Applies to:
SAP BusinessObjects BI 4.0 SP05 or above

Summary:
This paper provides a step-by-step description on how to start developing java applications in an Eclipse environment.
This sample simply shows how to log in and log off the platform.

Some samples in .net can be found in SDN:
http://scn.sap.com/community/bi-platform/blog/2012/11/22/sap-bi-platform-restful-web-service-sdk-demos

Official documentation:

Author Bio
Christian Champault (christian.champault@sap.com) is an expert on SAP BusinessObjects Enterprise architecture, SDKs, configuration and performance optimization.

Company: SAP
Last update: December 11 2012
# Table of Contents

Document update history ........................................................................................................... 3

1 Background ............................................................................................................................... 4

1.1 Business Intelligence Platform RESTful web service SDK .................................................. 4

1.2 Web Intelligence RESTful web service SDK ......................................................................... 4

1.3 SAP Crystal Reports RESTful web service SDK ................................................................. 5

2 Set up the development environment ....................................................................................... 6

2.1 Installing Eclipse .................................................................................................................. 6

2.2 http request handling tool .................................................................................................. 6

3 Creating and configuring a new Eclipse Project ....................................................................... 7

3.1 Creating the new project ....................................................................................................... 7

3.2 New project set up ................................................................................................................ 7

3.3 Default class ........................................................................................................................ 9

4 Login and logoff ....................................................................................................................... 11

4.1 Workflow ............................................................................................................................ 11

4.2 Output ................................................................................................................................ 11

4.3 Code .................................................................................................................................. 12

Copyright .................................................................................................................................. 16
## Document update history

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Alteration Reason</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.11.2012</td>
<td>Christian Champault</td>
<td>Initial document</td>
<td>0.1</td>
</tr>
</tbody>
</table>
1 Background

The SAP RESTful web services SDK complies with the RESTful methodology. You can access the RESTful web services SDK using any programming language that supports making HTTP requests. We are going to focus in this document in programming with java.

From an architecture point of view, The WACS server is responsible for handling RESTful web service and interfacing with the platform.
Dialog with applications is made with the HTTP protocol, on XML format

RESTful web services appeared in the BOE platform starting with BI 4.0 FP3. Its model has been extended with SP5. It is planned to be extended also in future release of Business Objects, in order to replace other SDKs. There are actually several webservice available:

1.1 Business Intelligence Platform RESTful web service SDK

The Business Intelligence platform RESTful web service SDK lets you access the BI platform using the HTTP protocol. You can use this SDK to log on to the BI platform, navigate the BI platform repository, access resources, and perform basic resource scheduling. You can access this SDK by writing applications that use any programming language that supports the HTTP protocol, or by using any tool that supports making HTTP requests

1.2 Web Intelligence RESTful web service SDK

The Web Intelligence RESTful web service SDK is an API used for manipulating the following:
• manipulating Web Intelligence documents and reports
• retrieving data from a dataprovider
• retrieving a list of available universes and details of an universes
• scheduling documents

It cannot be used to edit/create SAP Web Intelligence documents.

The Web Intelligence RESTful web service SDK relies on the BI platform RESTful web services API for session management and repository access.
1.3 SAP Crystal Reports RESTful web service SDK

SAP Crystal Reports RESTful web services allow report data managed in a SAP BusinessObjects Business Intelligence Platform repository to be consumed and embedded in mobile devices and web-enabled technology. You can fetch report content in XML or JSON format, and manipulate a report using the RESTful API and OData services. RESTful web services allow you to create applications using the development language of your choice.

SAP Crystal Reports RESTful web services allow you to:

- Export a report to a number of different file types
- Retrieve report metadata.
- Get rows of data and calculations.
- Push a row of data to the report.
- Get and set report parameters.
- Retrieve data in XML or JSON format.
2 Set up the development environment

2.1 Installing Eclipse

On the developer workstation:

Download Eclipse from the Eclipse web site:

http://www.eclipse.org/

The sample in this article has been built with Indigo for web developers SR1

http://www.eclipse.org/downloads/packages/release/indigo/sr1

Unzip the downloaded file.

Eclipse is not delivered with a jdk, so you may need to download one if it not already installed. (www.java.com)

Eclipse needs to know where the kdk is located, so make sure the java jdk is in the path environment variable of the workstation.  

Eclipse is ready to run!

2.2 http request handling tool

With Indigo, you won't get necessary classes to handle http requests.  
There are different projects that implement this functionality, like Jersey project, Apache CXF, and others.

We will use the Apache HttpClient project. I used version 4.2.1.

http://hc.apache.org/

the project is a set of seven jar file we will use in our application.
3 Creating and configuring a new Eclipse Project

3.1 Creating the new project

Launch Eclipse, and create a new java project:

Name it, ie Restful01, then finish

The project is created!

3.2 New project set up

Now, we need to reference the Apache httpClient project jars in the build path, so we can use it.

Configure the build path:

Add Library
We will create a user library that will include all the Apache httpclient jars.

Click on "User Libraries..." to create our library.

In this screen, create the library “httpclient” and add the Apache httpclient project jars:

Once done, you can add it to our project:
The build path contains now the jre system library, plus ours:

"OK"

The project is still empty, but we are now ready for the coding!

3.3 Default class

Create a class into the project:

Name it : (here, Raylight)
Please note that the use of the default package is discouraged, but we don’t really mind for this sample.

The class is ready for receiving the code

```java
public class RayLight {
}
```
4 Login and logoff

4.1 Workflow

To log into the system, we need to POST at the address:

http://<servername>:6405/biprws/logon/long

the following XML:

```xml
<attrs xmlns="http://www.sap.com/rws/bip">
 <attr name="userName" type="string">myUserName</attr>
 <attr name="password" type="string">myPassword</attr>
 <attr name="auth" type="string" possibilities="secEnterprise,secLDAP,secWinAD,secSAPR3">secEnterprise</attr>
</attrs>
```

In response to this POST request, we will received in the header the logon token, (X-SAP-LogonToken parameter)

But, when making any GET request at the same address, the RESTful web service returns to the client a template of this XML.

So, instead of building an xml from scratch to log in, we will:

1 – send a GET request to http://<servername>:6405/biprws/logon/long
2 – retrieve the template XML returned by the RESTful web service
3 – modify this XML with our login information (username, password, method)
4 – make a POST request back to the web service with this modified XML
5 – retrieve the logon token from the response of that request

4.2 Output

Running the program into Eclipse you will get the following output:
4.3 Code

The code is as follow (you can copy and paste into the class). You still need to modify the login information!

```java
import java.io.BufferedInputStream;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import javax.xml.parsers.*;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;
import org.xml.sax.InputSource;
import org.apache.http.Header;
import org.apache.http.HttpHost;

public class RayLight {
    static HttpHost _target;

    public static void main(String[] args) {
        String sUserName = "Administrator";
        String sPassword = "MDP";
        String sAuth = "secEnterprise";
        String sServer = "Win2003EEx64";
        String sPort = "6405";
        // creation of an http host
        // specify the host, protocol, and port
        _target = new HttpHost(sServer, Integer.parseInt(sPort), "http");

        try {
            String sLogonToken = LogonCMS(sUserName, sPassword, sAuth);
```
public static Document loadXMLFromString(String xml) throws Exception {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    InputSource is = new InputSource(new StringReader(xml));
    return builder.parse(is);
}

public static Document renseigneNode(Document myXmlDoc, String sItemName, String sTextContent) {
    NodeList nl = myXmlDoc.getElementsByTagName("attr");
    for (int i=0;i<nl.getLength();i++) {
        Node no = nl.item(i);
        if (sItemName.equals(no.getAttributes().getNamedItem("name").getNodeValue())) {
            no.setTextContent(sTextContent);
        }
    }
    return myXmlDoc;
}

public static String recupereStringInputStream (InputStream inputStream) throws Exception{
    byte[] buffer = new byte[1024];
    String sFinal="";
    int bytesRead = 0;
    BufferedReaderInputStream bis = new BufferedReaderInputStream(inputStream);
    while ((bytesRead = bis.read(buffer)) != -1) {
        sFinal  = sFinal + new String(buffer, 0, bytesRead);
    }
    return sFinal;
}

public static String logonCMS (String sUserName, String sPassword, String sAuth) throws Exception {
    final DefaultHttpClient httpClient = new DefaultHttpClient();
    try {
        Document xmlResponse =null;
        // send get and retrieve entity
        //
        HttpGet httpGetRequest = new HttpGet("/biprws/logon/logn");
        System.out.println("executing request to " + _target);
        HttpResponse httpResponse = httpClient.execute(_target, httpGetRequest);
        HttpEntity entity = httpResponse.getEntity();
        // If the response does not enclose an entity, there is no need
        // to bother about connection release
        String sFinal="";
        if (entity != null) {
            InputStream inputStream = entity.getContent();
            sFinal = recupereStringInputStream(inputStream);
            //
            // modify the xml for the post
            //
            try {
                xmlResponse = loadXMLFromString(sFinal);
                renseigneNode(xmlResponse, "username", sUserName);
                logoffCMS(sLogonToken);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    } finally {
        httpClient.getConnectionManager().shutdown();
    }
    return sFinal;
}
ренседжнед (xmlResponse, "password", sPassword);
ренседжнед (xmlResponse, "auth", sAuth);
}
catch (Exception e) {
    e.printStackTrace();
}

// post preparation
HttpPost httpPostRequest = new HttpPost("/biprws/logon/long");

// transforms xml in String
TransformerFactory tf = TransformerFactory.newInstance();
Transformer transformer = tf.newTransformer();
transformer.setOutputProperty(OutputKeys.OmitXmlDeclaration, "yes");
StringWriter writer = new StringWriter();
transformer.transform(new DOMSource(xmlResponse), new StreamResult(writer));
String s = writer.getBuffer().toString().replaceAll("\n|\r", "");

// put the xml entity
StringEntity myEntity = new StringEntity(s);

// send post
myEntity.setContentType("application/xml");
httpPostRequest.setEntity(myEntity);
httpResponse = httpClient.execute(_target, httpPostRequest);
Header hLogonToken = httpResponse.getFirstHeader("X-SAP-LogonToken");
String sReturn = hLogonToken.getValue();
return sReturn;
}
catch (Exception e) {
    // thrown by entity.getContent();
    e.printStackTrace();
}
finally {
    // When HttpClient instance is no longer needed,
    // shut down the connection manager to ensure
    // immediate deallocation of all system resources
    httpClient.getConnectionManager().shutdown();
}
return ";
}

public static void logoffCMS(String sLogonToken) throws Exception{
    final DefaultHttpClient httpClient = new DefaultHttpClient();
    // Post preparation
    try {
        HttpPost httpPostRequest = new HttpPost("/biprws/logoff");
        httpPostRequest.addHeader("Accept", "application/xml");
        httpPostRequest.addHeader("X-SAP-LogonToken", sLogonToken);
        StringEntity myEntity = new StringEntity("");
        myEntity.setContentType("application/xml");
httpPostRequest.setEntity(myEntity);
HttpResponse httpResponse = httpClient.execute(_target, httpPostRequest);
} catch (Exception e) {
    System.out.println(e.getStackTrace());
}
finally {
    // When HttpClient instance is no longer needed,
    // shut down the connection manager to ensure
    // immediate deallocation of all system resources
    httpClient.getConnectionManager().shutdown();
}