “localhost”, all the cookies created by these program examples can be seen.

By checking the cookies in browser, the proper functioning of the program example can be confirmed. The path, content, domain, time of creation and expiry can be seen.

In case of session cookies in expiry it says: “When the browsing session ends”, for persistent cookies the exact date of expiration. The cookies can be deleted here easily.



localhost locally stored data

JSESSIONID

myRexxCookie

Name
myRexxCookie

Content
RexxCookieValue

Domain
localhost

Path
/JSPTagLib

Send for
Any kind of connection

Accessible to script
Yes

Created
Tuesday, November 26, 2019 at 9:30:47 PM

Expires
When the browsing session ends

*Figure 10 - Cookies in Google Chrome*

## 3.2 Functioning of a Cookie

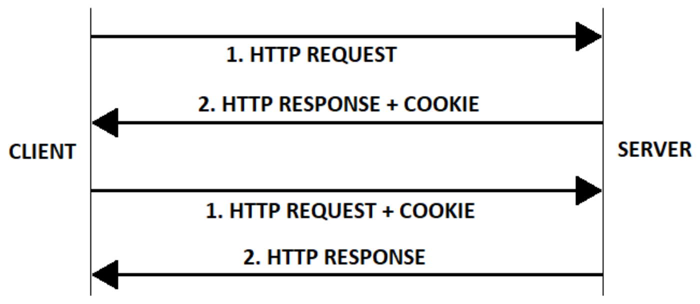


*Figure 11 - How are cookies created?*

The clients by opening a website for the first time send a request to the server. The server creates a cookie and sends it to the client. This cookie is saved on the client’s computer.

If the client already has a cookie from the website, this cookie is sent automatically to the server. The server sends a confirmation to the user.

## 3.3 The Cookie object

The constructor is a special method for creating Java object instances. The attributes given in the constructor must be specified. A cookie object can not be created without a `name` and a `value`. The `name` can not be an empty string and can not contain white spaces and after the instantiation can not be changed. In the following examples the white spaces are not checked.

A cookie object has the following attributes: `comment, domain, maxAge, name, path, secure, value, version`. These attributes can be accessed by the getters and setters.

# 4. JSP (Java Servlet Pages)

"Java Server Pages (JSP) is a Java standard technology that enables you to write dynamic, data-driven pages for your Java web applications." [13]

"A simple JSP page (.jsp) consists of HTML markup embedded with JSP tags. When the file is processed on the server, the HTML is rendered as the application view, a web page. The embedded JSP tags will be used to call server-side code and data." [13]

There is no need to programmatically compile, because the JSP is compiled in a special manner: the java code is compiled to a "servlet" to provide dynamic data. After changes made in the JSP, the code is compiled automatically to java code. The HTML remains the static data. This provides a possibility for easy modification, because if a file is changed, it is enough to refresh the browser; no need to recompile manually or to restart the server.

# 5 Taglib

„The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The Taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page." [14]

In this seminar paper two different Taglibs are used. A Taglib allows to create custom tags. Between these tags it is possible to use scripting languages.

## 5.1 JSR-223 Taglib

The JSR-223 Taglib can be downloaded from: [8]

From here the script-taglib.tld and In the lib directory, the script-taglib.jar file should be downloaded and copied in the project, as in 8.1 File Structure.

In the project the web.xml should be defined as follows:

```
<webapp>
 <jsp-config>
  <taglib>
   <taglib-uri></taglib-uri>
   <taglib-location>/WEB-INF/script-taglib.tld</taglib-location>
  </taglib>
 </jsp-config>
</webapp>
```

*Figure 12 - web.xml for JSR-223 Taglib*

## 5.2 BSF Taglib

The BSF Taglib can be downloaded with SVN from: [15]

A JAR File must be created of this source code with Apache Ant.

### 5.2.1 Apache Ant

“Apache Ant is a Java library and command-line tool whose mission is to drive processes described in build files as targets and extension points dependent upon each other. The main known usage of Ant is the build of Java applications.” [16]

This is necessary to build `jar` files from Taglibs and of the BSF in case of Jython. Apache Ant, version 1.10.7 can be downloaded from: [17]

The `environment variables` must be set. As `ANT_HOME` the location of Apache Ant must be given. Also the `Path` environment variable the “`%ANT_HOME%\bin`” must be given. (How to edit environment variables is in chapter 2.2 The Installation of Tomcat)

First the `built.xml` must be changed based on [18] :

```
<?xml version="1.0"?>
<project name="bsf" default="main">
	<property name="bin.dir" location="bin"/>
	<property name="src.dir" location="src"/>
	<property name="dist.dir" location="dist"/>
	<property environment="env"/>
	<property name="servlet.jar" value="${env.JAVA_HOME}/lib/missioncontrol/
plugins/javax.servlet_3.0.0.v201112011016.jar"/>
	<property name="jsp.jar" value="${env.JAVA_HOME}/lib/missioncontrol/
plugins/javax.servlet.jsp_2.2.0.v201112011158.jar"/>
```

```
       <property name="bsf.jar" value="${env.BSF_HOME}/build/lib/bsf.jar"/>
       <property name="classpath" value="${servlet.jar}:${bsf.jar}:${jsp.jar}"/>
       <property name="checkRequirements.pre" value="checkRequirements.pre"/>
       <property name="examples.pre" value="examples.pre"/>
       <target name="main">
            <mkdir dir="${bin.dir}"/>
            <mkdir dir="${dist.dir}"/>
            <javac srcdir="${src.dir}" destdir="${bin.dir}"
                  classpath="${classpath}" verbose="yes" source="1.4"/>
            <jar jarfile="${dist.dir}/bsf-taglib.jar" basedir="${bin.dir}"/>
       </target>
</project>
```

*Figure 13 - built.xml*

The location of `servlet.jar`, `jsp.jar`, `bsf.jar` might be different. The BSF Taglibs should be opened in the terminal. The building can be started with the command: `ANT MAIN`. The `bsf-taglib.jar` is put in the `dist` directory of the BSF Taglib. The `bsf-taglib.jar` should be copied and put in the Tomcat `WEB-INF` directory of the project. This `jar` file contains 2 compiled java classes, the `scriptlet.class` and the `expression.class`.

At the beginning of the JSP the attributes of the taglib `directive` must be set. These attributes are the `uri` and the `location`, which is the location in file system for the `taglib.tld`. If the directory structure is organized as in the examples below it is always: `taglib-location="/WEB-INF/taglib.tld"`. The `prefix` is used to define the taglib tags `(prefix="bsf").` In this example it is set to bsf, so each taglib tag starts with "`<bsf: … >`".

### 5.2.2 Settings for the BSF Taglib in web.xml

For Jython and for Rexx cookie, the BSF Taglib is used. In this case `web.xml` file of the project must be changed. In both example this `web.xml` file is put in the `WEB-INF` directory. The `jsp-config` should be made as follows:

```
<webapp>
 <jsp-config>
  <taglib>
   <taglib-uri>http://jakarta.apache.org/taglibs/bsf-2.0</taglib-uri>
```

```
        <taglib-location>/WEB-INF/taglib.tld</taglib-location>
      </taglib>
    </jsp-config>
  </webapp>
```

*Figure 14 - web.xml for BSF Taglib*

### 5.2.3 Tag Library Descriptor (.tld)

To create the `taglib.tld` the below code must be put in a file and saved as `taglib.tld`.

The `taglib.tld` file is also placed in the `WEB-INF` directory. This file defines that `scriptlet` (line 17) and `expression` (line 28) tags are forwarded to the `taglibs`. The `language` attribute is set as required (lines 22-23 and lines 33-34), which means by each starting tag the language must be given.

```
1   <?xml version="1.0" encoding="ISO-8859-1" ?>
2   <!DOCTYPE taglib
3       PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
4       "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
5
6   <!-- a tab library descriptor -->
7
8   <taglib>
9
10      <tlibversion>2.0</tlibversion>
11      <jspversion>1.1</jspversion>
12      <shortname>BSF JSP Support</shortname>
13      <uri>http://jakarta.apache.org/taglibs/bsf-2.0</uri>
14      <info> Just testing </info>
15
16      <tag>
17          <name>scriptlet</name>
18          <tagclass>org.apache.taglibs.bsf.scriptlet</tagclass>
19          <bodycontent>tagdependent</bodycontent>
20          <info>Run script</info>
21          <attribute>
22            <name>language</name>
23            <required>true</required>
24          </attribute>
25      </tag>
```

```xml

    <tag>
        <name>expression</name>
        <tagclass>org.apache.taglibs.bsf.expression</tagclass>
        <bodycontent>tagdependent</bodycontent>
        <info>Run expression</info>
        <attribute>
          <name>language</name>
          <required>true</required>
        </attribute>
    </tag>

</taglib>
```

*Figure 15 - Tag Library Descriptor*

# 6. Jython Cookie in JSP

The Jython is a scripting language, which allows with Python programming language to create Java applications. The 2.5.3 version of Jython is recommended. This can be downloaded from: [19]

## 6.1 File Structure

The `JythonTagLib` is created in `C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`. The `bsf.jar` for Jython, the bsf-`taglib.jar` and the `jython.jar` are in the `WEB-INF\lib` directory.
The `JythonCookie.jsp` is saved in the `JythonTagLib`.
The `JythonCookie.jsp` can be accessed in browser via the following link:
*http://localhost:8080/JythonTaglib/JythonCookie.jsp*

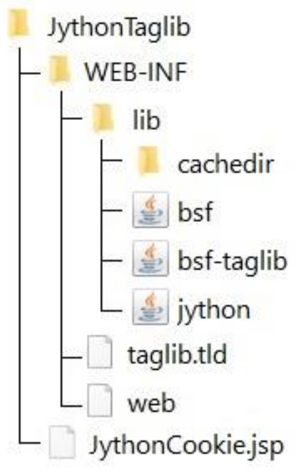


*Figure 16 - Directory structure for Jython*

## 6.2 The BSF

In the `bsf` for Jython a small change must be done. The `JythonEngine.java` is placed in the BSF directory `\commons-bsf-master\src\main\java\org\apache\bsf\engines\jython\JythonEngine.java`. In this Java class the `PyJavaInstance` object must be replaced by the `PyInstance` object. This object in the BSF exists only in the `JythonEngine.java`. This change must be done, because the BSF had been written for older versions of Jython.

In the following code parts of the `JythonEngine.java` are shown, which included the `PyJavaInstance` object.

```
                                                ...
import org.apache.bsf.BSFDeclaredBean;
import org.apache.bsf.BSFException;
import org.apache.bsf.BSFManager;
import org.apache.bsf.util.BSFEngineImpl;
import org.apache.bsf.util.BSFFunctions;
import org.python.core.Py;
import org.python.core.PyException;
import org.python.core.PyInstance;
import org.python.core.PyObject;
import org.python.core.PySystemState;
import org.python.util.InteractiveInterpreter;

                                                ...

              if (result != null && result instanceof PyInstance)
                 result = ((PyInstance)result).__tojava__(Object.class);
              return result;

                                                ...

              if (result != null && result instanceof PyInstance)
                 result = ((PyInstance)result).__tojava__(Object.class);
              return result;
```

*Figure 17 - JythonEngine.java*

After changing the `JythonEngine.class` a `jar` file should be created with Apache Ant. Details and download information about Ant can be found in chapter 5.2.1 Apache Ant.

The jar file of the BSF is created as in case of the Taglibs. In the terminal the `BSF\commons-bsf-master` should be opened and a `jar` file is created with the `ANT ALL` command.

The jar file is put in the "`dist`" directory. It should be copied and put in the `JythonTaglib/WEB-INF/lib` directory.

## 6.3 Jython Cookie in JSP with BSF Taglib

The JythonCookie.jsp is divided into different parts upon the functionalities (creating, modifying, deleting and listing the cookies).

### 6.3.1. Creating a Cookie

This part includes the necessary settings and imports for the JSP, the HTML header and the HMTL `form` and the Jython code for creating a cookie.

```

<%@ page language="java" contentType="text/html"%>
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="bsf"%>


<html>
<head>
     <title>JYTHON COOKIE</title>
</head>
<body>
<h3>Create your own Cookie</h3>
<form method=POST>
     <input type="text" name="cookieName" value="">
     <input type="text" name="cookieValue" value="">
     <input type="submit" value="OK">
</form>

     <bsf:scriptlet language="jython">

# CREATING COOKIE

from javax.servlet.http import Cookie

cookieName = request.getParameter("cookieName")         #name of the cookie from request
cookieValue = request.getParameter("cookieValue")       #value of the cookie from request
if cookieName is not None and cookieValue is not None:  #checking if parameters are null
     if len(cookieName) > 0:                            #checking if name is empty
         # creating the cookie
         cookie = Cookie(cookieName, cookieValue)
         # adding cookie to response
         response.addCookie(cookie)
         print >> out, "Cookie created! name: %s value: %s" % (cookie.name, cookie.value)
     else:
         print >> out, "Cookie could not be created!"
else:
     print >> out, ""
```

```
    </bsf:scriptlet>

```

*Figure 18 - JythonCookie.jsp 1/4*

Each JSP starts with defining the default `language` and `contentType` (line 2). In line 3 the `taglib-uri` (the location of the `taglib.tld`) and `prefix` are defined as follows: `<%@ taglib uri="/WEB-INF/taglib.tld" prefix="bsf"%>`.

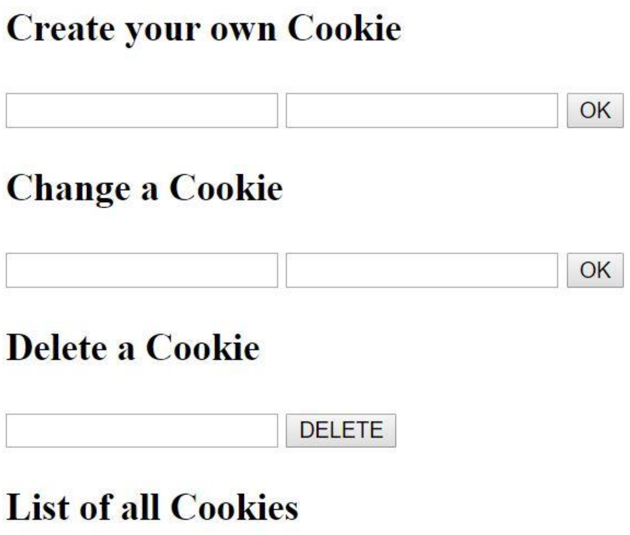


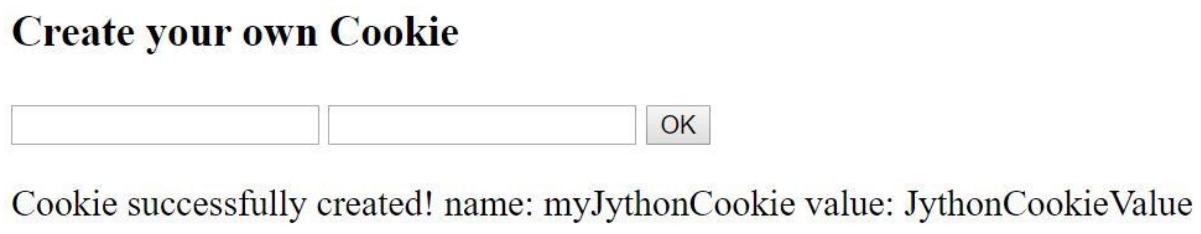
*Figure 19 - Jython cookie in browser*

The user interface is designed in JSPs with HTML code. The `title` tag in `header` sets the title of the tab. The `body` part contains different elements. The `<h3>` tag creates a heading in the browser. (`<h1>, <h2>` would create a text with bigger text size). Each HTML starting tag must have a closing tag (`</h3>`). A `form` allows to get user input. The `method` of the form should be set to `POST`. In the form the `input type=”text”` means, a text field is shown in browser. The `name` defines the parameter name, the inputs can be accessed via this parameter `name`. The `input type=”submit”` creates a button. After pushing this button the parameters are sent to the server.

To allow Jython code on JSPs the Taglib starting tag should be used. It contains the `prefix` and the `language` attribute, which is set to “`jython`”.

To create a cookie in Jython the Cookie object must be imported with: “`from javax.servlet.http import Cookie`”.

From the Jython code, we can get the given parameters by the user with `request.getParameter(“cookieName”)` and `request.getParameter (“cookieValue”)`. It must be checked, if the arguments are null unless a Null Pointer Exception occurs (line 26). The cookie must have a name, so it needs to be checked if it is an empty string. (line 27). The cookie object is created in line 29: `name` and `value` must be used in the constructor. In order to save the cookie in the browser of the client, it must be added to the response (line 31). On success “`Cookie created! name: cookieName and value: cookieValue`” is shown in browser. If the creation of the cookie was not successful, the message “`Cookie could not be created`” is shown.


Create your own Cookie

Cookie successfully created! name: myJythonCookie value: JythonCookieValue

*Figure 20 - Jython cookie created!*

A session cookie is created in this example, because no “`MaxAge`” was set. At the `browser settings – all cookies and site data`, the created cookies are listed (see chapter 3.1 Introduction to Cookies). In case of session cookies, the `expiry` says: “`When the browsing session ends`”. This means that if the browser is closed, the session cookies are automatically deleted.

In order to create a persistent cookie, the `maxAge` attribute must be added. It changes the `expiry` to an exact date, which can be seen in the `settings – all cookies and site data`. The browser can be closed, the cookies are stored until the expiry date. For setting the `maxAge` attribute an integer value is needed. This integer value defines the lifetime of the cookie in seconds. For example `cookie.setMaxAge(3600)` means that this cookie exists for one hour. The same method is used for deleting a cookie – look at line 78.

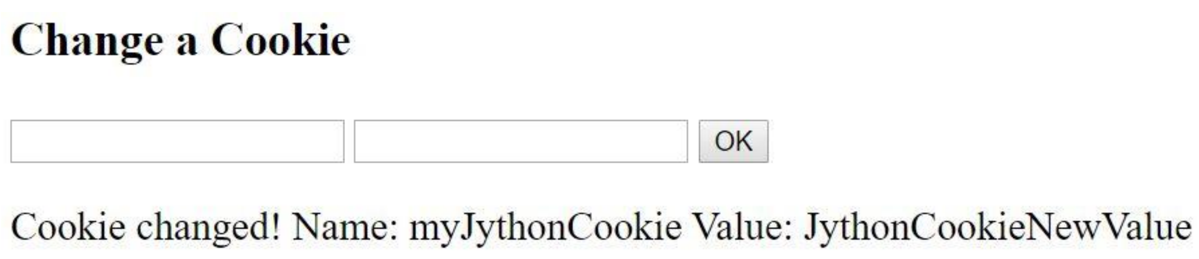
### 6.2.2 Changing a Cookie

The HTML `form` and the Jython code for changing a cookie is as follows:

```
39     <h3>Change a Cookie</h3>
40     <form method=POST>
41        <input type="text" name="cookieModifyName" value="">
42        <input type="text" name="cookieModifyValue" value="">
43        <input type="submit" value="OK">
44     </form>
45
46     <bsf:scriptlet language="jython">
47
48 # MODIFYING COOKIE
49
50 cookieName = request.getParameter("cookieModifyName")       #cookie name of request
51 if request.cookies is not None and cookieName is not None:  #null-check
52     if len(cookieName) > 0:                                 #checking if name is empty
53        for c in request.cookies.tolist():                   #finding the cookie, which
54           if c.name == cookieName:                          #equals to cookieModify
55              c.value = request.getParameter("cookieModifyValue")     #setting value
56              response.addCookie(c)                          #adding cookie to response
57              print >>out, "Cookie changed! Name: %s Value: %s" % (c.name, c.value)
58        print >>out, "Something went wrong. Cookie could not be changed."
59 else:
60     print >>out, ""
61     </bsf:scriptlet>
62
```

*Figure 21 - JythonCookie.jsp 2/4*

For changing a cookie, another form for `POST` request is made (lines 40-44). First on the parameter, `name` of the cookie given by the client and on the list of all the cookies a `null-check` should be made. The `null-check` in line 51 is important to avoid the Null Pointer Exceptions. A cookie must have a name, it can not be empty, so the length of the `cookieName` must be bigger than 0 (line 52). With the help of a `for` cycle it is checked if any element in the `request.cookies.toList()` equals `cookieName` (line 53). If they are the same, a new `value` is set for the cookie (line 55). By adding the cookie with the new value to the `response`, the `response` is sent to the client (line 56) and in browser success message is shown (line 57). If something went wrong, the message “`Something went wrong. Cookie could not be changed`” is shown.


Change a Cookie

Cookie changed! Name: myJythonCookie Value: JythonCookieNewValue

*Figure 22 - Jython cookie changed!*

## 6.2.3 Deleting a Cookie

In the following, a HTML and a Jython code for deleting a cookie is shown.

```
63      <h3>Delete a Cookie</h3>
64      <form method=POST>
65           <input type="text" name="cookieDelete" value="">
66           <input type="submit" value="DELETE">
67      </form>
68
69      <bsf:scriptlet language="jython">
70
71 # DELETING COOKIE
72
73 cookieName = request.getParameter("cookieDelete")         #name of the cookie
74 if request.cookies is not None and cookieName is not None:  #check if parameters are null
75      if len(cookieName) > 0:                              #checking if name is empty
76           for c in request.cookies.tolist():                #finding the cookie, which
77                if c.name == cookieName:                    #equals to cookieDelete
78                     c.maxAge = 0                           #setting max age to 0
79                     response.addCookie(c)                  #adding cookie to response
80                     print >>out, "Cookie, %s has been deleted!" % c.name
81           print >>out, "Something went wrong. Cookie could not be deleted."
82 else:
83      print >>out, ""
84      </bsf:scriptlet>
```

*Figure 23 - JythonCookie.jsp 3/4*

In the `form` for deleting a cookie, only the name is necessary. The `request.cookies`, which contains all cookies from request, and the `name` of the cookie must be checked if null. The `name` also needs to be checked if empty, with `len(cookieName) > 0`. The same way as by changing a cookie, the `for` cycle is used to find the element, which has the same `name` as the client given parameter (lines 76-77). The attribute `maxAge` can be used to define an expiry date. To delete a cookie the `maxAge` attribute is set to 0 (line 78). By adding this modified cookie to the response, the client's browser deletes it, because of the expiry date (line 79). Success or error message is shown in browser (line 80 or line 81).

### 6.2.4 Show the List of Cookies

The following code is the last part of the `JythonCookie.jsp`. It shows the list of the existing cookies in the browser.

```
85
86      <h3>List of all Cookies</h3>
87      <bsf:scriptlet language="jython">
88
89  # LIST COOKIES
90
91  str = ""
92  if request.cookies is not None:                    #check if cookies array is
                                                        null
93      for c in request.cookies.tolist():              #create a string from all
                                                        items
94          str += "Name: %s - Value: %s<br/>" % (c.name, c.value)
95  print >>out, str
96      </bsf:scriptlet>
97
98      </body>
99  </html>
```

*Figure 24 - JythonCookie.jsp 4/4*

In line 86 a heading is created. The `request` contains the list of the all `cookies` created, no client given parameter is necessary, so no HTML is needed. The null-check of the `request.cookies`, is important (line 92). An empty string is created in line 91. To this string during the `for` cycle the `name` and `value` of the cookie is added. The `<br/>` is a HTML tag, that makes sure each cookie is shown in a new line in the browser. This string is shown in browser by `print >>out, str.` This is the last part of the `JythonCookie.jsp`, so the HTML tags should be closed (lines 98-99).

# 7. Rexx Cookie in JSP with BSF Taglib

In this example the Apache BSF Taglib is used with the Rexx scripting language to create an application.

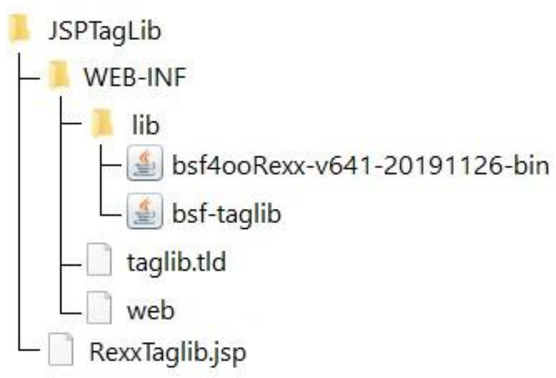
## 7.1 File Structure

The project, `JSPTagLib` is created in `C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`. The `BSF4ooRexx` and the `bsf-taglib` are in the `WEB-INF\lib` directory.

The JSP files are saved in the `JSPTagLib`.

The `RexxTaglib.jsp` can be accessed in browser via the following link:

*http://localhost:8080/JSPTagLib/RexxTaglib.jsp*



```
JSPTagLib
├─ WEB-INF
│  ├─ lib
│  │  ├─ bsf4ooRexx-v641-20191126-bin
│  │  └─ bsf-taglib
│  ├─ taglib.tld
│  └─ web
└─ RexxTaglib.jsp
```

*Figure 25 - File structure for Rexx Cookie with BSF Taglib*

### 7.2 RexxTaglib.jsp

The JSP is divided into different parts upon the functionalities (creating, modifying, deleting and listing the cookies).

### 7.2.1. Creating a Cookie

This part includes the necessary settings and imports for the JSP, the HTML header and the HMTL `form` and the Rexx code for creating a cookie.

```

<%@ page language="java" contentType="text/html"%>
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="bsf"%>


<html>
    <head>
       <title>TAGLIB TO ENABLE REXX</title>
    </head>
    <body>
       <h3>Create your own Cookie</h3>
       <form method=POST>
            <input type="text" name="cookieName" value="">
           <input type="text" name="cookieValue" value="">
            <input type="submit" value="OK">
       </form>

<bsf:scriptlet language="rexx">

-- The arguments cannot be easily taken over in taglibs with the command:
-- use arg out, request, response

--GET ARGUMENTS
out = bsf.lookupBean('out')                          -- get the out object
response = bsf.lookupBean('response')                -- get the response object
request = bsf.lookupBean('request')                  -- get the request object

--CREATE COOKIE
str = ""                                             -- creates an empty string
cookieName = request~getParameter("cookieName")      -- name of the cookie from request
cookieValue = request~getParameter("cookieValue")   -- value of the cookie from request
if (cookieName \= .nil & cookieValue \= .nil) then   -- checking if parameters are null
    Do
       if cookieName <> "" then                      -- checking if name is empty
          Do
          -- creating the cookie with bsf
          cookie =  .bsf~new("javax.servlet.http.Cookie", cookieName, cookieValue)
          --adding cookie to response
          response~addCookie(cookie)
          str = "Cookie successfully created! name: " cookie~name "value: " cookie~value
          End
       else str = "Cookie could not be created!"
    End
out~println(str)                                     -- shows the string in browser

::requires BSF.cls                                   -- loads BSF4ooRexx support

</bsf:scriptlet>
```

*Figure 26 - RexxTaglib.jsp 1/4*

The language of the JSP must be set to Java (line 2). The `location` of the `taglib.tld` and the `prefix` is defined in line 3.

After the necessary imports the first part of the HTML is written. Using the `<form method=POST>` we can define a POST request, with the parameters `cookieName` and `cookieValue`.

To enable the execution of Rexx code on Tomcat, the Taglib is necessary. At the beginning of the Rexx code, in the Taglib opening tag the `language` must be set to `rexx`. (line 18). At the end of the Rexx code the Taglib closing tag is put (line 48).

The execution of a Rexx program starts by checking the requirements. In this case in line 46, the requirement is the `BSF4ooRexx` (`::REQUIRES BSF.CLS`). This means that at first the `BSF4ooRexx` is imported and this class can be used for creating Java objects in Rexx. The execution of the Rexx code goes line by line from the 1st to the last line, unless another instruction is given.

Usually Rexx can take over arguments from java with the statement "`use arg`". In case of Taglibs, this command does not work and the `bsf.lookupBean("out")` command should be used (lines 24-26). The `out` argument is the `JSPWriter`, which allows to show information in the browser. The `request` and `response` allow the communication between the server and client.

In line 29 an empty String is declared. If the server does not have any specific message for the client this empty String is shown.

The request object in the HTML form defined parameters, `cookieName` and `cookieValue` (lines 13-14). The name of the cookie given in browser by the client can be accessed by `request~getParameter("cookieName")`, and the value with `request~getParameter("cookieValue")`.

It can not be guaranteed that the client given parameters are not null or empty. For controlling the null or empty value of an object, more steps are necessary. First of all, it has to be checked if the `cookieName` or `cookieValue` are `.nil` (line 32). Every time the page is reloaded, and no arguments are given, these parameters are empty. A cookie can take an empty string as value, but the name can not be empty (line 34). If the name is empty, the Rexx code shows the message in browser: “`Cookie could not be created`” (line 42) To create a cookie object with Rexx, the `BSF4ooRexx` needs to be used. To create a Java object in Rexx the fully qualified name of the object must be declared (“`javax.servlet.http.Cookie`”) and the constructor (name, value) must be used for creating the object (line 37). The cookie must be sent to the client by adding it to the `response` (line 39). If all these happened successfully the response is shown in browser: “`Cookie successfully created! Name: name value: value`”.

## 7.2.2 Changing a Cookie

The HTML `form` and the Rexx code for changing a cookie is as follows:

```
49
50          <h3>Change a Cookie</h3>
51          <form method=POST>
52              <input type="text" name="cookieModifyName" value="">
53              <input type="text" name="cookieModifyValue" value="">
54              <input type="submit" value="OK">
55          </form>
56
57  <bsf:scriptlet language="rexx">
58
59  --GET ARGUMENTS
60  out = bsf.lookupBean('out')                            -- get the out object
61  response = bsf.lookupBean('response')                  -- get the response object
62  request = bsf.lookupBean('request')                    -- get the request object
63
64  --CHANGE A COOKIE
65  str = ""                                               -- creates an empty string
66  cookieName = request~getParameter("cookieModifyName")  --get cookie name from request
67  if (request~COOKIES \= .nil & cookieName \= .nil) then do  -- null check
68      if cookieName <> "" then do                        --checking if name is empty
69          listCookies = request~COOKIES                  --get cookies from request
70              do c over listCookies                      --finding the cookie, which
71              if c~name == cookieName then               --equals to cookieModify
72              Do
73                  c~value(request~getParameter("cookieModifyValue"))        --set value
74                  response~addCookie(c)                  --adding cookie to response
75                  str = "Cookie changed! Name: " c~name " Value: " c~value
76              End
77              else str = "Something went wrong. Cookie could not be changed."
78          End
79      End
80  End
81  out~println(str)
```

```
82
83 ::requires BSF.cls                                    -- Loads BSF4ooRexx support
84
85 </bsf:scriptlet>
86
```

*Figure 27 - RexxTaglib.jsp 2/4*

A HTML `form` is created to change the values of cookies (lines 50-55).

Another Taglib `scriptlet` is started and the language is set to `rexx`. (line 57). The arguments are retrieved (lines 59-62) with the `bsf.lookup` method as in case of creating a cookie.

For identifying a cookie its `name` is used. Two cookies can not exist if the `name`, `path` and `domain` are the same. All the cookies in this example are created with the same `path` and `domain`, so only the `name` needs to be controlled. This allows changing the cookie and not creating a new one. The `request` contains all cookies as array list. In line 67 it must be checked if this list and the name of the cookie are `.nil`. If the name of the cookie to be changed is not empty (line 68), the `listCookies` - containing all cookies sent via the request - is created. The block (lines 72-76) starting with “`do c over listCookies`” instruction, controls each element of the `listCookies` list if the name of the cookie equals the name given by the client. In case of being equal the `value` of this cookie is changed to the client given value (line 10). After that the cookie is added to the `response`. The message is shown in browser: “`Cookie changed! Name: name Value: value`”.

## 7.2.3 Deleting a Cookie

In the following, a HTML and a Rexx code for deleting a cookie is shown.

```
87         <h3>Delete a Cookie</h3>
88         <form method=POST>
89             <input type="text" name="cookieDelete" value="">
90             <input type="submit" value="DELETE">
91         </form>
92
```

```
93  <bsf:scriptlet language="rexx">
94
95  --GET ARGUMENTS
96  out = bsf.lookupBean('out')                            -- get the out object
97  response = bsf.lookupBean('response')                  -- get the response object
98  request = bsf.lookupBean('request')                    -- get the request object
99
100 --DELETE A COOKIE
101 cookieName = request~getParameter("cookieDelete")     --name of the cookie from request
102 str = ""
103 if (request~COOKIES \= .nil & cookieName \= .nil) then do --null-check
104     if cookieName <> "" then do                        --checking if name is empty
105         listCookies = request~COOKIES                  --getting cookies from request
106         do c over listCookies
107             if c~name == cookieName then               --finding the cookie, which
108             Do                                         --equals to cookieDelete
109                 c~MaxAge = 0                           --setting max age to 0
110                 response~addCookie(c)                  --adding cookie to response
111                 str = "Cookie, " c~name "has been deleted!"
112             End
113             else str = "Something went wrong. Cookie could not be deleted."
114         End
115     End
116 End
117 out~println(str)                                       -- shows the string in browser
118
119 ::requires BSF.cls                                     -- loads BSF4ooRexx support
120
121 </bsf:scriptlet>
```

*Figure 28 - RexxTaglib.jsp 3/4*

For the request to delete a cookie an individual HTML `form` is created with the `cookieDelete` parameter.

The `request` `response` and `out` object are retrieved by the Rexx side (line 95-99). The method to delete a cookie is to the one modifying it: if any of the cookies from the list has the same `name` as the client given parameter, it must be changed. In case of changing a cookie, a new `value` is set, but to delete the `maxAge` attribute must be changed to 0 (line 109). In case of success the message "`Cookie, cookieName has been deleted!`" is shown in browser.

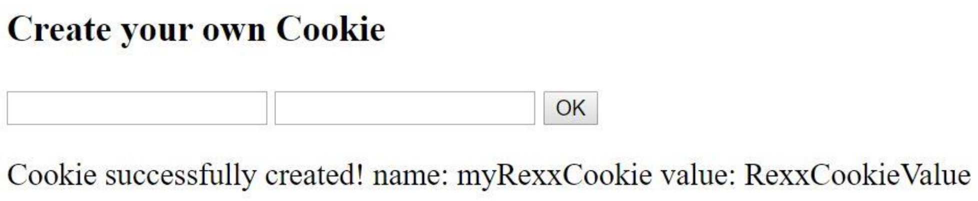
## 7.2.4 Show the List of Cookies

The following code is the last part of the `RexxTaglib.jsp`. It shows the list of the existing cookies in the browser.

```
122
123         <h3>List of all Cookies</h3>
124
125 <bsf:scriptlet language="rexx">
126
127 --GET ARGUMENTS
128 out = bsf.lookupBean('out')                         -- get the out object
129 request = bsf.lookupBean('request')                 -- get the request object
130
131 -- SHOW ALL COOKIES
132 str = ""
133 if request~COOKIES \= .nil then do                  -- null check
134     listCookies = request~COOKIES
135     do c over listCookies                           --creating a string from all items
136         str = str "Name: " c~name "- Value: " c~value "<br>"
137     End
138 End
139 out~println(str)                                    -- shows the string in browser
140
141 ::requires BSF.cls                                  -- loads BSF4ooRexx support
142
143 </bsf:scriptlet>
144     </body>
145 </html>
```

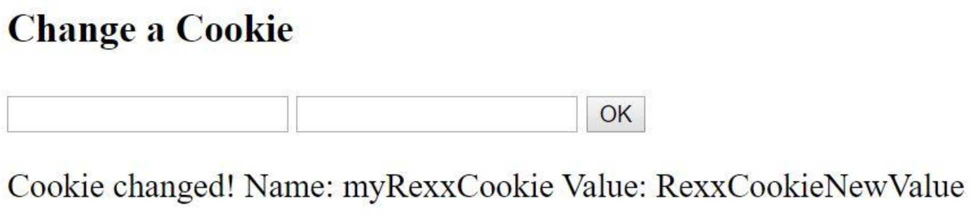
*Figure 29 - RexxTaglib.jsp 4/4*

For creating a list of all cookies, the arguments `out` and `request` are retrieved and the `null`-check on the cookie list from the client's request is made (lines 133-134). A string (str) of all elements of the list is created (lines 135-137). This string value contains the `names` and `values` of the cookies. The HTML tag "`<br>`" makes sure that in the browser each cookie is placed in a new line. This string is shown in browser by the command "`out~println(str)`" (line 139).


## Create your own Cookie

Cookie successfully created! name: myRexxCookie value: RexxCookieValue

*Figure 30 - Rexx cookie created!*


## Change a Cookie

Cookie changed! Name: myRexxCookie Value: RexxCookieNewValue

*Figure 31 - Rexx cookie modified!*

# 8. Rexx Cookie in JSP with JSR-223 Taglib

In the following example the JSR-223 Taglib is used with Rexx to create an application that creates, changes, deletes and shows cookies.

## 8.1 File Structure#

The project, `JSPJSRTagLib` should be placed in the webapps directory of Tomcat. The file structure is same as in case of the BSF Taglib (7.1 File Structure). The necessary jar files are put in the `WEB-INF/lib` directory. In case of the JSR-223 it is the jar file of the `script-taglib` and the `bsf4ooRexx`. The Tag Library Descriptor, the `script-taglib.tld` and the `web.xml` are in the WEB-INF directory. The `RexxJSRTaglib.jsp` is in the `JSPJSRTagLib.`

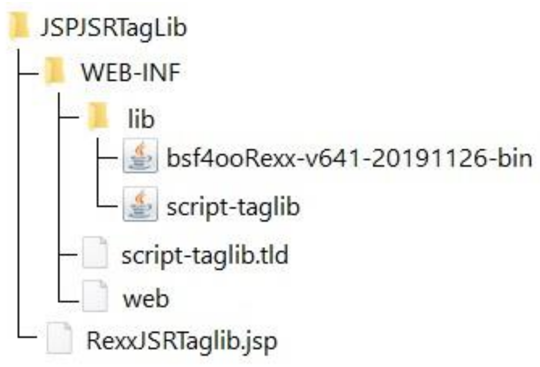


*Figure 32 - File structure for Rexx Cookie with JSR-223 Taglib*

## 8.2 RexxJSRTaglib.jsp

In this example the JSR-223 Taglib is used to create, change, delete cookie and get a list of the cookies.

### 8.2.1 Creating a Cookie

First the language of the JSP should be defined and the imports should be done. The Taglib is put in the `WEB-INF` directory. As `taglib-uri`, the `/WEB-INF/script-taglib.tld` should be given.

```

<%@ page language="java" contentType="text/html"%>
<%@ taglib uri="/WEB-INF/script-taglib.tld" prefix="bsf"%>

```

```
 5
 6 <html>
 7        <head>
 8           <title>ENABLE REXX WITH JSR-223 TAGLIB</title>
 9        </head>
10        <body>
11           <h3>Create your own Cookie</h3>
12           <form method=POST>
13                <input type="text" name="cookieName" value="">
14                <input type="text" name="cookieValue" value="">
15                <input type="submit" value="OK">
16           </form>
17
18 <bsf:scriptlet language="rexx">
19 --CREATE COOKIE
20 /* @get(out response request)*/ --GET ARGUMENTS
21
22 str = ""                                            -- creates an empty string
23 cookieName = request~getParameter("cookieName")     -- name of the cookie from request
24 cookieValue = request~getParameter("cookieValue")   -- value of the cookie
25 if (cookieName \= .nil & cookieValue \= .nil) then  -- checking if parameters are null
26        do
27           if cookieName <> "" then                  -- checking if name is empty
28                do
29                -- creating the cookie with bsf
30                cookie = .bsf~new("javax.servlet.http.Cookie", cookieName, cookieValue)
31                --adding cookie to response
32                response~addCookie(cookie)
33                str = "Cookie created! name: " cookie~name "value: " cookie~value
34                end
35           else str = "Cookie could not be created!"
36        end
37 out~println(str)                            -- shows the string in browser
38 ::requires BSF.cls
39
40 </bsf:scriptlet>
```

*Figure 33 - RexxJSRTaglib.jsp 1/4*

In the HTML part, we define a POST request, with the parameters `cookieName` and `cookieValue`.

In the Taglib opening tag the `language` attribute must be set to `rexx`. (line 18). In difference to the BSF Taglib, with the JSR-223 the arguments are get via an annotation. An annotation starts with `/*@` and ends with `*/`. The annotation, `/* @get(out response request)*/` in line 20 is important, because it gets the arguments from the JSP. Without this, the objects, out, request and response would not be accessible from the Rexx code.

The Rexx code is exact the same as in case of the BSF Taglib (see chapter 7.2.1. Creating a Cookie). The parameters are get from the `request` object (lines 23-24). Null-check and the length of the `cookieName` is checked. (lines 25-27). The cookie is created and added to the `response`. The closing tag of the JSR-223 Taglib is necessary to separate the Rexx code from the HTML.

## 8.2.2 Changing a Cookie

The HTML and the Rexx code for changing a cookie is as follows:

```
        <h3>Change a Cookie</h3>
        <form method=POST>
            <input type="text" name="cookieModifyName" value="">
            <input type="text" name="cookieModifyValue" value="">
            <input type="submit" value="OK">
        </form>

<bsf:scriptlet language="rexx">

--CHANGE A COOKIE
--GET ARGUMENTS
/* @get(out response request)*/
str = ""                                                -- creates an empty string
cookieName = request~getParameter("cookieModifyName")   --get cookie name from request
if (request~COOKIES \= .nil & cookieName \= .nil) then do     --null - check
    if cookieName <> "" then do                         --checking if name is empty
        listCookies = request~COOKIES                   --getting cookies from request
            do c over listCookies                       --finding the cookie, which
            if c~name == cookieName then                --equals to cookieModify
            do
                c~value(request~getParameter("cookieModifyValue"))   --setting value
                response~addCookie(c)                   --adding cookie to response
                str = "Cookie changed! Name: " c~name " Value: " c~value
            end
            else str "Something went wrong. Cookie could not be changed."
        end
    end
end
out~println(str)
```

```
70
71 ::requires BSF.cls                                      -- loads BSF4ooRexx support
72
73 </bsf:scriptlet>
```

*Figure 34 - RexxJSRTaglib.jsp 2/4*

With HTML a `form` for `POST` request is made. The `cookieModifyName` and `cookieModifyValue` parameters are given by the client (lines 41-46).

The objects, out, request and response are retrieved by the Rexx code via the annotation, /* @get(out response request)*/ .

The `request` contains all cookies. It must be checked if the list of the cookies or the `cookieName` is null. If the name of the cookie to be changed is not empty, the `listCookies` - containing all cookies sent via the request - is created. The block (lines 58-64) starting with do `c over listCookies` instruction, checks the elements of the `listCookies`, if the name of the cookie element equals the name given by the client. In case of being equal the `value` of this cookie is changed to the value given by the client (line 61). After that the cookie is added to the `response`. The rexxCode returns with the response: “`Cookie changed! Name: name Value: value`”.

## 8.2.3 Deleting a Cookie

In the following HTML and Rexx code, it is shown, how to delete an existing cookie:

```
74          <h3>Delete a Cookie</h3>
75          <form method=POST>
76              <input type="text" name="cookieDelete" value="">
77              <input type="submit" value="DELETE">
78          </form>
79
80 <bsf:scriptlet language="rexx">
81 --DELETE A COOKIE
82 -- GET ARGUMENTS
83 /* @get(out response request)*/
84 cookieName = request~getParameter("cookieDelete")    --name of the cookie from request
85 str = ""
86 if (request~COOKIES \= .nil & cookieName \= .nil) then do   --checking if parameters are.nil
87     if cookieName <> "" then do                      --checking if name is empty
88         listCookies = request~COOKIES                --getting cookies from request
89         do c over listCookies
90             if c~name == cookieName then             --finding the cookie, which
```

```
 91             do                                      --equals to cookieDelete
 92                 c~MaxAge = 0                        --setting max age to 0
 93                 response~addCookie(c)               --adding cookie to response
 94                 str = "Cookie, " c~name "has been deleted!"
 95             end
 96             else str = "Something went wrong. Cookie could not be deleted."
 97         end
 98     end
 99 end
100 out~println(str)                                    -- shows the string in browser
101
102 ::requires BSF.cls                                  -- loads BSF4ooRexx support
103
104 </bsf:scriptlet>
```

*Figure 35 - RexxJSRTaglib.jsp 3/4*

The HTML form for deleting a cookie is created in lines 74-78. The objects, `out`, `request` and `response` are retrieved by the annotation, `/* @get(out response request)*/`.The code is the same as in case of the BSF Taglib, see chapter 7.2.3 Deleting a Cookie. The null-check is made. The cookie to be deleted can be found in the list of the cookies by checking each element of the `listCookies`. The `maxAge` attribute of the Cookie is set to `0`.

### 8.2.4 Show the List of Cookies

The following code is the last part of the `RexxJSRTaglib.jsp`. In this code the existing cookies are shown in the browser.

```
105         <h3>List of all Cookies</h3>
106
107 <bsf:scriptlet language="rexx">
108
109 /* @get(out request)*/
110 -- SHOW ALL COOKIES
111 str = ""
112 if request~COOKIES \= .nil then do                --checking if cookies array is null
113     listCookies = request~COOKIES
114     do c over listCookies                         --creating a string from all items
115         str = str "Name: " c~name "- Value: " c~value "<br>"
116     end
117 end
118 out~println(str)                                  -- shows the string in browser
119
120 ::requires BSF.cls                                -- loads BSF4ooRexx support
```

```

</bsf:scriptlet>
    </body>
</html>
```

*Figure 36 - RexxJSRTaglib.jsp 4/4*

For creating a list of all cookies, the arguments are retrieved with the annotation and the null-check of the cookie list made (line 112). A string (`str`) of all elements of the list is created. This string value contains the names and values of the cookies. The HTML tag, `<br>` makes sure that in the browser each cookie is placed in a new line.

# 9. Conclusion

JSPs are techniques to create web content on Tomcat servers. Because of the automated compilation and easy structure of JSPs, they are recommended for beginners. Taglibs allow to create web application with scripting languages, without the knowledge of Java. Jython and Rexx are scripting languages, which are easier to learn as Java. To configure the Tomcat server with Taglibs takes a few hours, but the creation of complex web application goes much faster with scripting languages.

The codes above are just examples, there are many different scripting languages to allow the execution on Tomcat servers. In each case, it is important to find the right version of the Taglibs and the necessary BSF. This seminar paper proves that with the usage of Taglibs, the execution of scripting languages can be allowed on Tomcat. Not only Jython and Rexx can be used for creating applications on Tomcat, but also other scripting languages supported by Taglibs and BSFs.

# Literaturverzeichnis

[1] „JDK 8 download,“ 23 12 2019. [Online]. Available: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. [Zugriff am 23 12 2019].

[2] „Rexx 5 download,“ 23 12 2019. [Online]. Available: https://sourceforge.net/projects/oorexx/files/oorexx/. [Zugriff am 23 12 2019].

[3] „BSF4ooRexx download,“ 23 12 2019. [Online]. Available: https://sourceforge.net/projects/bsf4oorexx/. [Zugriff am 23 12 2019].

[4] „Apache BSF download,“ 23 12 2019. [Online]. Available: https://github.com/apache/commons-bsf. [Zugriff am 23 12 2019].

[5] „Jython download,“ 23 12 2019. [Online]. Available: https://search.maven.org/search?q=g:org.python%20AND%20a:jython-installer&core=gav. [Zugriff am 23 12 2019].

[6] „Apache Ant download,“ 23 12 2019. [Online]. Available: https://ant.apache.org/. [Zugriff am 23 12 2019].

[7] „BSF Taglib download,“ 23 12 2019. [Online]. Available: http://svn.apache.org/repos/asf/jakarta/taglibs/deprecated/bsf/trunk/. [Zugriff am 23 12 2019].

[8] „JSR-223 download,“ 23 12 2019. [Online]. Available: https://www.dropbox.com/sh/068uqxmj83dle56/AACdMtvMk2HC8HVHkJr7TJgZa?dl=0. [Zugriff am 23 12 2019].

[9] „Wikipedia - Server (computing),“ 17 12 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Server_(computing)&oldid=929416573. [Zugriff am 17 12 2019].

[10] „Tomcat download,“ 17 12 2019. [Online]. Available: https://tomcat.apache.org/download-90.cgi. [Zugriff am 17 12 2019].

[11] "What are cookies," 23 12 2019. [Online]. Available: http://www.whatarecookies.com/. [Accessed 23 12 2019].

[12] „Cisco,“ 1 10 2019. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.html.

[13] "Java World," 1 10 2019. [Online]. Available: https://www.javaworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html.

[14] „Tutorialspoint," 23 12 2019. [Online]. Available: https://www.tutorialspoint.com/jsp/taglib_directive.htm. [Zugriff am 23 12 2019].

[15] „BSF Taglib download," 17 12 2019. [Online]. Available: http://svn.apache.org/repos/asf/jakarta/taglibs/deprecated/bsf/trunk/. [Zugriff am 17 12 2019].

[16] „Apache Ant," 05 12 2019. [Online]. Available: https://ant.apache.org/.

[17] „Apache Ant download," 17 12 2019. [Online]. Available: https://ant.apache.org/bindownload.cgi.. [Zugriff am 17 12 2019].

[18] S. Ryabenkiy, *Java Web Scripting und Apache Tomcat,* 2010.

[19] „Jython download," 17 12 2019. [Online]. Available: https://search.maven.org/search?q=g:org.python%20AND%20a:jython-installer&core=gav. [Zugriff am 17 12 2019].

[20] "Oracle Docs," 1 10 2019. [Online]. Available: https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html.

[21] "Geeks for geeks," 1 10 2019. [Online]. Available: https://www.geeksforgeeks.org/java-class-file/.

[22] "Oracle Docs," 1 10 2019. [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javac.html.

[23] "Geeks for geeks," 1 10 2019. [Online]. Available: https://www.geeksforgeeks.org/difference-between-servlet-and-jsp/.

[24] „Oracle Docs," 1 10 2019. [Online]. Available: https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html.

[25] „Codesjava," 2019 12 04. [Online]. Available: https://codesjava.com/jsp-taglib-directive.

[26] "Apache," 1 10 2019. [Online]. Available: https://commons.apache.org/proper/commons-bsf/manual.html.

[27] J. 8. download, „1," 23 12 2019. [Online]. [Zugriff am 23 12 2019].

[28] J. J. download, „[1]," 23 12 2019. [Online]. Available: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. [Zugriff am 23 12 2019].