Creating a Java SOAP Service with Eclipse and Tomcat

Posted on **May 1, 2016** by **Bruno**

In this example, the **Eclipse Java EE IDE** is used to create a new Java SOAP service. The **Apache Tomcat** web server is used to deploy and run the Java SOAP service and SOAP-UI is used to test the service operations. The Java SOAP service that is created here represents a simple product catalog and provides methods to search and insert products.

Related Posts

- Setup of Java EE Development Environment with Eclipse Neon and Tomcat 9
- Example of Pega 7 SOAP Web Service Integration
- Creating a REST Web Service with Eclipse Neon, Tomcat 9, JAX-RS Jersey 2.24 and Jackson

Summary

Software Downloads

The open-source software used in this example can be downloaded from these sources:

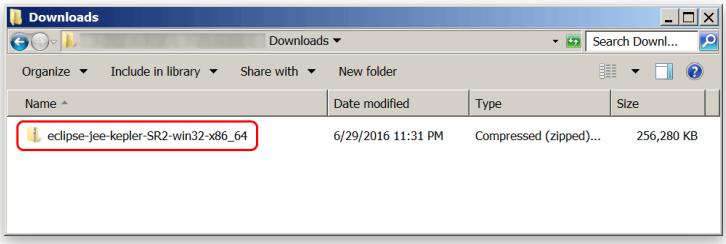
Java Runtime Environment	from Oracle *(e.g version 8, update 111)*
Apache Tomcat Web Server	*(here version 7)*
Eclipse Java EE IDE	from the Eclipse Downloads page *(here Kepler SR2)*
SOAP-UI	from the SOAP-UI open source downloads

1 Installation of Eclipse Java EE IDE

- **Download** the latest Java Runtime Environment for your operating system. I used JRE version 8, update 111 *(released: October 18, 2016)* for this post.

- For a related example, refer to Creating a REST Web Service with Eclipse Neon, Tomcat 9, JAX-RS Jersey 2.24 and Jackson on this blog.
- Download the Eclipse Java EE IDE for Web Developers from the Eclipse download page.
- In this example, ***Kepler Service Release 2*** *(build id: 20140224-0627)* of the Eclipse EE IDE was used.

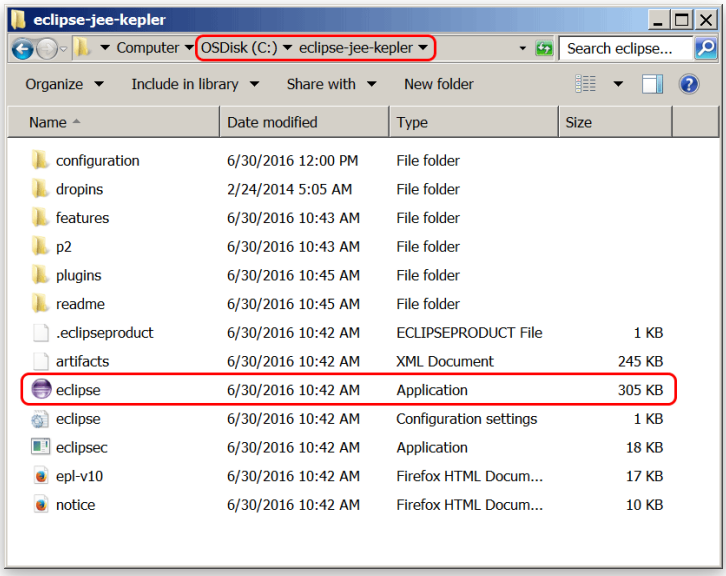
- The Eclipse IDE is packaged as a ZIP file. After downloading the file, unzip it to a folder on your hard drive.



- In this case, Eclipse is installed in: `C:\eclipse-jee-kepler`.
- After unzipping the file, the folder should contain the files shown below.

Note: Make sure that you have a Java Runtime Environment *(JRE)* installed before trying to run Eclipse.

- A JRE can be downloaded from Oracle at https://java.com/en/download/manual.jsp.

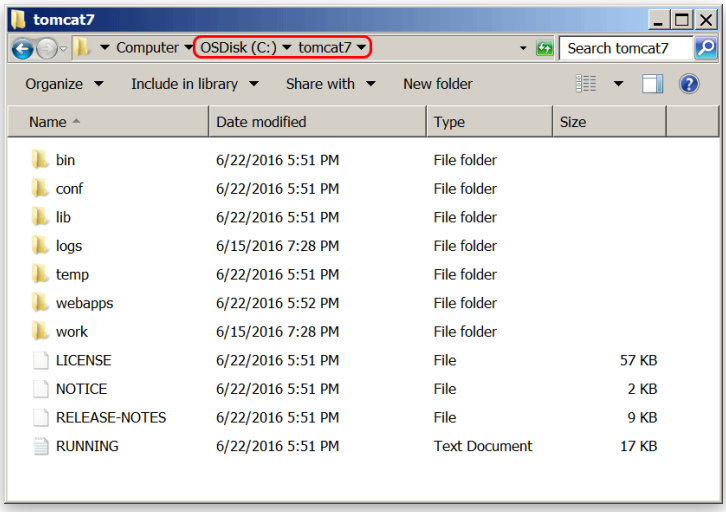


- ***Run Eclipse*** by clicking on the application icon as shown in the screen shot.

2 Installation of Apache Tomcat 7 Web Server

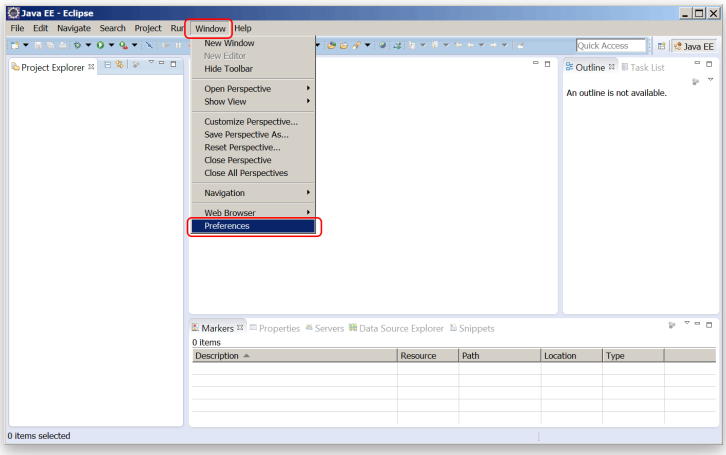
RECEN
- Create a WebSocket C 23, 2018
- How to Read Pega 7 E February 18, 2018
- Create Spring Web MV Archetype in Eclipse I
- Example of Creating a Java January 28, 2018
- Eclipse Neon – Create Spring – Without Mave
- Creating a Simple Jav Socket January 23, 20
- Eclipse Neon – Export WAR File January 18, 2
- Create Java Project in User Library January 1
- Export a Java Project Executable JAR with I
- Creating a Simple Jav Client Socket Decemb
- Configure Pega 7 Acti Page November 21, 20
- Configuring Pega 7 Ac Page November 14, 20
- Setting up Maven on I Project October 19, 20
- Create Pega 7 REST S Rule October 9, 2017
- Configure Pega 7 HTT Calls September 21, 20

- Visit https://tomcat.apache.org/index.html and ***download the Apache Tomcat Web Server***. For this example, version 7 was used.
- It is recommended to download the binary distribution for your operating system (e.g. Windows 7 32-bit), which provides a ZIP file.
- Unzip the downloaded file to a folder on your hard drive, for example to `C:\tomcat7`. The Tomcat 7 root folder should look like this:

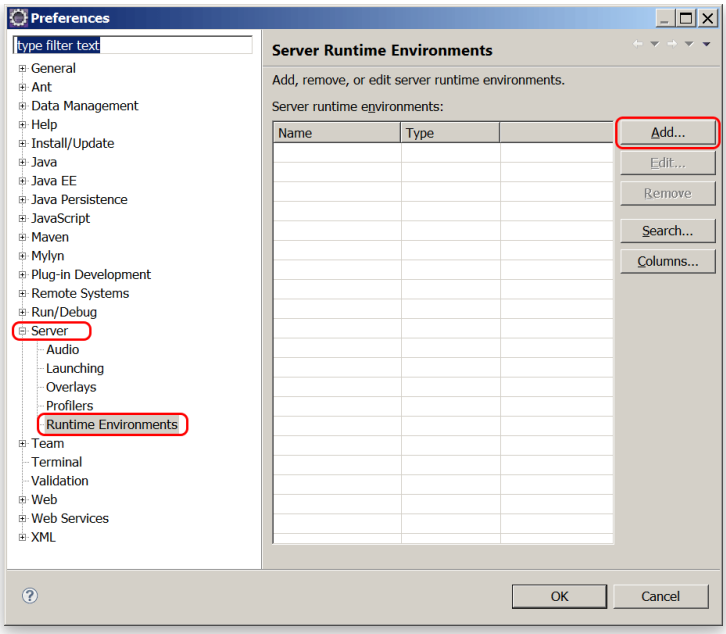


3 Configuration of Tomcat 7 Runtime Environment in Eclipse

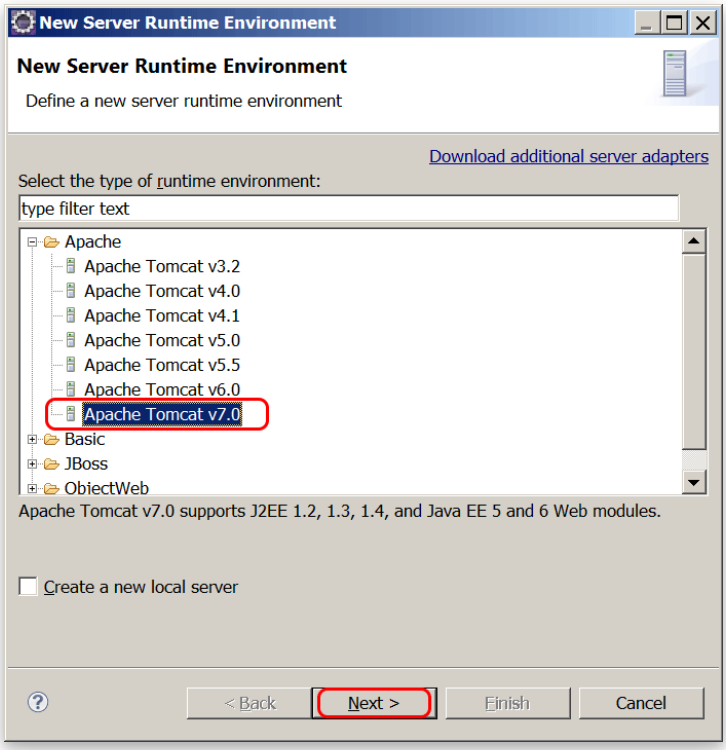
- Start the Eclipse IDE by clicking on the Eclipse application icon shown earlier.
- In Eclipse, navigate to: ***Window*** > ***Preferences***.



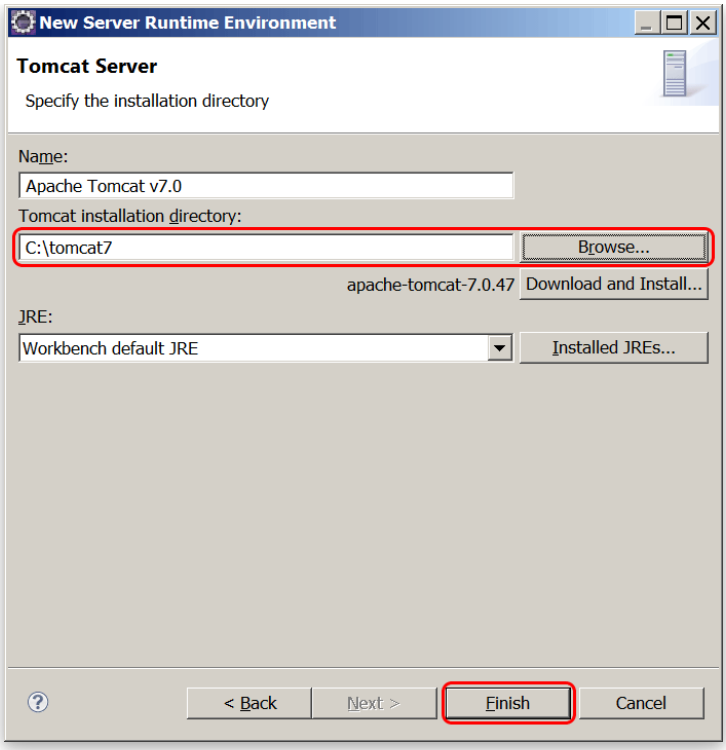
- In the ***Preferences*** dialog window, extend the ***Server*** node and click on ***Runtime Environments***.
- Then click on the ***Add...*** button to add a new runtime environment.



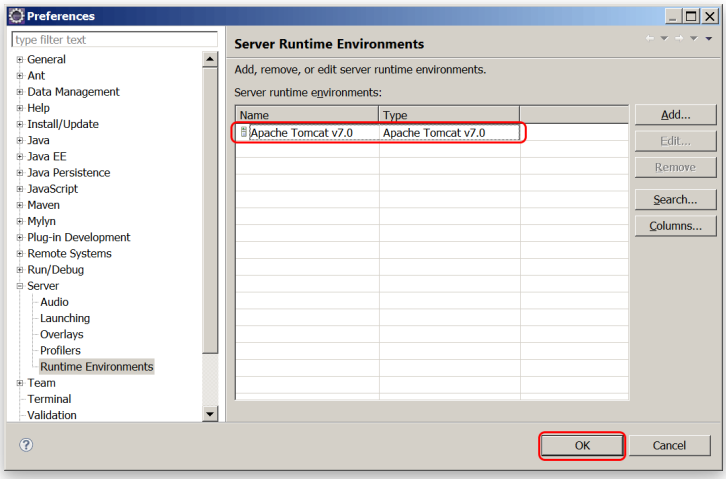
- In the ***New Server Runtime Environment*** dialog, select ***Apache Tomcat v7.0*** and click on ***Next***.



- On the following screen, use the ***Browse...*** button to select the root folder of the Tomcat web server.
- This is the folder into which the Tomcat ZIP file was unzipped. It contains the `bin` folder.



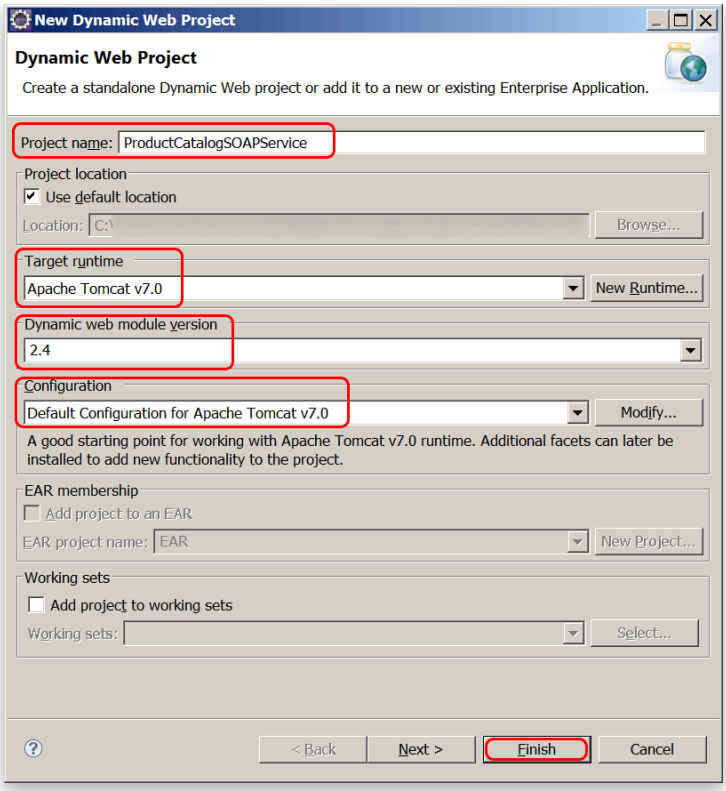
- Click on ***Finish*** to complete the runtime environment setup. The Apache Tomcat v7.0 web server should now be listed in the ***Server Runtime Environments*** screen.



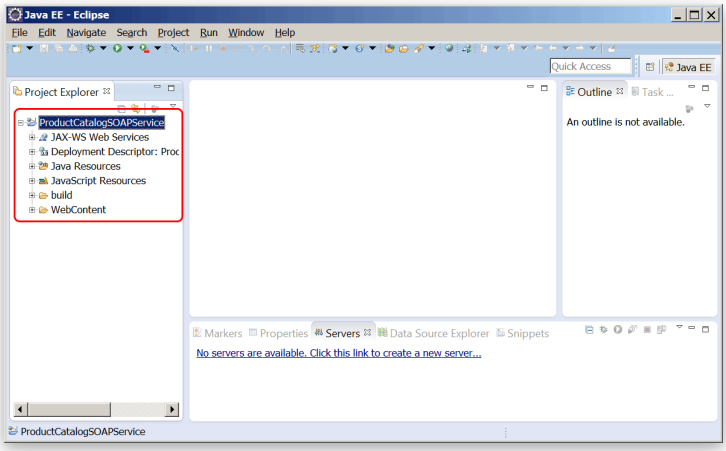
- Click on ***OK*** to close the runtime environment setup.

4 Setup of New Dynamic Web Project in Eclipse

- In the Eclipse IDE, navigate to the main menu and click on ***File*** > ***New*** > ***Dynamic Web Project***.
- In the configuration screen, enter the project name and select the target runtime.
- In this case, the target runtime will be the previously configured ***Apache Tomcat v7.0*** web server.
- Click on ***Finish*** to complete the setup.

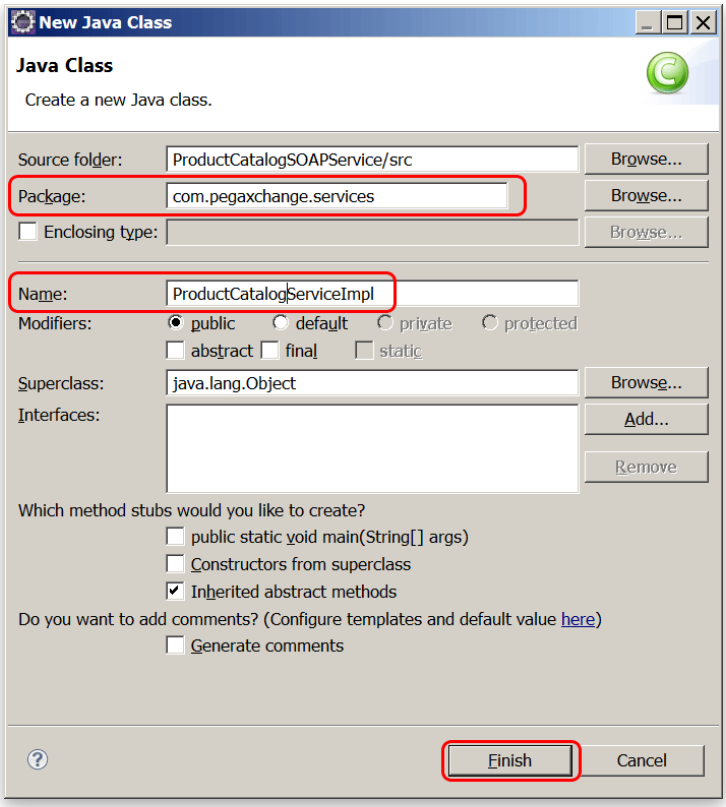


- The new dynamic web project should now show up in the ***Project Explorer*** in the Eclipse IDE as shown below:



5 Setup of Service Implementation Java Class

- Right click on the project name and select ***New*** > ***Class*** in the context menu to create a new Java class that will serve as the implementation class for the web service.



- Select an appropriate package name and complete the name of the class. Then click on ***Finish*** and open the class in the Java editor.
- For this example, the `ProductCatalogServiceImpl` Java class provides 3 methods and maintains a simple product catalog as an in-memory list:

```
1.  package com.pegaxchange.services;
2.
3.  import java.util.*;
4.
5.  public class ProductCatalogServiceImpl {
6.
7.      private static List<Product> productCatalog;
8.
9.      public ProductCatalogServiceImpl() {
10.         initializeProductCatalog();
11.     }
12.
13.     public Product searchById(int id) throws Exception {
14.         for (Product p : productCatalog) if (p.getId() == id) return p;
15.         throw new Exception("No product found with id " + id);
16.     }
17.
18.     public Product[] getAllProducts() {
19.         Product[] products = new Product[productCatalog.size()];
20.         int i = 0;
21.
22.         for (Product p : productCatalog) {
23.             products[i] = p;
24.             i++;
25.         }
26.
27.         return products;
28.     }
29.
30.     public void insertProduct(Product product) {
31.         productCatalog.add(product);
32.     }
33.
34.     private void initializeProductCatalog() {
35.
36.         if (productCatalog == null) {
37.           productCatalog = new ArrayList();
38.           productCatalog.add(new Product(1, "Keyboard", "Electronics", 29.99D));
39.           productCatalog.add(new Product(2, "Mouse", "Electronics", 9.95D));
40.           productCatalog.add(new Product(3, "17\" Monitor", "Electronics", 159.49D));
41.           productCatalog.add(new Product(4, "Hammer", "Hardware", 9.95D));
42.           productCatalog.add(new Product(5, "Slot Screwdriver", "Hardware", 7.95D));
```

```
            productCatalog.add(new Product(6, "The British Invasion of Java", "Books", 11.39D));
            productCatalog.add(new Product(7, "A House in Bali", "Books", 15.99D));
            productCatalog.add(new Product(8, "An Alaskan Odyssey", "Books", 799.99D));
            productCatalog.add(new Product(9, "LCD Projector", "Electronics", 1199.19D));
        }
    }
}
```

The 3 public methods in the class are:

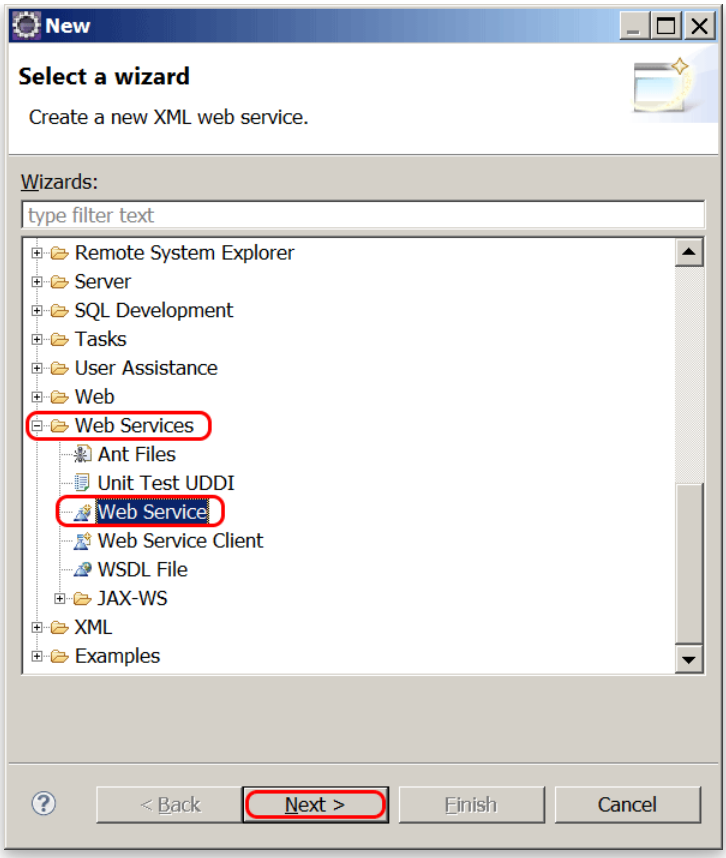
- `searchById` – Returns the product with the given id
- `getAllProducts` – Returns a list of all products
- `insertProduct` – Adds a new product to the product catalog list object

Note: The `initializeProductCatalog` method is called in the constructor and creates an in-memory ArrayList of Product objects. In reality, the product catalog would be backed by a database or some other system of record. In addition, a Java bean class that represents a product in the catalog was created:

```
package com.pegaxchange.services;

public class Product {

    private int id;
    private String name;
    private String category;
    private double unitPrice;

    public Product() {}

    public Product(int id, String name, String category, double unitPrice) {
        this.id = id;
        this.name = name;
        this.category = category;
        this.unitPrice = unitPrice;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public double getUnitPrice() {
        return unitPrice;
    }

    public void setUnitPrice(double unitPrice) {
        this.unitPrice = unitPrice;
    }
}
```

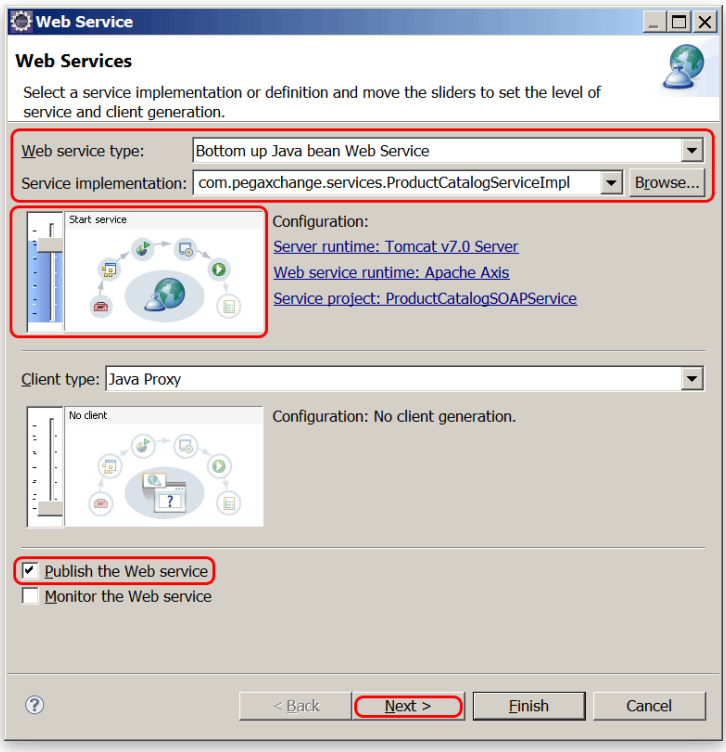
6 Setup of New SOAP 1.1 Web Service in Eclipse

- In the Eclipse main menu, click on ***File*** > ***New*** > ***Other***.
- In the Wizard dialog, expand the ***Web Services*** node, select ***Web Service*** and click on ***Next***.

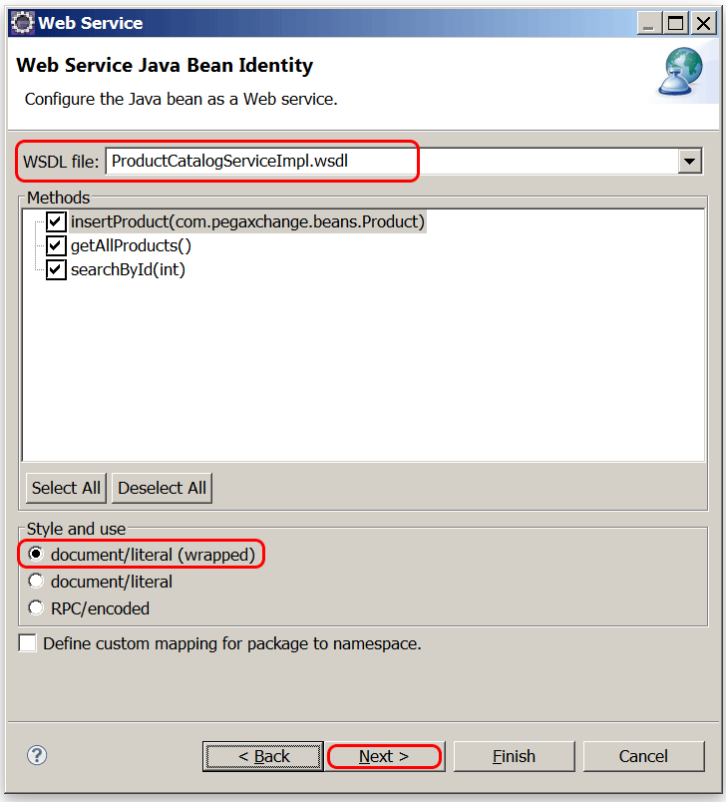


- On the ***Web Services*** wizard screen, select ***Bottom up Java bean Web Service*** for the web service type and use the ***Browse*** button to select the service implementation class.
- In this case it is the `com.pegaxchange.services.ProductCatalogServiceImpl` Java class created in step 5.
- In the configuration section, leave the slider at the ***Start*** level. This will configure Eclipse to automatically deploy and start the web service on the Apache Tomcat v7 runtime.
- Omit the client configuration for now and make sure the option ***Publish the Web service*** is checked. Click on ***Next*** to continue.

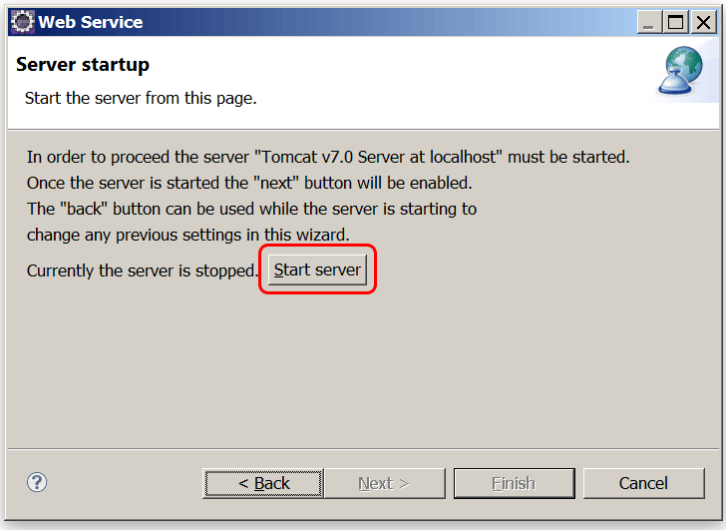
Note: Under ***Configuration:***, the entry `Web service runtime: Apache Axis` refers to Axis version 1.4 which generates SOAP version 1.1 web services.



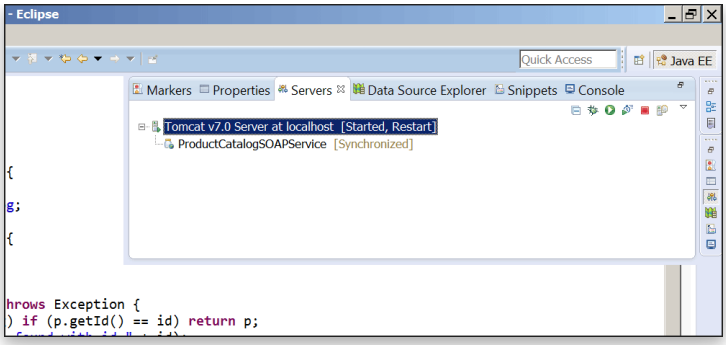
- The next screen lists the name of the WSDL file that will be created and the public methods available in the service implementation class that can be exposed through the Java SOAP service.
- The ***document type*** can be set on this screen as well.



- Click on ***Next*** to continue. The Eclipse IDE will now generate the web service files. If the Apache Tomcat web server is NOT running at this point, the following ***Server startup*** screen will be shown:

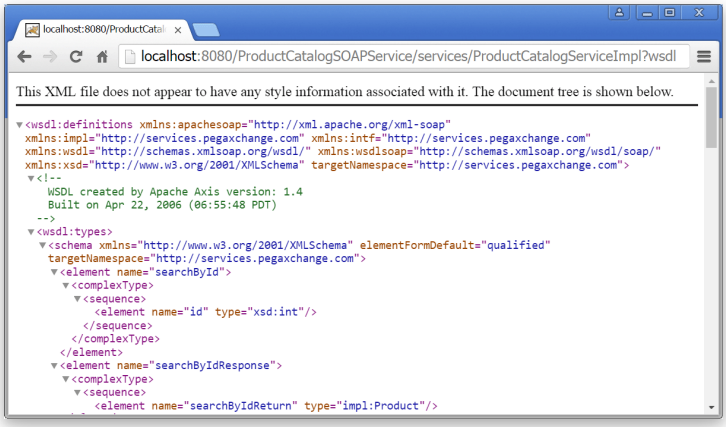


- Click on the ***Start server*** button to start the web server and then click on the ***Next*** button and after the service has been deployed, click on the ***Finish*** button to close the Web Service wizard.
- The service should now be up and running on the local Tomcat web server as shown below.



- Confirm that the WSDL file is accessible by navigating to the following URL in a web browser:

```
https://localhost:8080/ProductCatalogSOAPService/services/ProductCatalogServiceImpl?wsdl
```



- Note that the namespace of the generated WSDL is `https://schemas.xmlsoap.org/wsdl/soap`, which indicates a SOAP 1.1 compliant web service.
- For more information on SOAP 1.1 vs. 1.2 see this post on W3C.
- For completeness, here is the entire WSDL file for the `ProductCatalogSOAPService` that was generated by Eclipse:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="https://services.pegaxchange.com"
    xmlns:apachesoap="https://xml.apache.org/xml-soap"
    xmlns:impl="https://services.pegaxchange.com"
    xmlns:intf="https://services.pegaxchange.com"
    xmlns:wsdl="https://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="https://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="https://www.w3.org/2001/XMLSchema">
  <!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48 PDT)-->
  <wsdl:types>
    <schema elementFormDefault="qualified"
        targetNamespace="https://services.pegaxchange.com"
        xmlns="https://www.w3.org/2001/XMLSchema">
      <element name="searchById">
        <complexType>
          <sequence>
            <element name="id" type="xsd:int"/>
          </sequence>
        </complexType>
      </element>
      <element name="searchByIdResponse">
        <complexType>
          <sequence>
            <element name="searchByIdReturn" type="impl:Product"/>
          </sequence>
        </complexType>
      </element>
      <complexType name="Product">
        <sequence>
          <element name="category" nillable="true" type="xsd:string"/>
          <element name="id" type="xsd:int"/>
          <element name="name" nillable="true" type="xsd:string"/>
          <element name="unitPrice" type="xsd:double"/>
        </sequence>
      </complexType>
```

```
      <element name="insertProduct">
        <complexType>
          <sequence>
            <element name="product" type="impl:Product"/>
          </sequence>
        </complexType>
      </element>
      <element name="insertProductResponse">
        <complexType/>
      </element>
      <element name="getAllProducts">
        <complexType/>
      </element>
      <element name="getAllProductsResponse">
        <complexType>
          <sequence>
            <element maxOccurs="unbounded" name="getAllProductsReturn" type="impl:Product"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
  <wsdl:message name="insertProductResponse">
    <wsdl:part element="impl:insertProductResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="getAllProductsRequest">
    <wsdl:part element="impl:getAllProducts" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="searchByIdRequest">
    <wsdl:part element="impl:searchById" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="getAllProductsResponse">
    <wsdl:part element="impl:getAllProductsResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="searchByIdResponse">
    <wsdl:part element="impl:searchByIdResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="insertProductRequest">
    <wsdl:part element="impl:insertProduct" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="ProductCatalogServiceImpl">
    <wsdl:operation name="searchById">
      <wsdl:input message="impl:searchByIdRequest" name="searchByIdRequest"/>
      <wsdl:output message="impl:searchByIdResponse" name="searchByIdResponse">
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="insertProduct">
      <wsdl:input message="impl:insertProductRequest" name="insertProductRequest"/>
      <wsdl:output message="impl:insertProductResponse" name="insertProductResponse"/>
    </wsdl:operation>
    <wsdl:operation name="getAllProducts">
      <wsdl:input message="impl:getAllProductsRequest" name="getAllProductsRequest"/>
      <wsdl:output message="impl:getAllProductsResponse" name="getAllProductsResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ProductCatalogServiceImplSoapBinding" type="impl:ProductCatalogServiceImpl">
    <wsdlsoap:binding style="document" transport="https://schemas.xmlsoap.org/soap/https"/>
    <wsdl:operation name="searchById">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="searchByIdRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="searchByIdResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="insertProduct">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="insertProductRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="insertProductResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getAllProducts">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getAllProductsRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="getAllProductsResponse">
        <wsdlsoap:body use="literal"/>
```

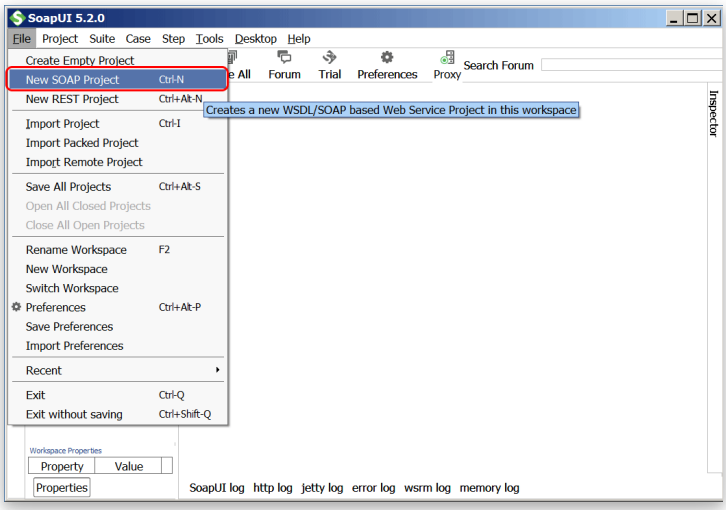
```
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ProductCatalogServiceImplService">
    <wsdl:port binding="impl:ProductCatalogServiceImplSoapBinding"
        name="ProductCatalogServiceImpl">
      <wsdlsoap:address
        location="https://prpc:8080/ProductCatalogSOAP11/services/ProductCatalogServiceImpl"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

7 Testing the new Java SOAP Service with SOAP UI

There are many options for testing a SOAP service, including generating a test client in Eclipse. In this example, a free external tool called ***SOAP-UI*** is used to test the web service operations.

Note: Make sure that the Tomcat web server is running at this point and that the web service is deployed as described in step 6, where the service's WSDL file is accessed in a browser.

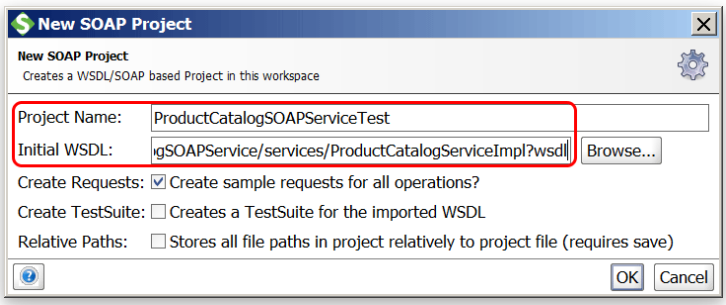
- SOAP UI can be downloaded at this URL: https://www.soapui.org/downloads/latest-release.html.
- Follow the installation instructions and once completed, navigate to ***File*** > ***New SOAP Project***.



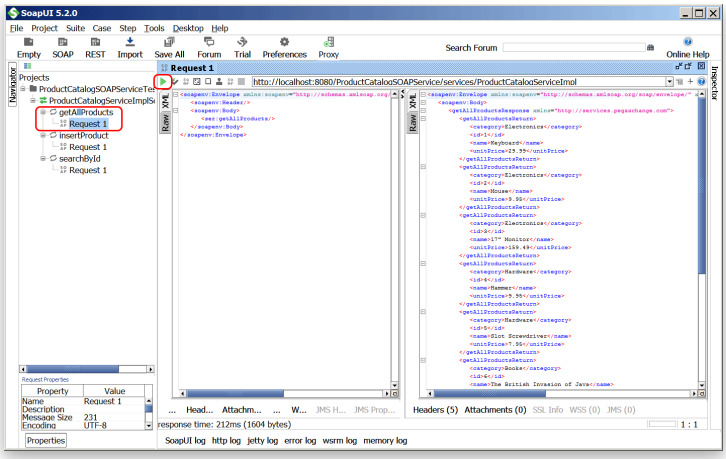
- A new dialog window prompts for the project name and the WSDL file.
- In this case, the URL shown earlier is copied into the ***Initial WSDL*** field:

```
https://localhost:8080/ProductCatalogSOAPService/services/ProductCatalogServiceImpl?wsdl
```

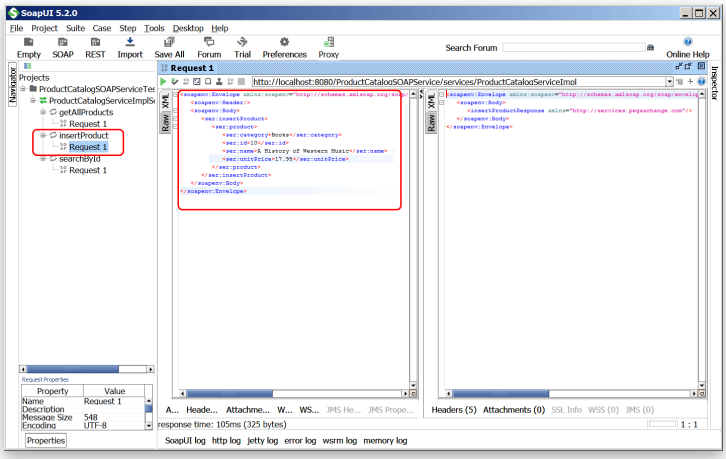
- Check the ckeckbox ***Create sample requests for all operations?*** is checked and click on ***OK***.



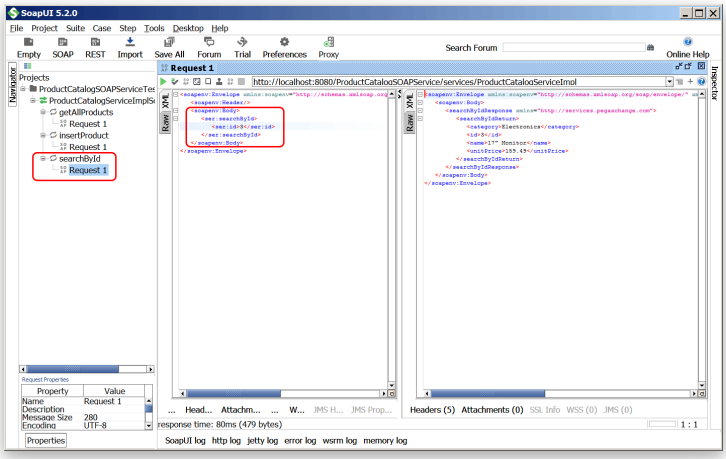
- SOAP-UI will automatically download the WSDL, process it and create request stubs for each service operation.
- In order to call one of the operations of the web service, double click on the ***Request 1*** item under that operation.
- Then click on the green triangle inside the request window to run the request.
- E.g.: calling the operation `getAllProducts` results in the following request and response XML.



- Similarly, calling the operation `insertProduct` results in the following request and response XML.



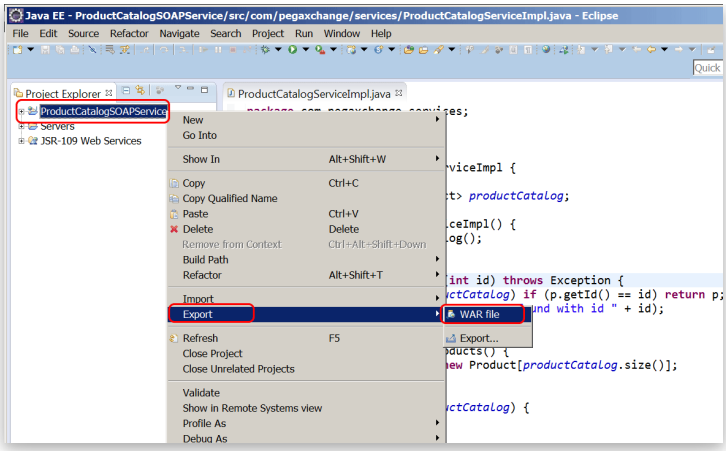
- Calling the operation `searchByID` results in the following request and response XML.



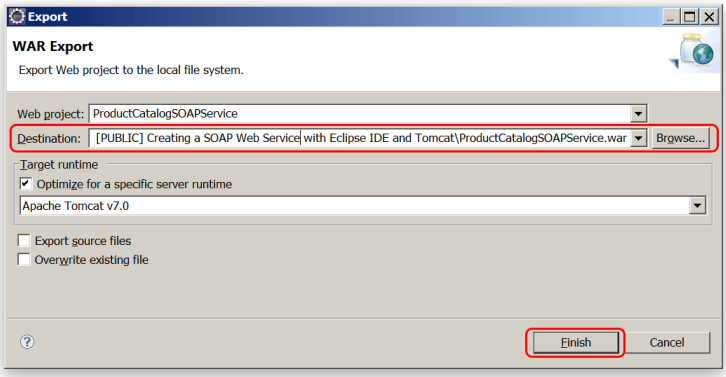
- The Java SOAP service is functioning as designed and can now be called from a Pega application using a CONNECT-SOAP rule. For an example see Example of Pega 7 SOAP Web Service Integration.

8 Creating a WAR File for Deployment of the SOAP Service

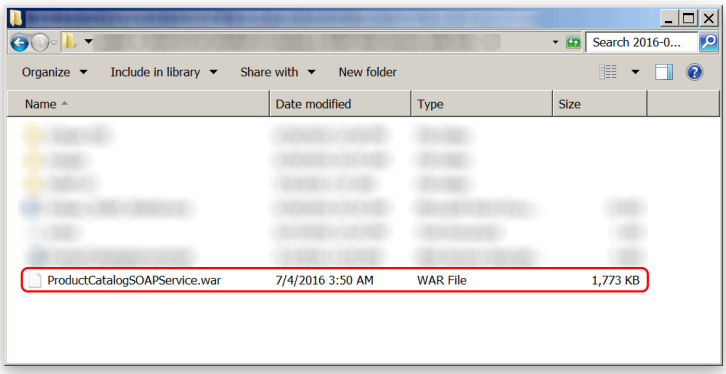
- A WAR file *(Web Archive)* can be created by exporting the dynamic web project in Eclipse.
- The WAR file is used to deploy the Java SOAP service to a Jave EE compliant application server.
- In the Eclipse IDE, in the ***Project Explorer***, right click on the node for the dynamic web project
- Select ***Export*** > ***WAR file*** to launch the WAR Export dialog.



- In the ***WAR Export*** dialog, select the ***Destination*** using the ***Browse*** button and then click on ***Finish*** to complete the WAR export.



- The generated WAR file will be copied into the specified ***destination directory:***



- The WAR file can now be used to deploy the Java SOAP service on any Java EE compliant application server or servlet container *(…such as Tomcat)*.
- The WAR file can be deployed on the Tomcat web container of a Pega 7 exercise system running on VMWare.
- For more information on how to access the Pega 7 Tomcat web server admin console, see: Access Pega 7 Tomcat Admin Console for PVS Installations
- For an example on how to create a REST web service with Eclipse Neon and Tomcat 9, please refer to: Creating a REST Web Service with Eclipse Neon, Tomcat 9, JAX-RS Jersey 2.24 and Jackson.

---

21 THOUGHTS ON “CREATING A JAVA SOAP SERVICE WITH ECLIPSE AND TOMCAT”

rohancreddy on **March 13, 2017 at 10:41 pm** said:

Hi Bruno,
Thanks for the post but getting a “Type mismatch: cannot convert from element type

Object to Product” at the line :

for (Product p : productCatalog) if (p.getId() == id) return p;

in the ProductCatalogSOAPImpl.java.
Any help is appreciated.

Thanks

Log in to Reply

Bruno
on **March 20, 2017 at 11:13 pm** said:

Thanks for letting me know. I will take a look. Did you try casting? … productCatalog should contain objects of Product, if not you may need to cast to Product type in the loop.

Log in to Reply

madhu
on **November 15, 2017 at 8:27 am** said:

hi are you managed to fix you error?
if you done please let me know
thanks.

Log in to Reply

Bruno
on **November 30, 2017 at 1:27 am** said:

Thanks to syirasky,

I corrected `ProductCatalogServiceImpl` at line 7 to use:

```
private static List<Product> productCatalog;
```

Log in to Reply

syirasky
on **November 27, 2017 at 5:38 am** said:

at line 7, change to this:

```
private static List<Product> productCatalog;
```

Log in to Reply

syirasky
on **November 29, 2017 at 10:53 am** said:

Sorry for misinformation.
At line 4, I change from
private static List productCatalog;

to this
private static List<Product> productCatalog;

hope it helps.

Log in to Reply

Bruno

on **November 30, 2017 at 1:45 am** said:

Hi syirasky, thank you VERY much for finding this error and letting me know. I think the < and > characters where filtered out when I pasted the source code.

It is necessary to use HTML encoding for these and other special characters:
for < the encoding **<** should be used
for > the encoding **>** should be used

It is generally a good idea to use some online HTML encoder to make sure all characters in a post are displayed properly. Here is one I use: https://www.web2generators.com/html-based-tools/online-html-entities-encoder-and-decoder

Log in to Reply

Bruno
on **November 30, 2017 at 1:34 am** said:

Please see the updated code for the implementation of `ProductCatalogSOAPImpl`. Line 7 was changed to specify the object class of the list items:

```
private static List<Product> productCatalog;
```

Log in to Reply

Pirate on **April 2, 2017 at 5:39 pm** said:
Hii .. Thank you very much for creating such a valuable blog…Please keep on adding new articles … If possible can you create new post on – Creating REST service in Pega 7 and accessing it from external application-Java. Thanks.

Log in to Reply

Bruno
on **April 4, 2017 at 12:46 am** said:

Hello,

thanks for the message. Yes, I was planning to write posts on how to create REST and SOAP services in Pega. Hopefully soon 😉

Log in to Reply

Bruno
on **November 27, 2017 at 7:05 pm** said:

Hello,

please see: https://www.pegaxchange.com/2017/10/09/create-a-rest-service-in-pega-7-1-9-with-service-rest-rule/

Log in to Reply

mmelo on **April 19, 2017 at 2:46 pm** said:
Hi!

Everything from this tutorial worked just fine for me. But when I try to make requests from a Python Script instead of SOAP UI, I get an error message. How can I fix it?

The python script:
import requests

url="http://localhost:8088/ProductCatalogSOAPService/services/ProductCatalogServiceImpl"
#headers = {'content-type':'application/soap+xml'}
headers = {'content-type': 'text/xml'}
body = """

"""

response = requests.post(url,data=body,headers=headers)

print response
print response.content

The response:

ns1:Client.NoSOAPAction
no SOAPAction header!

Marcuss-MacBook-Pro.local

Log in to Reply

mmelo
on **April 19, 2017 at 2:49 pm** said:

I don'y know why some parts of my python script were suppressed when I posted (inserted some spaces to see if it goes). I will try again:

Log in to Reply

Bruno
on **November 30, 2017 at 1:37 am** said:

You have to use HTML encoding. See this site for an online encoder:
https://www.web2generators.com/html-based-tools/online-html-entities-encoder-and-decoder
you can paste Python code or SOAP XML and then post the encoded text in a blog comment.

Log in to Reply

Bruno
on **April 19, 2017 at 3:42 pm** said:

Hi, are you able to post the entire SOAP message? If you compare the SOAP message that your script is trying to send to the one you use in SOAP-UI, you can check the differences.

Log in to Reply

mmelo
on **April 19, 2017 at 5:18 pm** said:

Im having trouble to post it. But Im using in the python script the same message used by soap ui to make the request.
Lets see if it goes now:
>? xml version="1.0″ encoding="UTF-8″?soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ser="http://services.pegaxchange.com"soapenv:Header/soapenv:Bodyser:getAllProducts//soapenv:Body/soapenv:Envelope
and < signs to see if it posts the comment right

Log in to Reply

> mmelo
> on **April 19, 2017 at 5:19 pm** said:
>
> (?xml version=”1.0″ encoding=”UTF-8″?)
> (soapenv:Envelope
> xmlns:soapenv=”http://schemas.xmlsoap.org/soap/envelope/”
> xmlns:ser=”http://services.pegaxchange.com”)
> (soapenv:Header/)
> (soapenv:Body)
> (ser:getAllProducts/)
> (/soapenv:Body)
> (/soapenv:Envelope)
>
> Log in to Reply
>
> > Bruno
> > on **November 30, 2017 at 1:54 am** said:
> >
> > Any luck resolving the issue? If not, let me know… send me the Python code and I can try myself. And as per my other reply, use HTML encoding to post: https://www.web2generators.com/html-based-tools/online-html-entities-encoder-and-decoder

lucenajj on **May 9, 2018 at 2:54 pm** said:
I’m trying to pass a double value however this is returning me an exception can not serialize 10.0, could anyone tell me how to handle this?

Log in to Reply

Bruno on **May 21, 2018 at 6:22 pm** said:
Hi, any code snippet you can share? Are you trying to use the primitive double type or the Double wrapper class?

Log in to Reply

sujatha0711 on **August 16, 2018 at 7:29 am** said:
Very good example with clear and detailed steps. I am able to create a wsdl and test with SOAPUI following the details provided here. Thank you so much !

Log in to Reply

Leave a Reply

You must be logged in to post a comment.