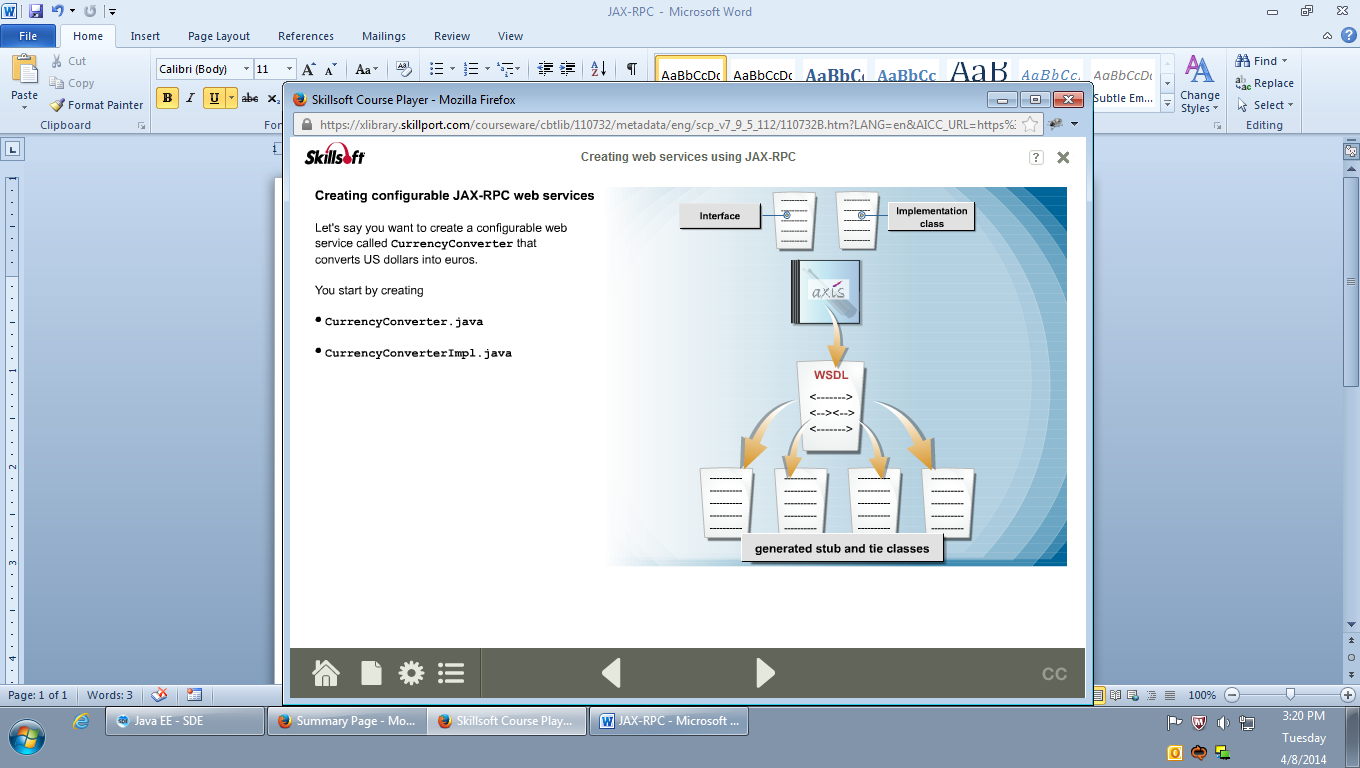
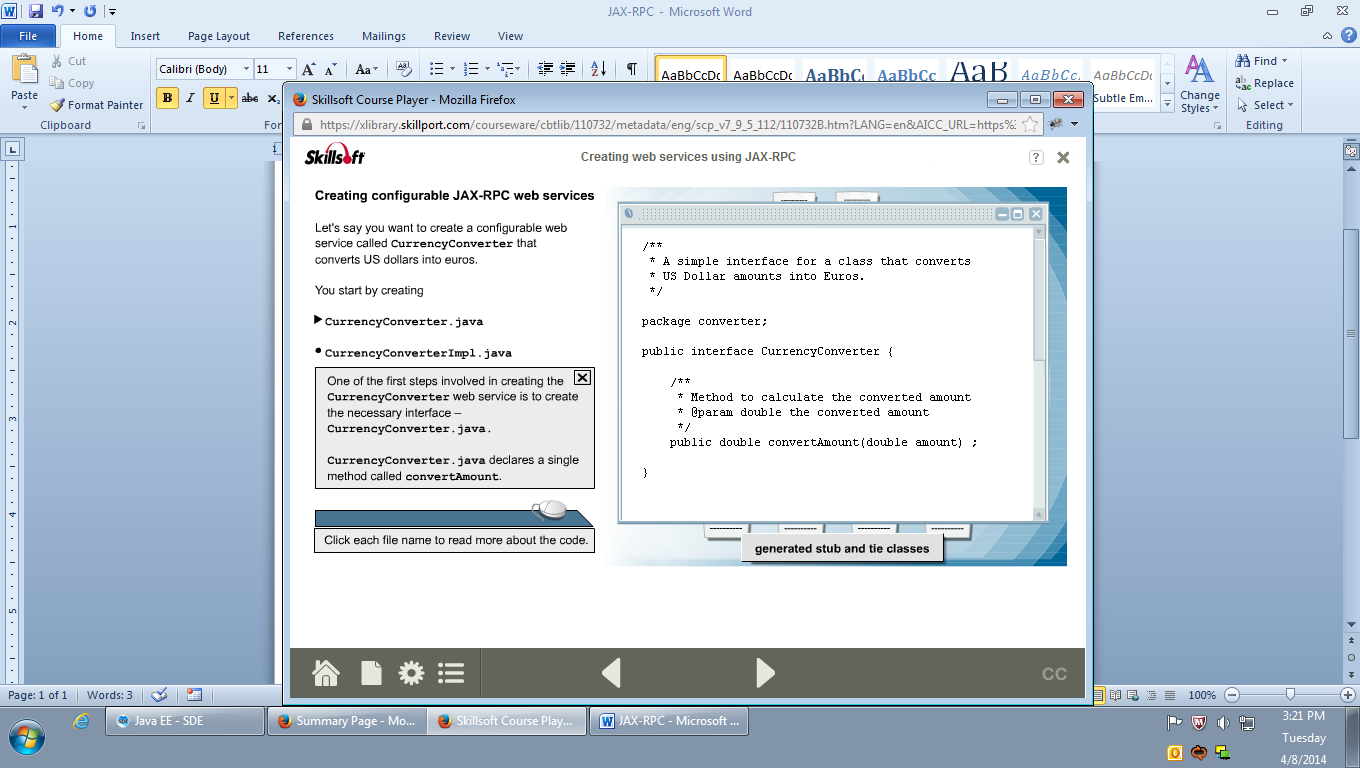
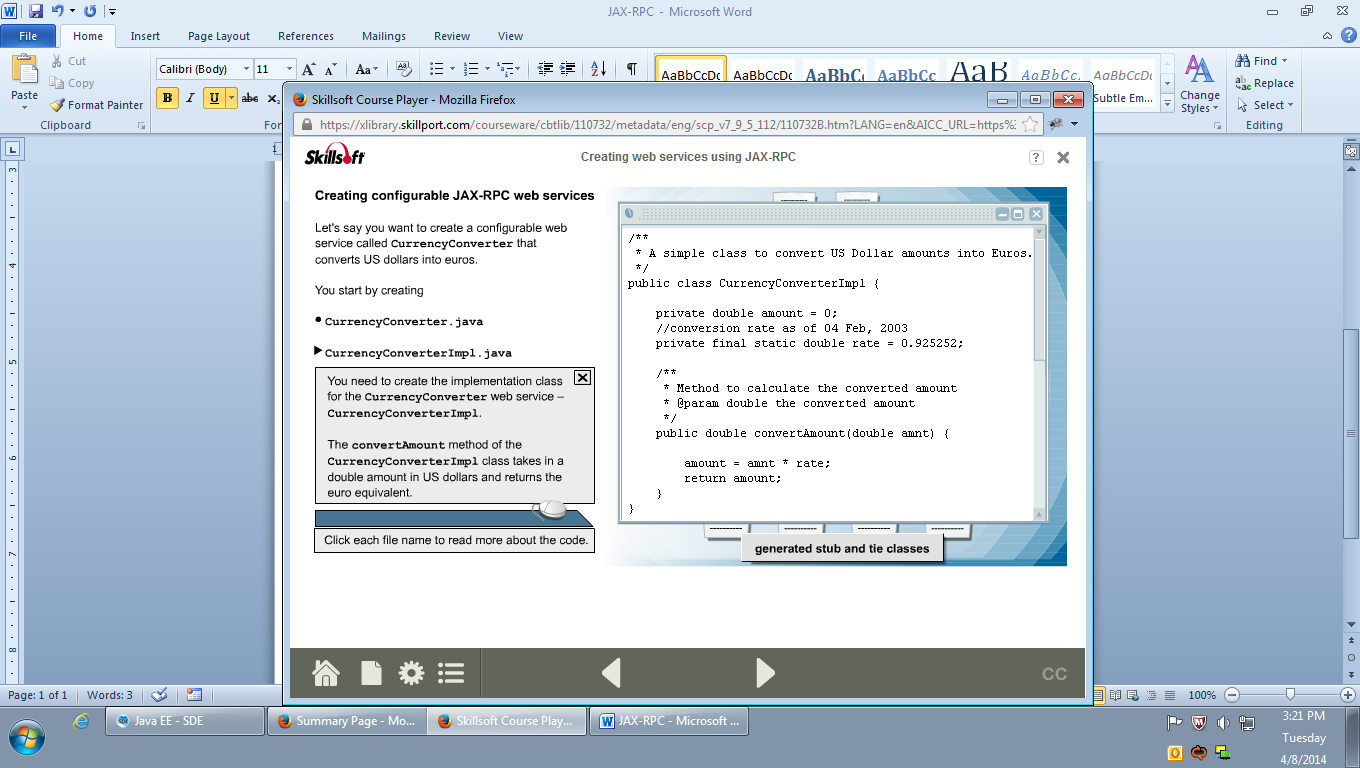
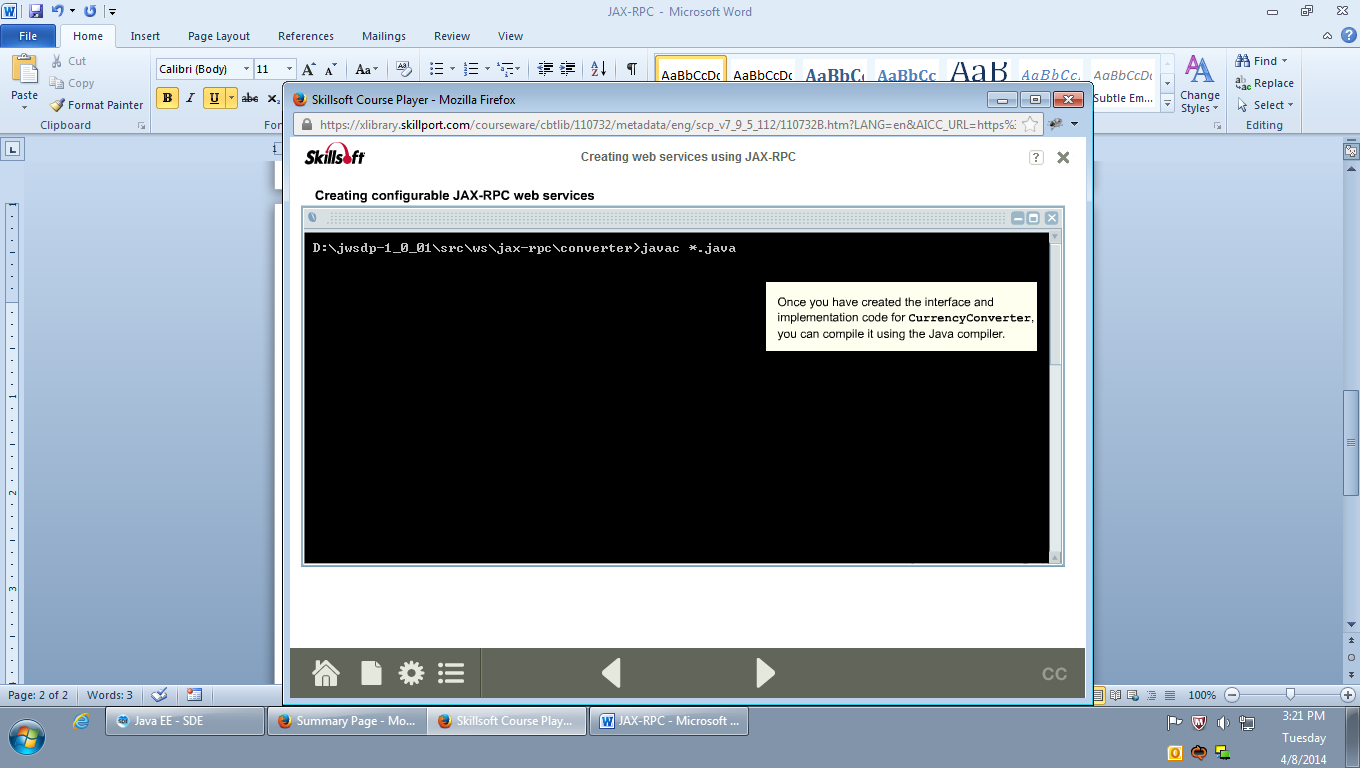
**JAX-RPC Web Services:**

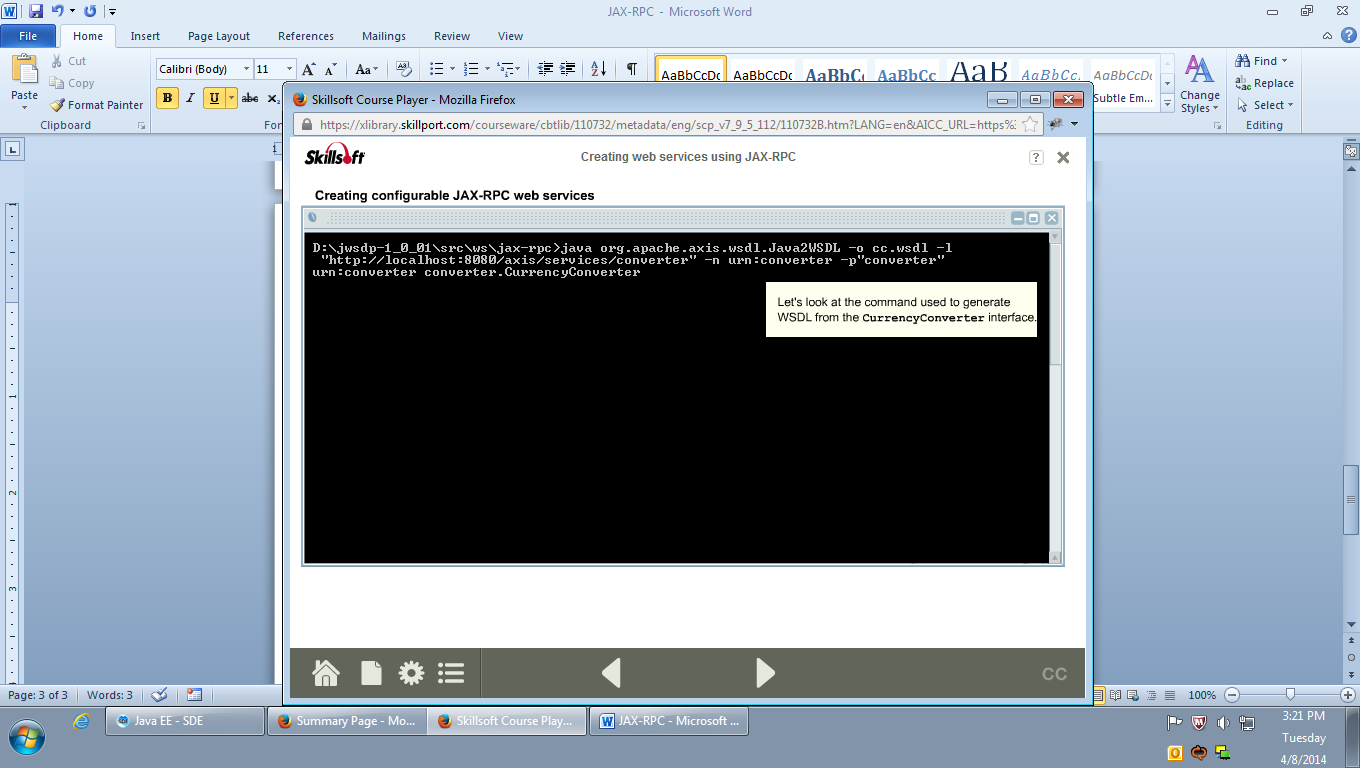


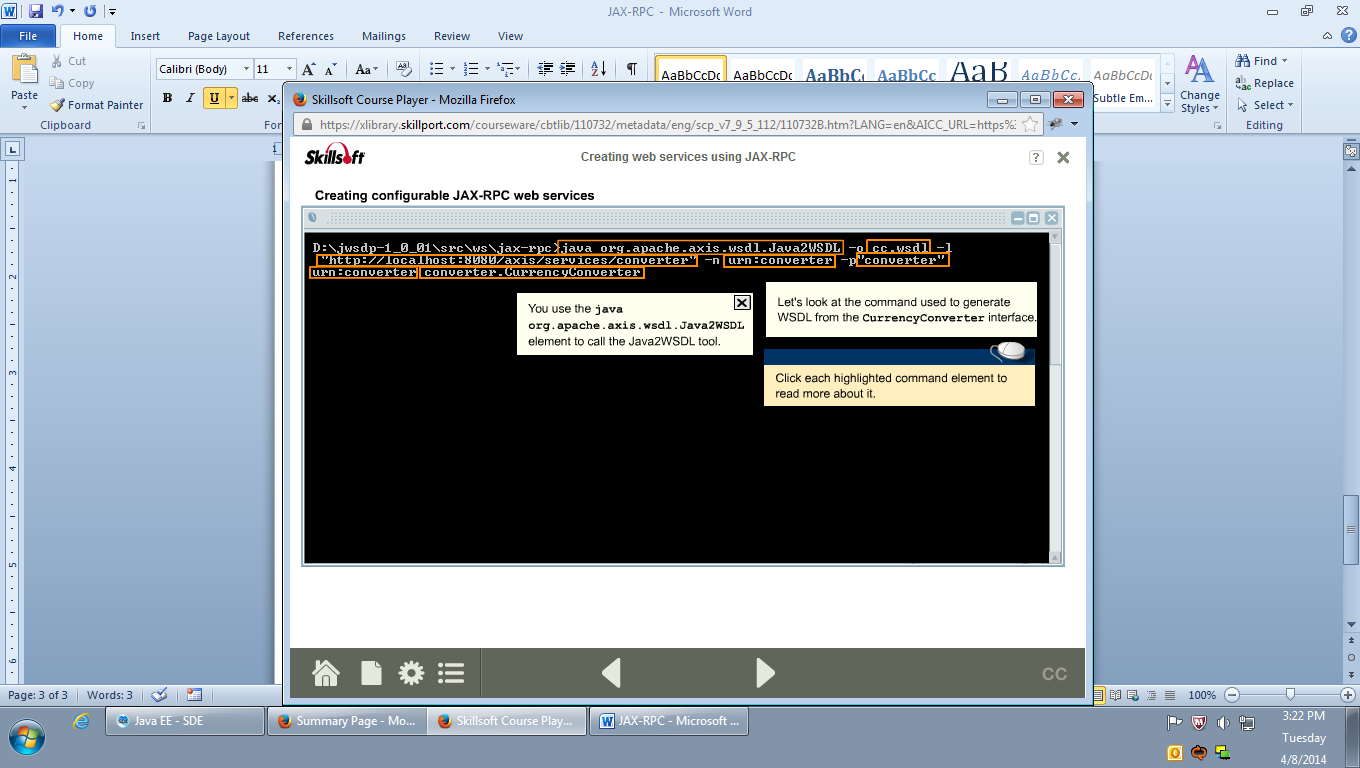


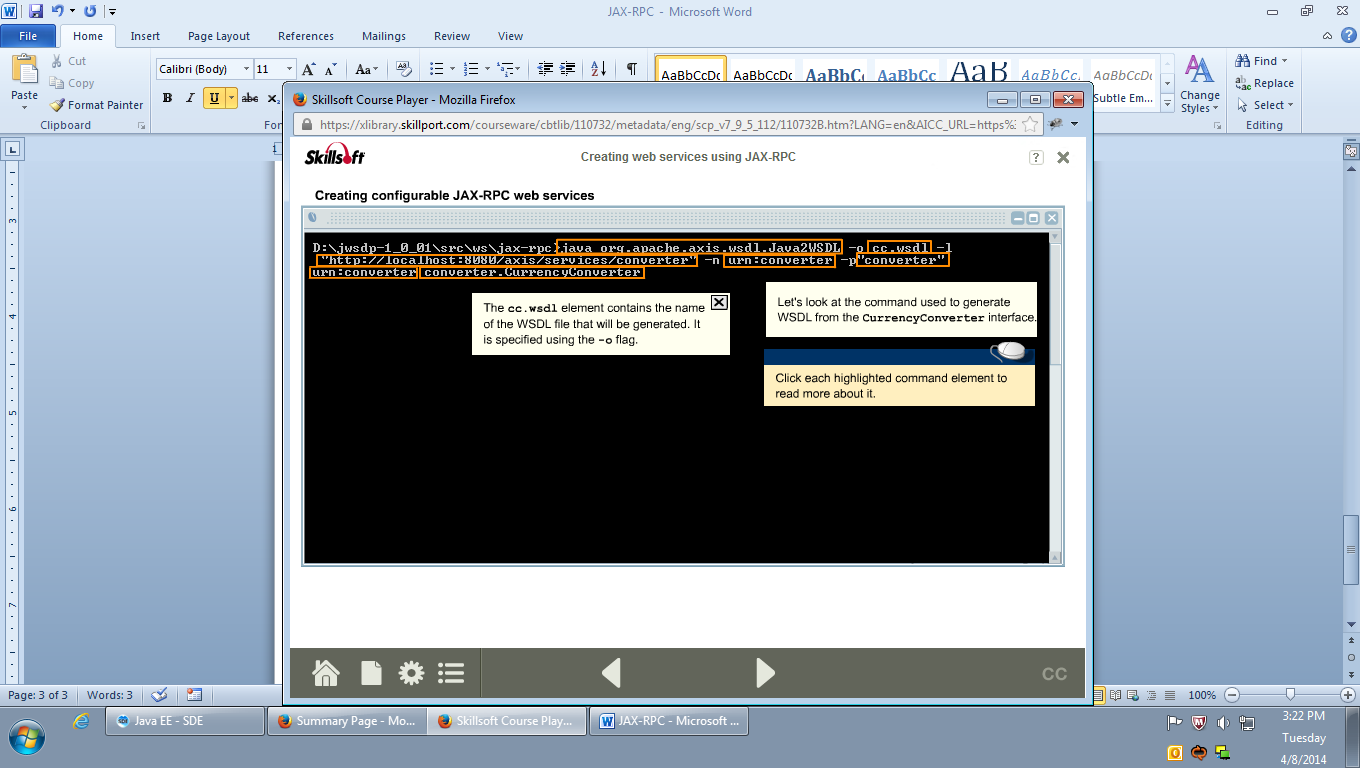


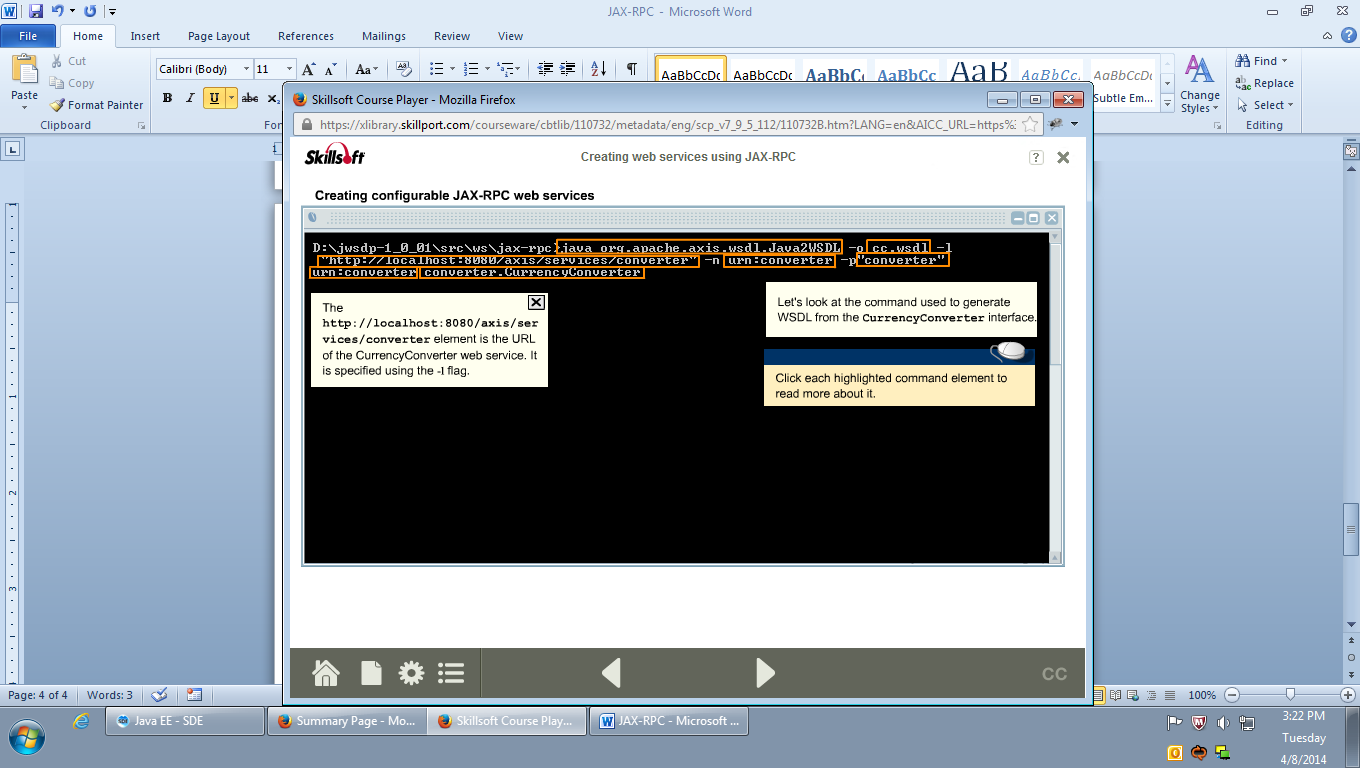
**Java-To-WSDL>>**

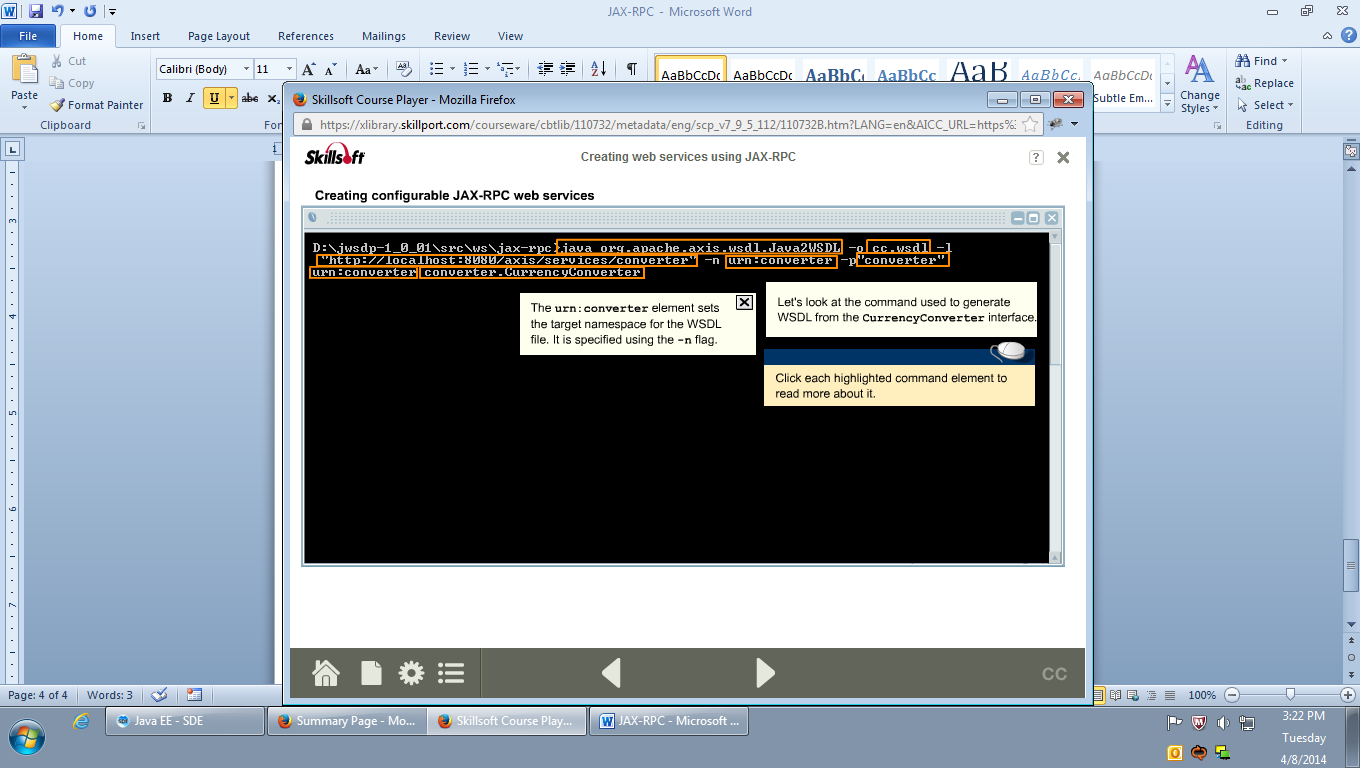


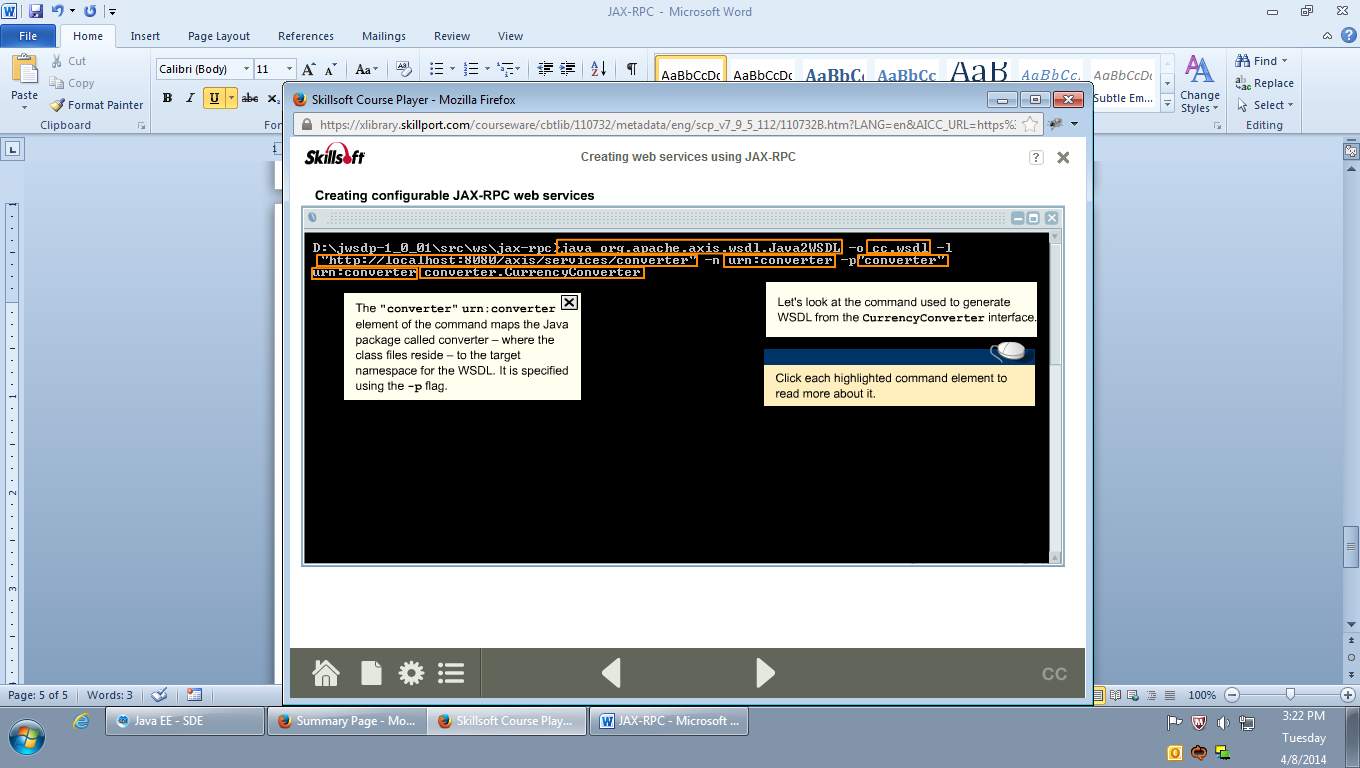


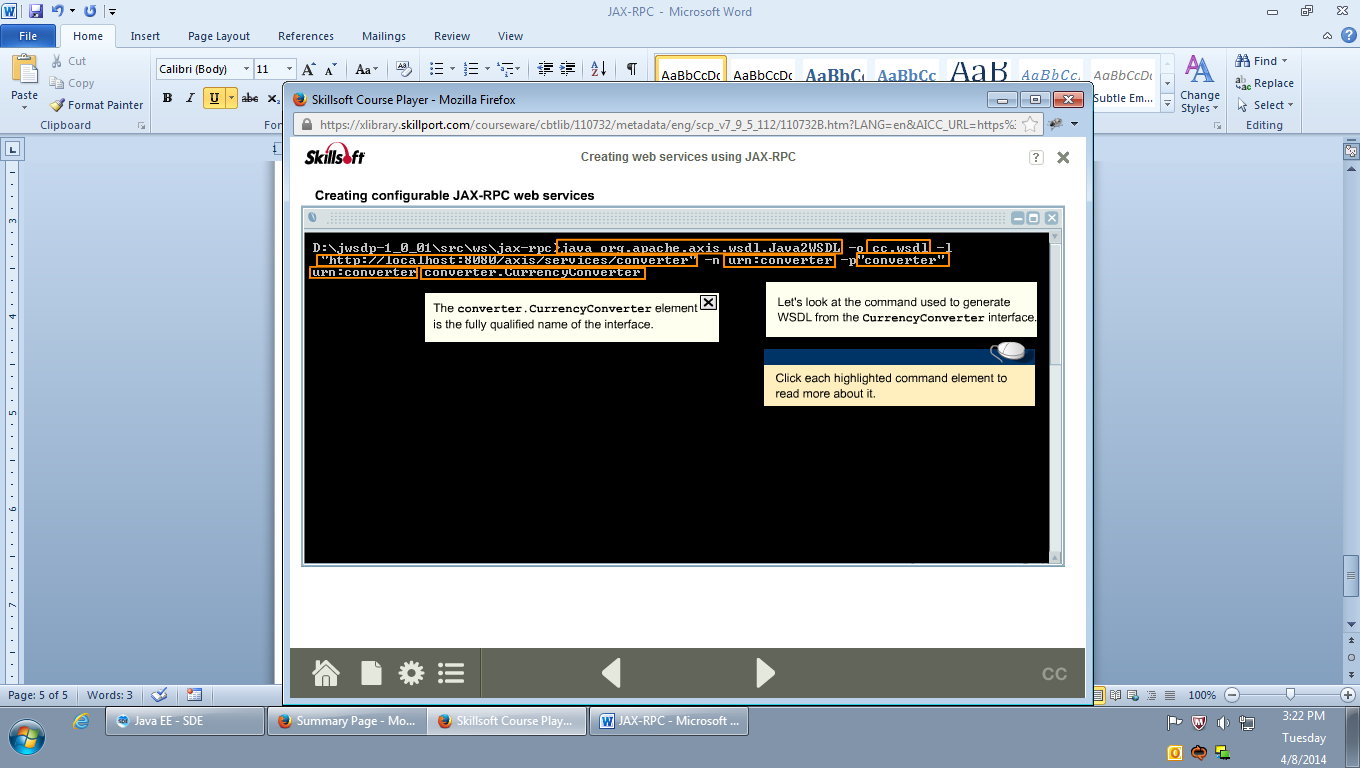


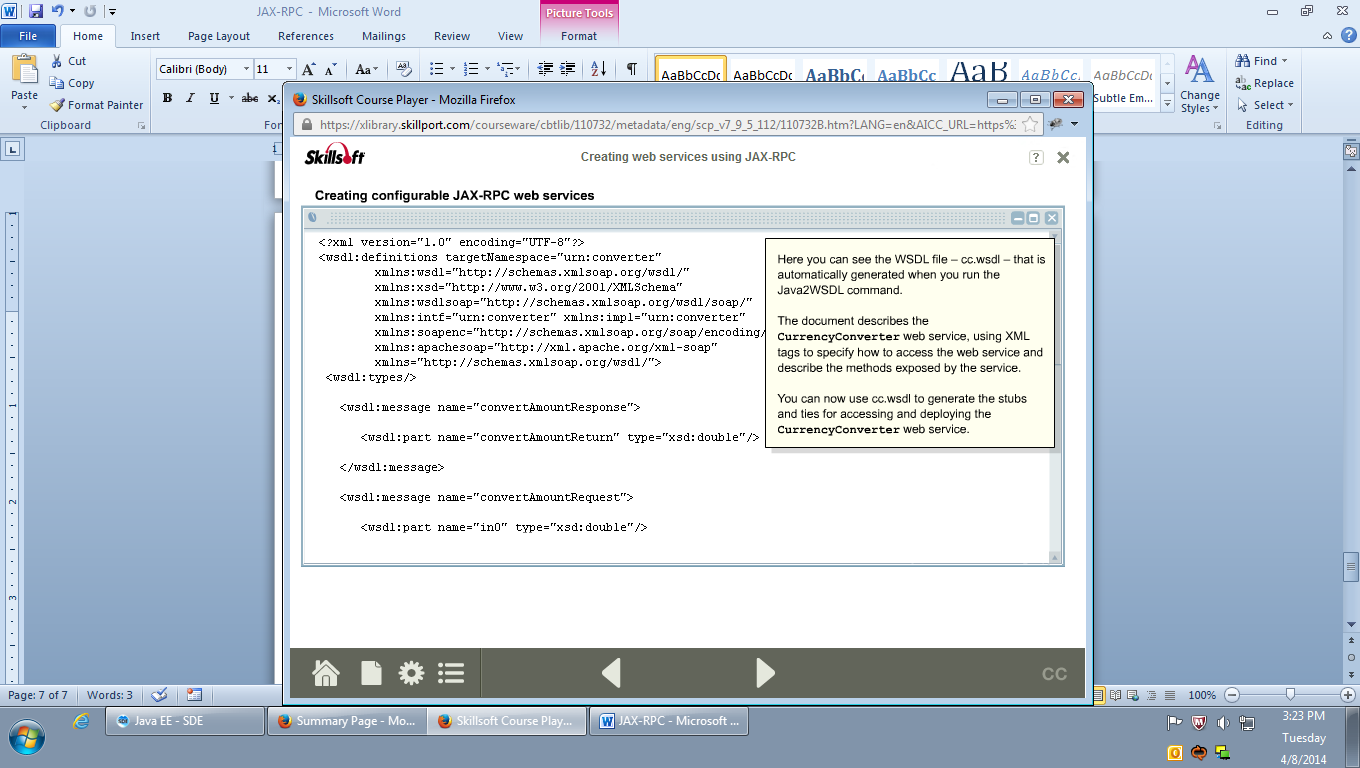












**cc.wsdl:**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<wsdl:definitions targetNamespace=*"urn:converter"*

xmlns:wsdl=*"http://schemas.xmlsoap.org/wsdl/"* xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*

xmlns:wsdlsoap=*"http://schemas.xmlsoap.org/wsdl/soap/"* xmlns:intf=*"urn:converter"*

xmlns:impl=*"urn:converter"* xmlns:soapenc=*"http://schemas.xmlsoap.org/soap/encoding/"*

xmlns:apachesoap=*"http://xml.apache.org/xml-soap"* xmlns=*"http://schemas.xmlsoap.org/wsdl/"*>

<wsdl:types />

<wsdl:message name=*"convertAmountResponse"*>

<wsdl:part name=*"convertAmountReturn"* type=*"xsd:double"* />

</wsdl:message>

<wsdl:message name=*"convertAmountRequest"*>

<wsdl:part name=*"in0"* type=*"xsd:double"* />

</wsdl:message>

<wsdl:portType name=*"CurrencyConverter"*>

<wsdl:operation name=*"convertAmount"* parameterOrder=*"in0"*>

<wsdl:input name=*"convertAmountRequest"* message=*"intf:convertAmountRequest"* />

<wsdl:output name=*"convertAmountResponse"* message=*"intf:convertAmountResponse"* />

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name=*"converterSoapBinding"* type=*"intf:CurrencyConverter"*>

<wsdlsoap:binding style=*"rpc"*

transport=*"http://schemas.xmlsoap.org/soap/http"* />

<wsdl:operation name=*"convertAmount"*>

<wsdlsoap:operation soapAction=*""* />

<wsdl:input name=*"convertAmountRequest"*>

<wsdlsoap:body use=*"encoded"*

encodingStyle=*"http://schemas.xmlsoap.org/soap/encoding/"*

namespace=*"urn:converter"* />

</wsdl:input>

<wsdl:output name=*"convertAmountResponse"*>

<wsdlsoap:body use=*"encoded"*

encodingStyle=*"http://schemas.xmlsoap.org/soap/encoding/"*

namespace=*"urn:converter"* />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name=*"CurrencyConverterService"*>

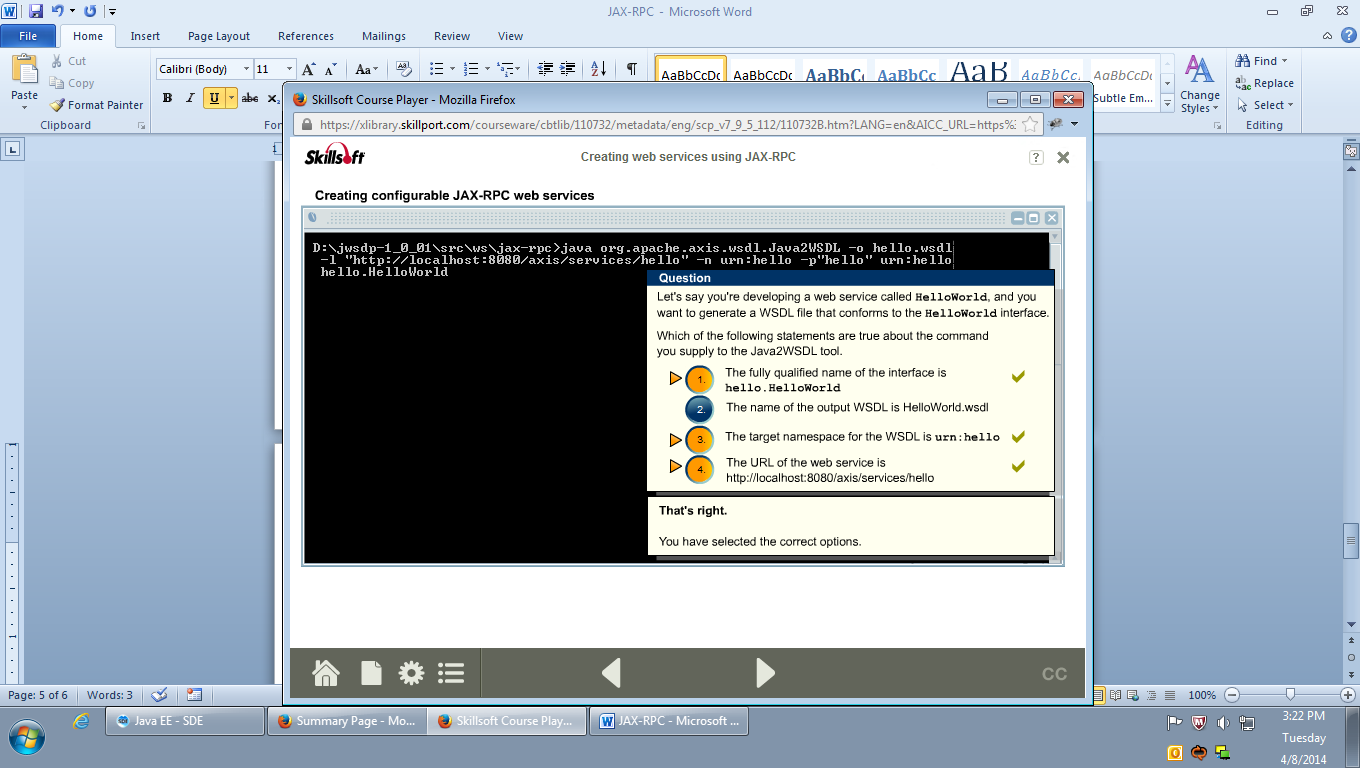
<wsdl:port name=*"converter"* binding=*"intf:converterSoapBinding"*>

<wsdlsoap:address location=*"http://localhost:8080/axis/services/converter"* />

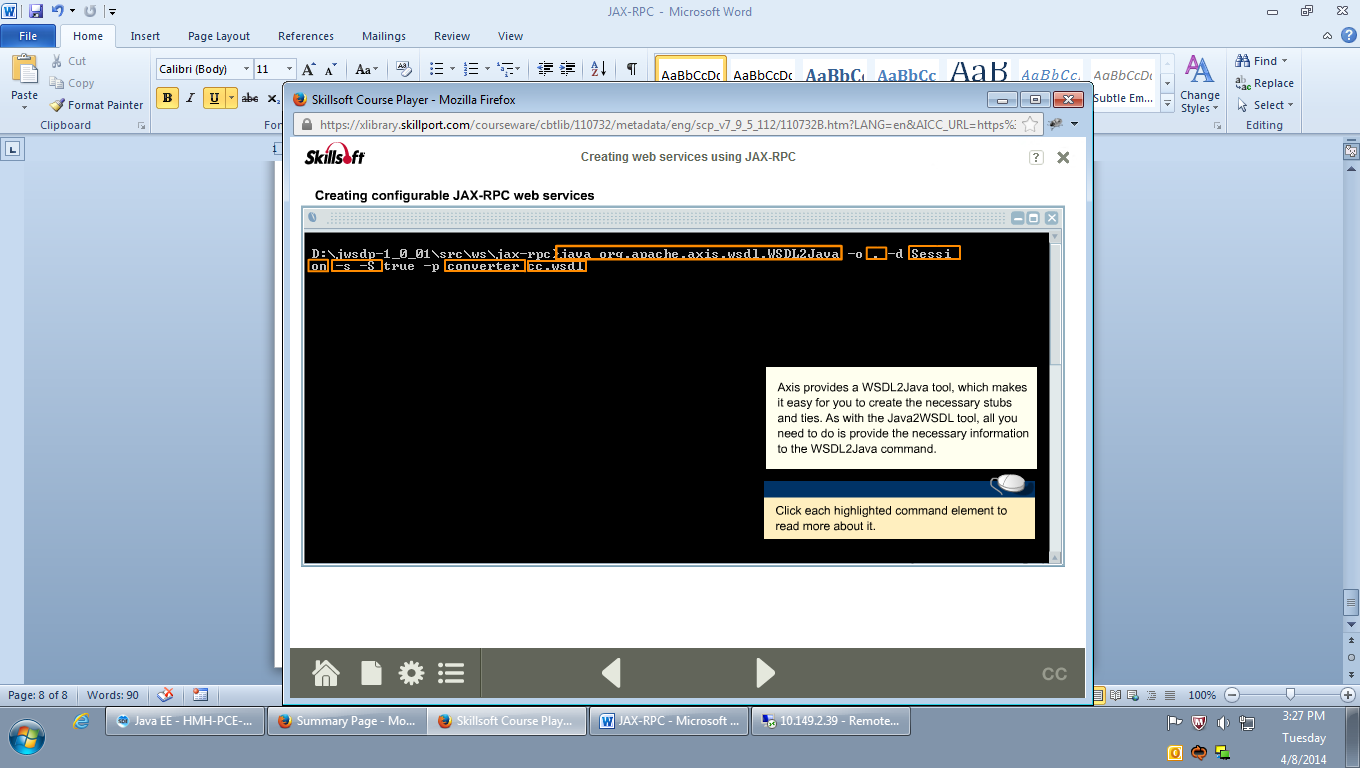
</wsdl:port>

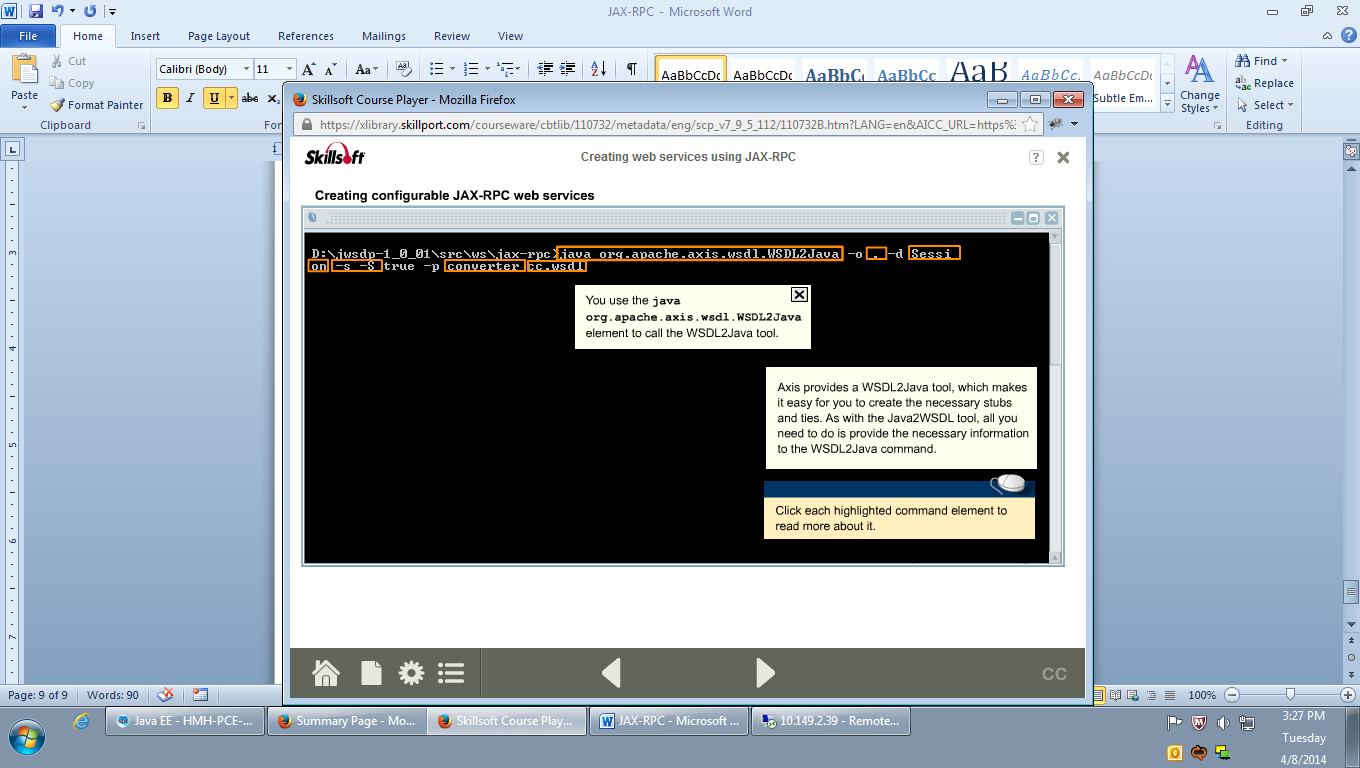
</wsdl:service>

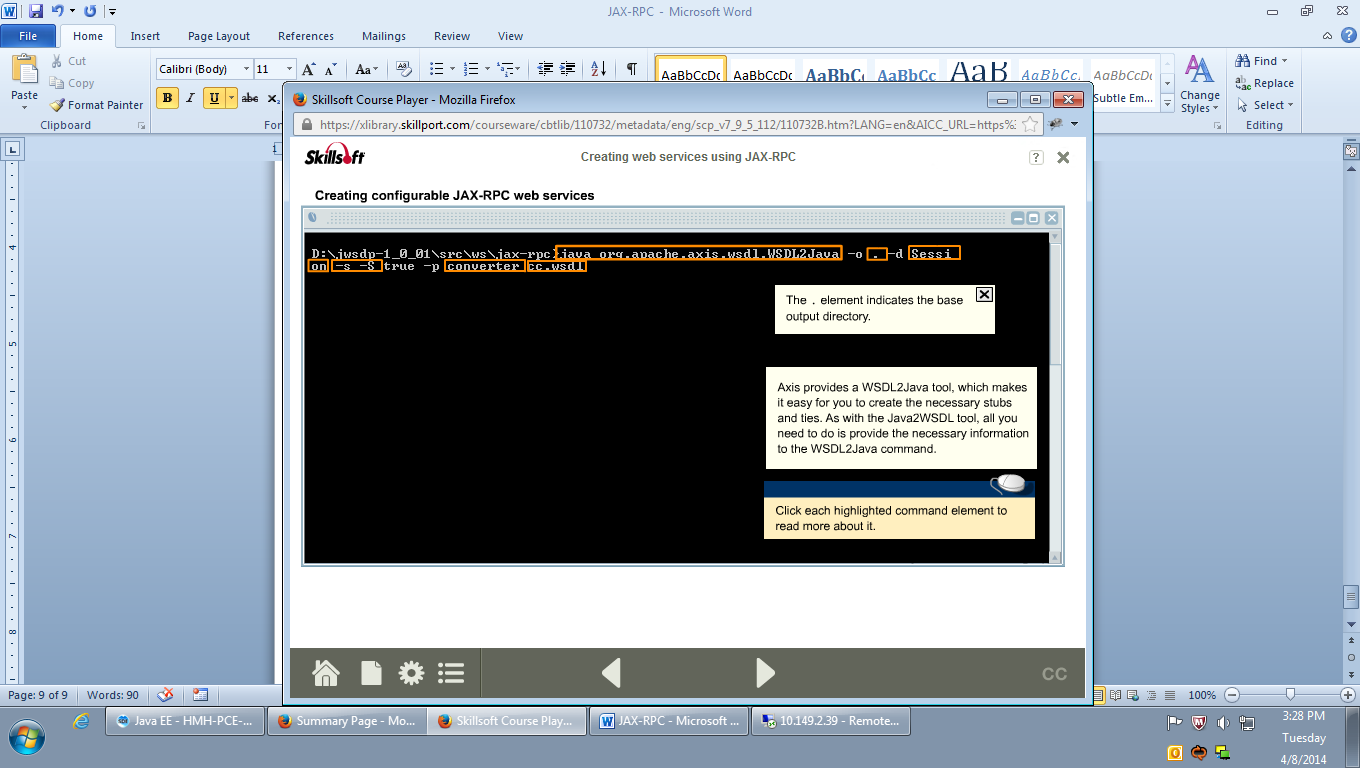
</wsdl:definitions>

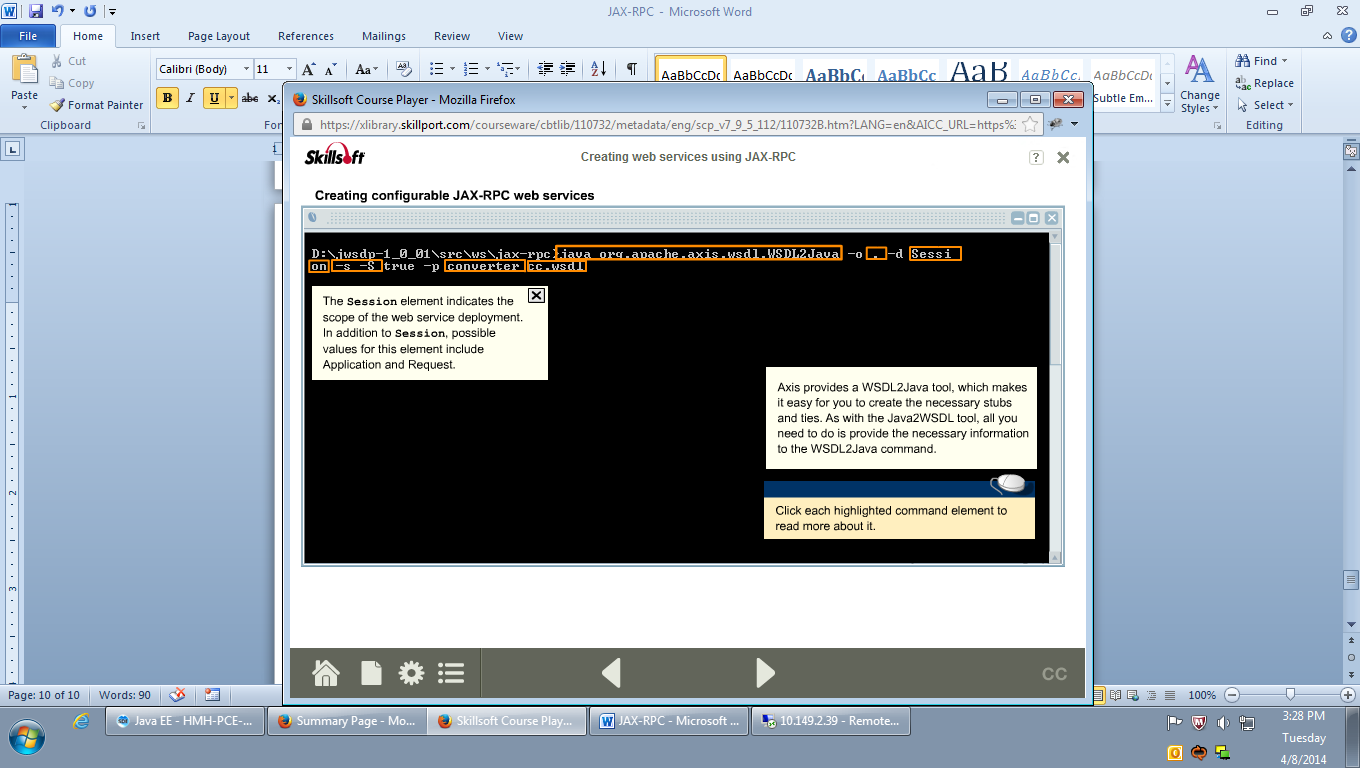


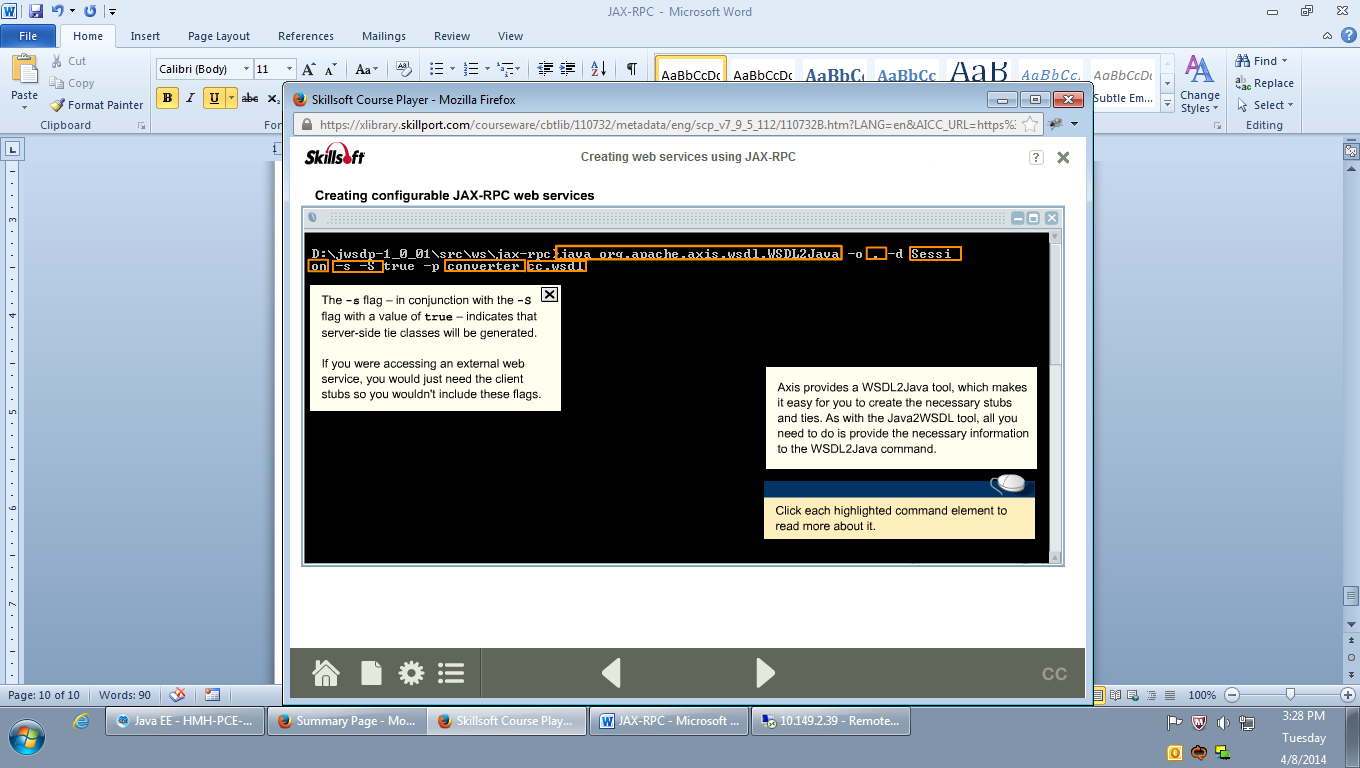
**WSDL-To-Java>>**

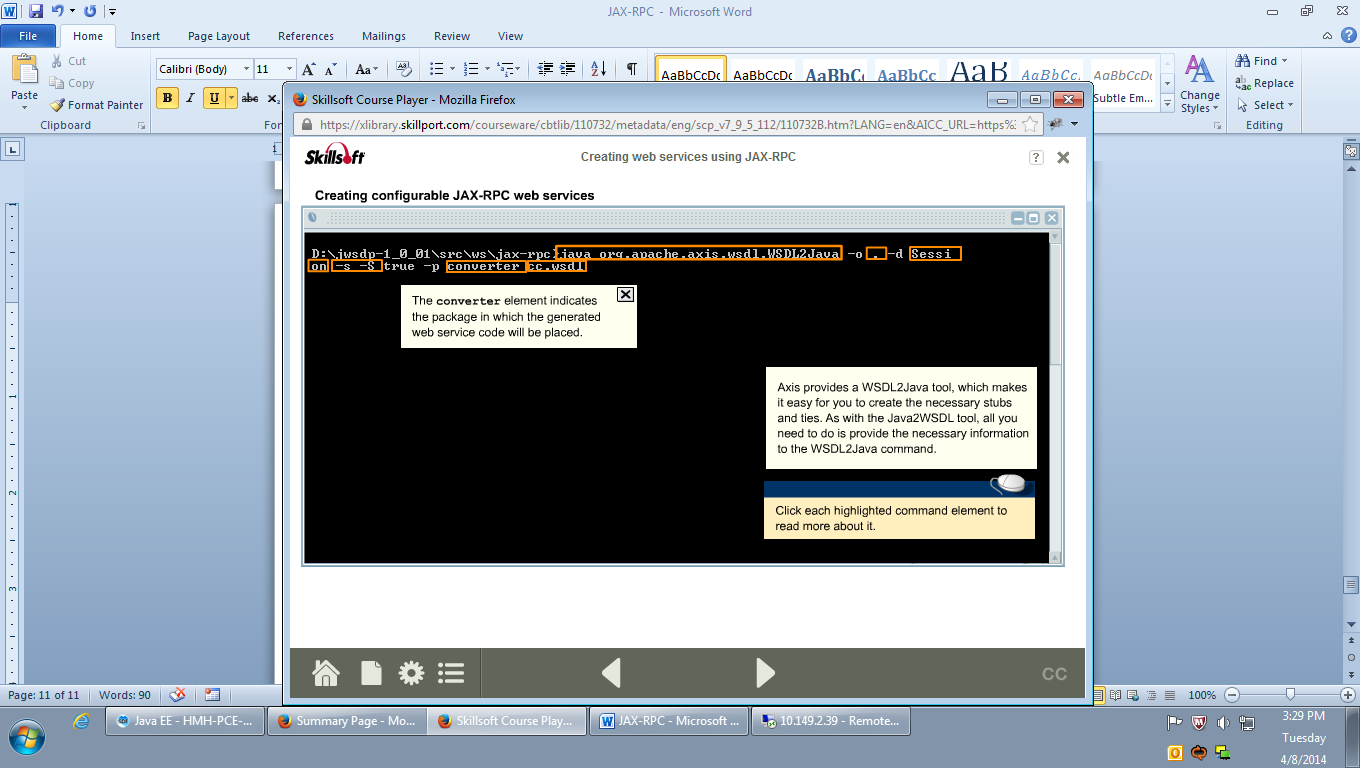


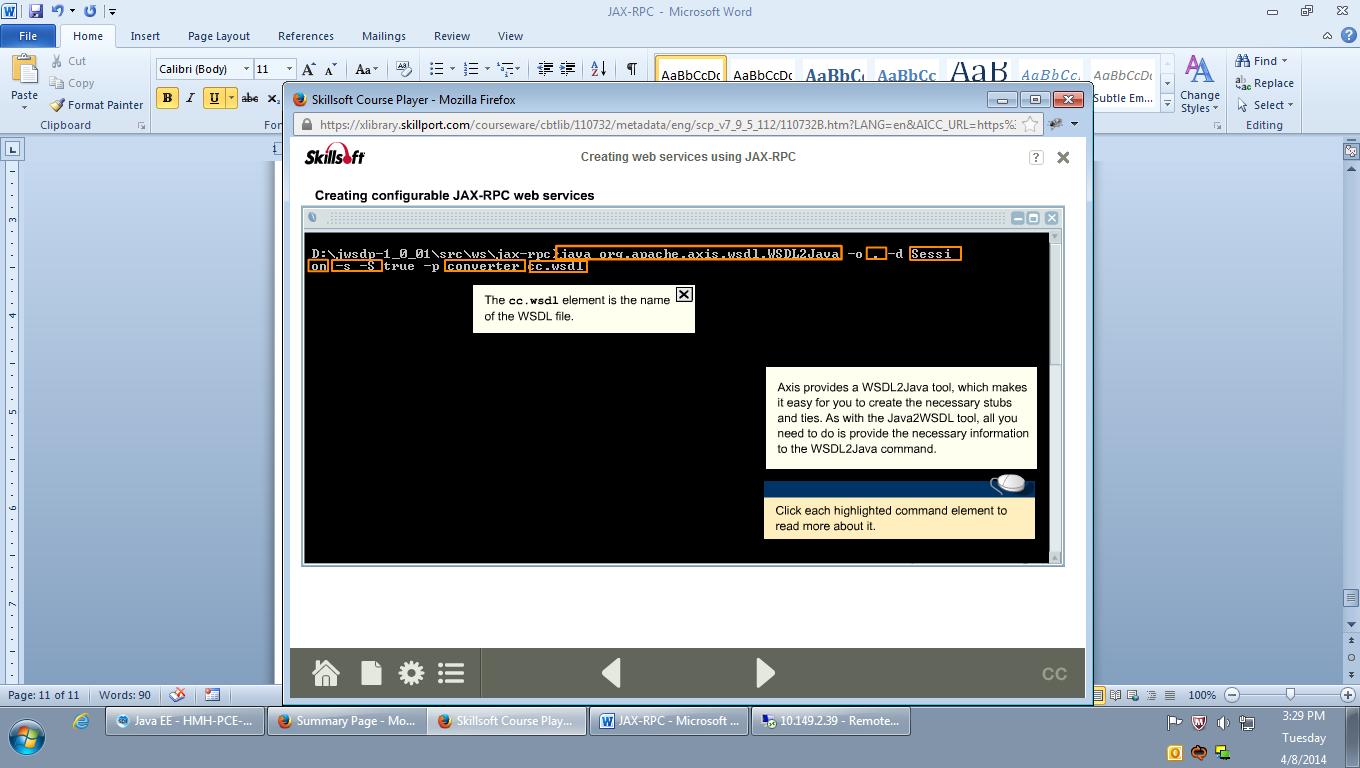


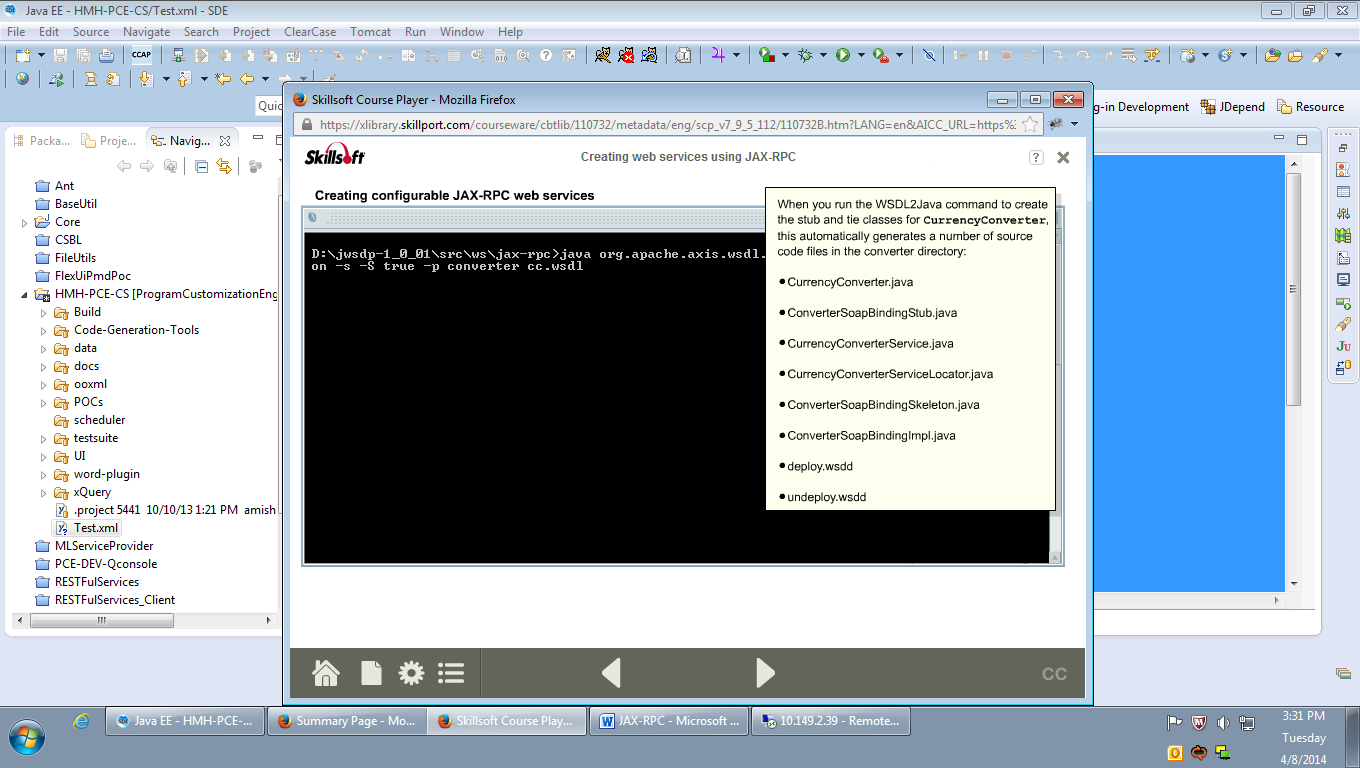


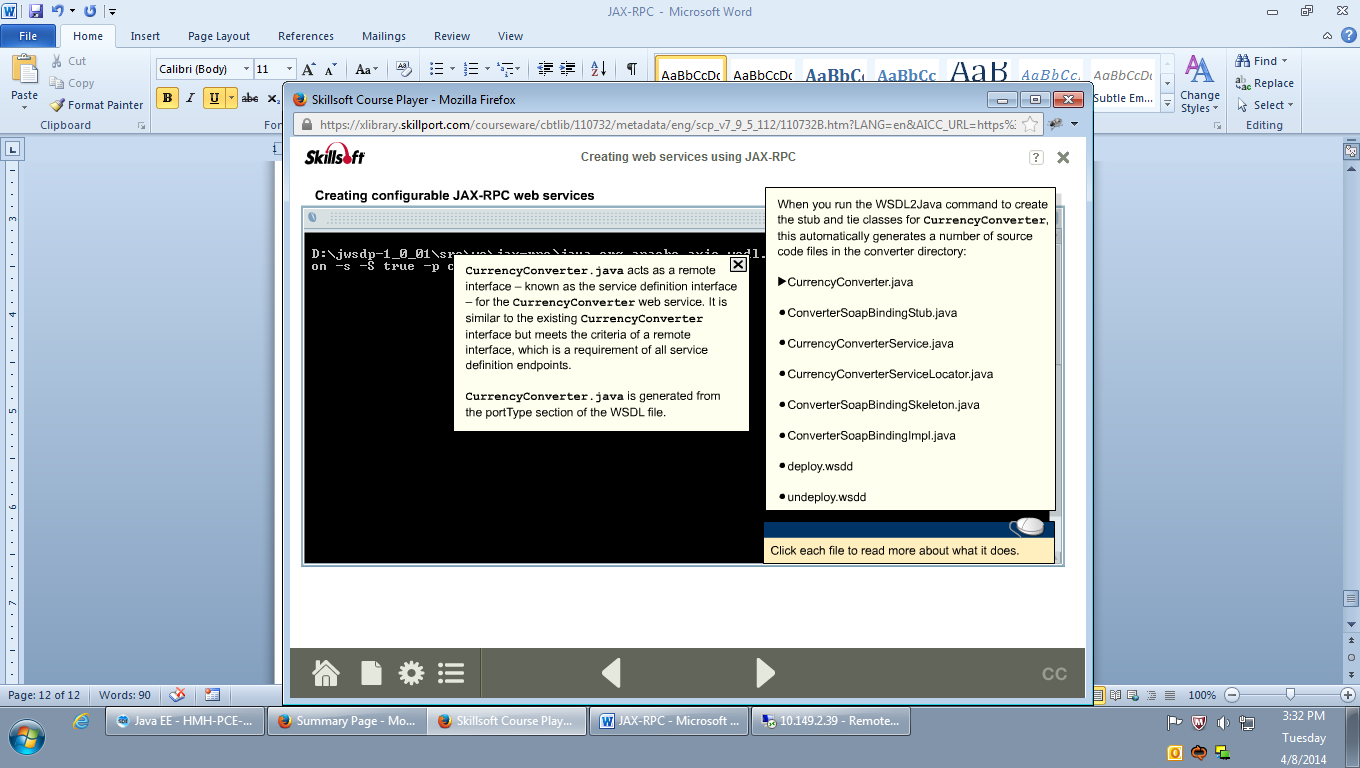




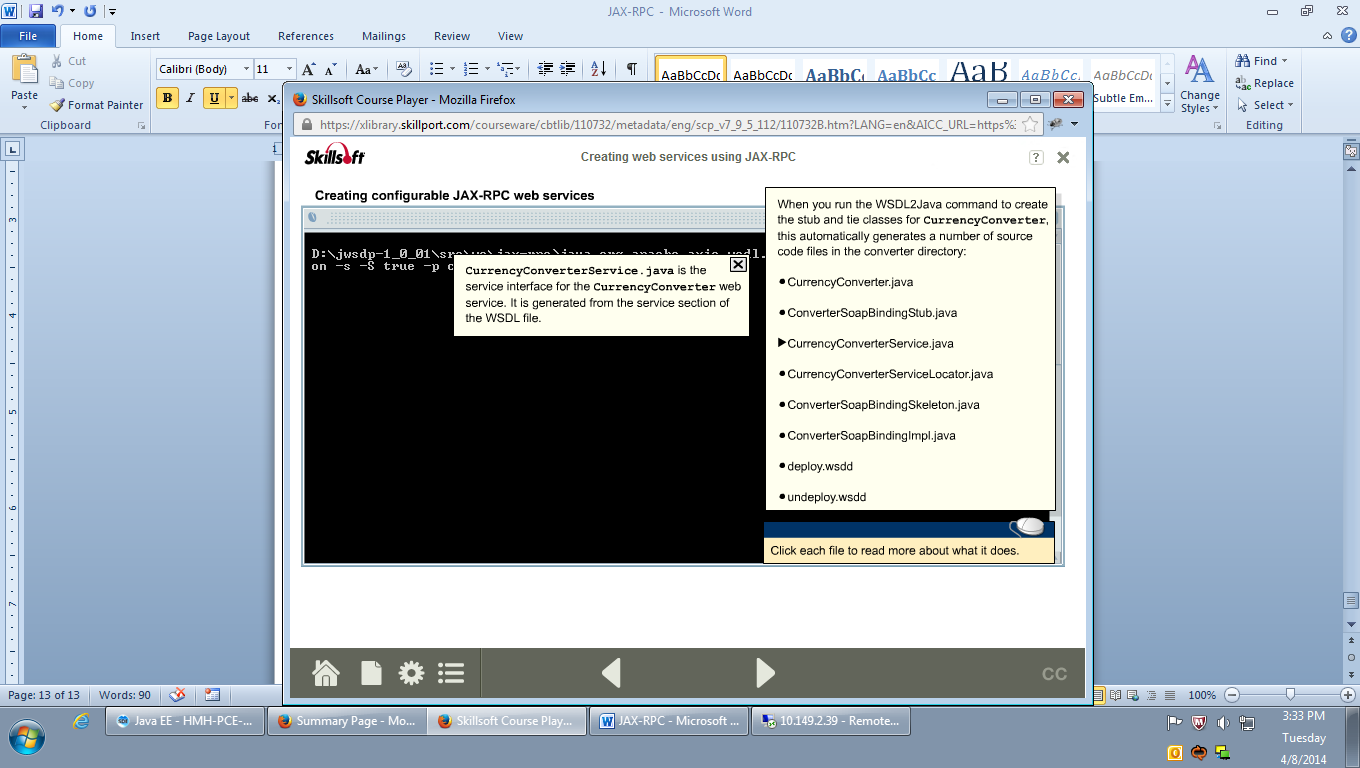


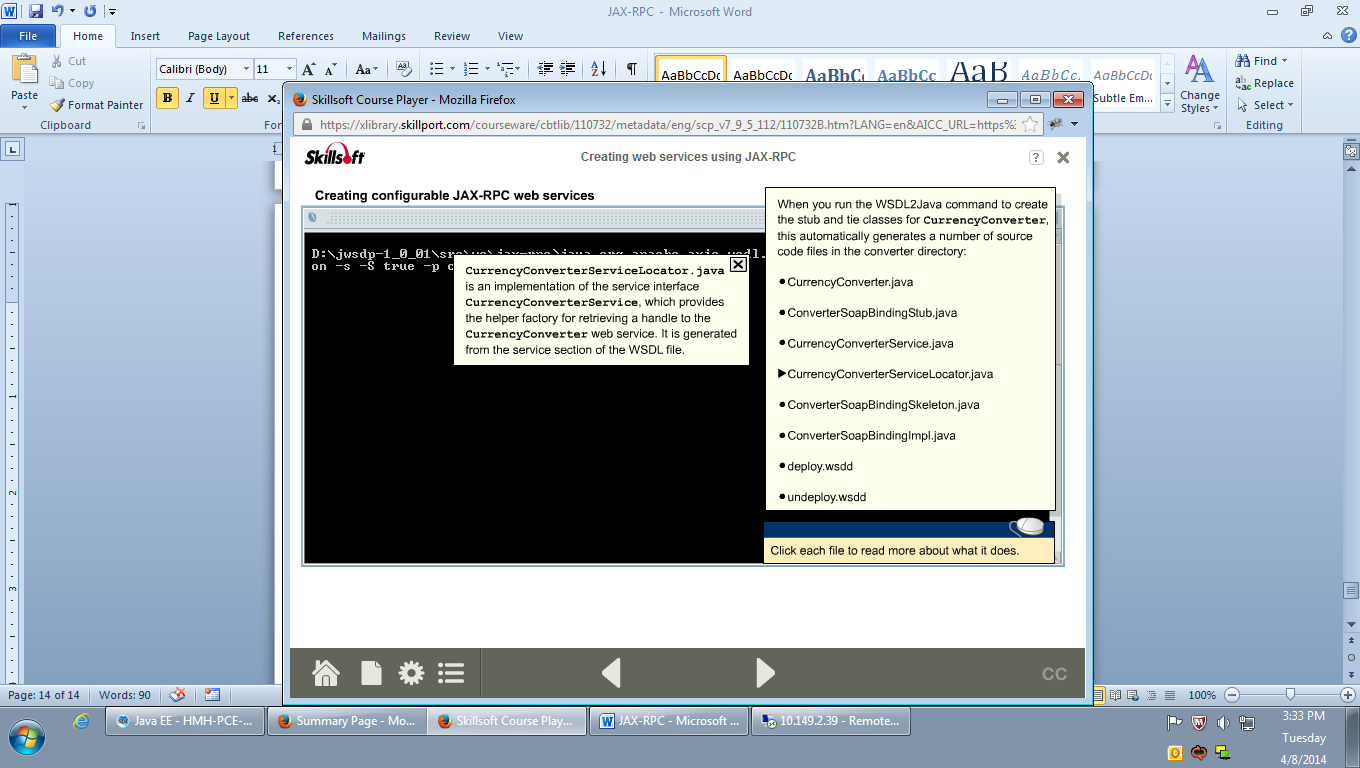




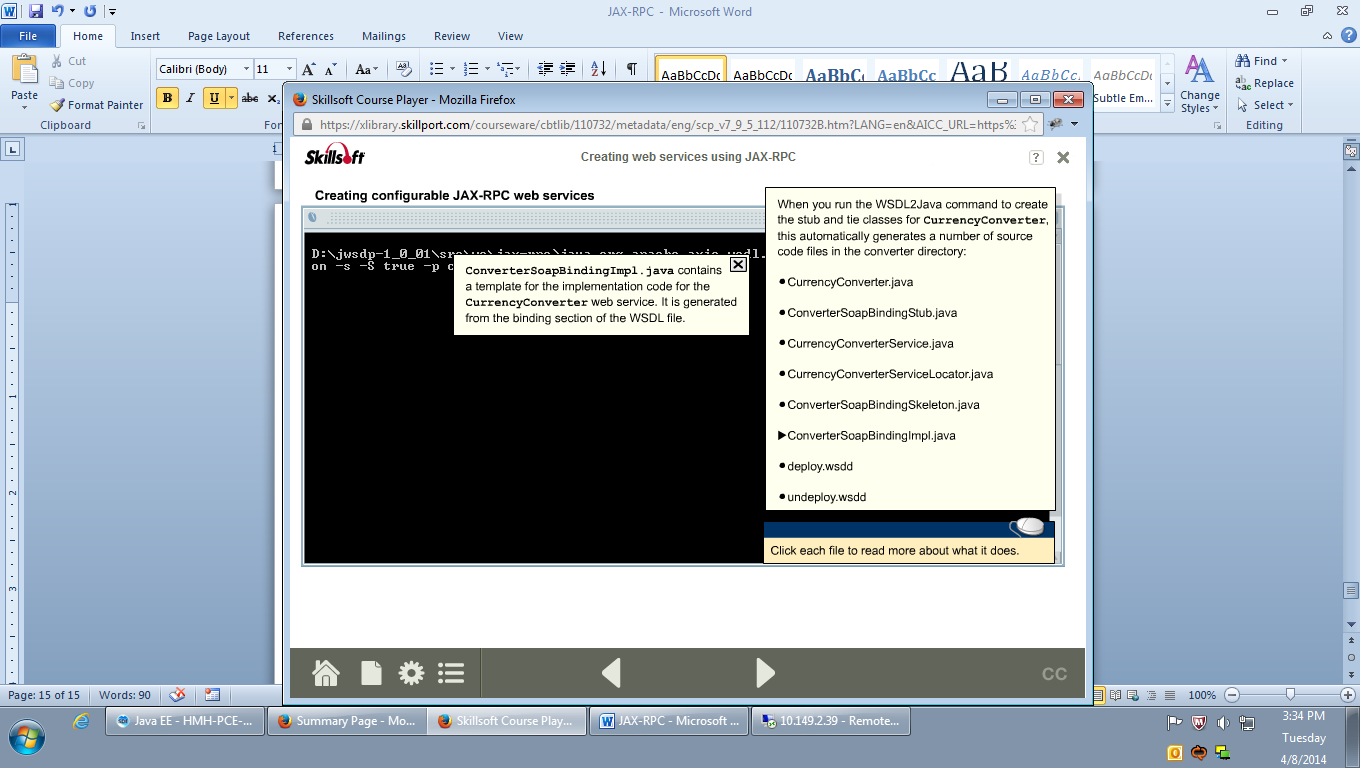


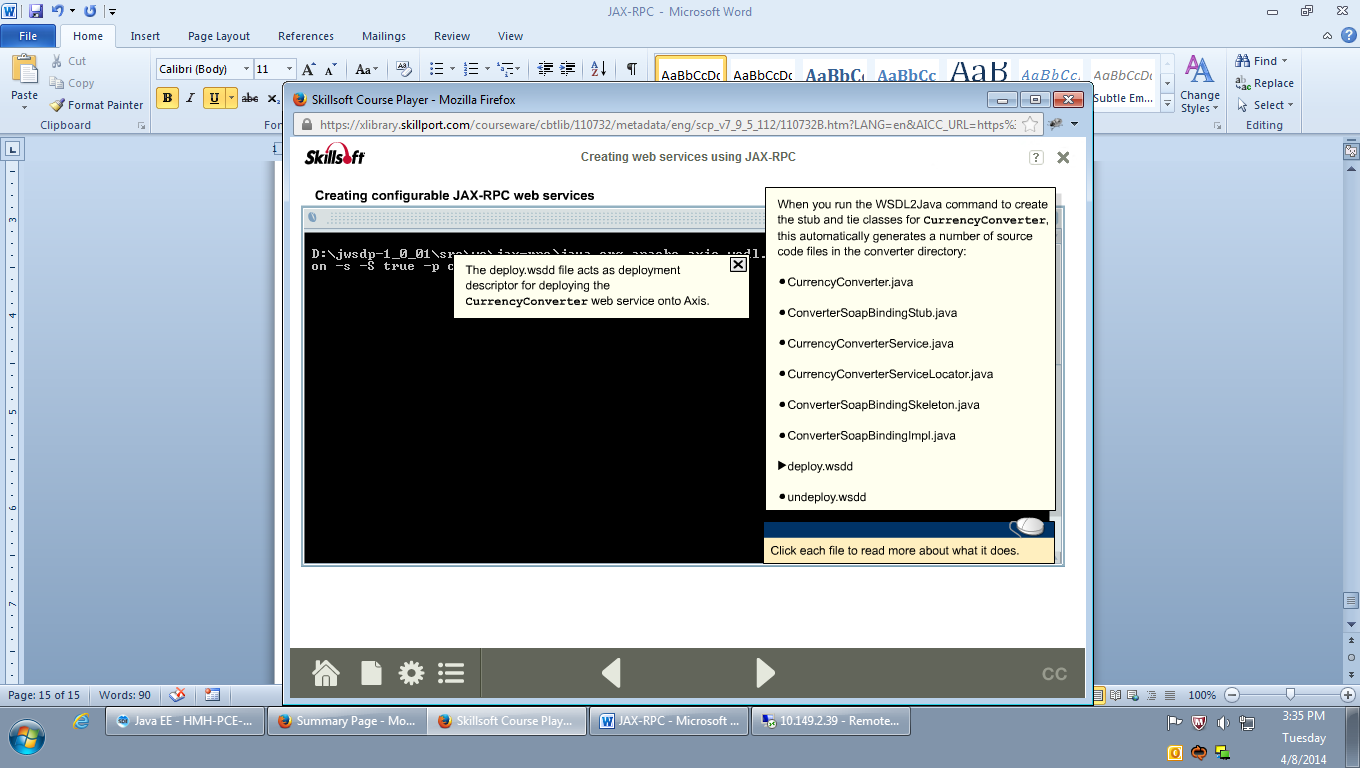


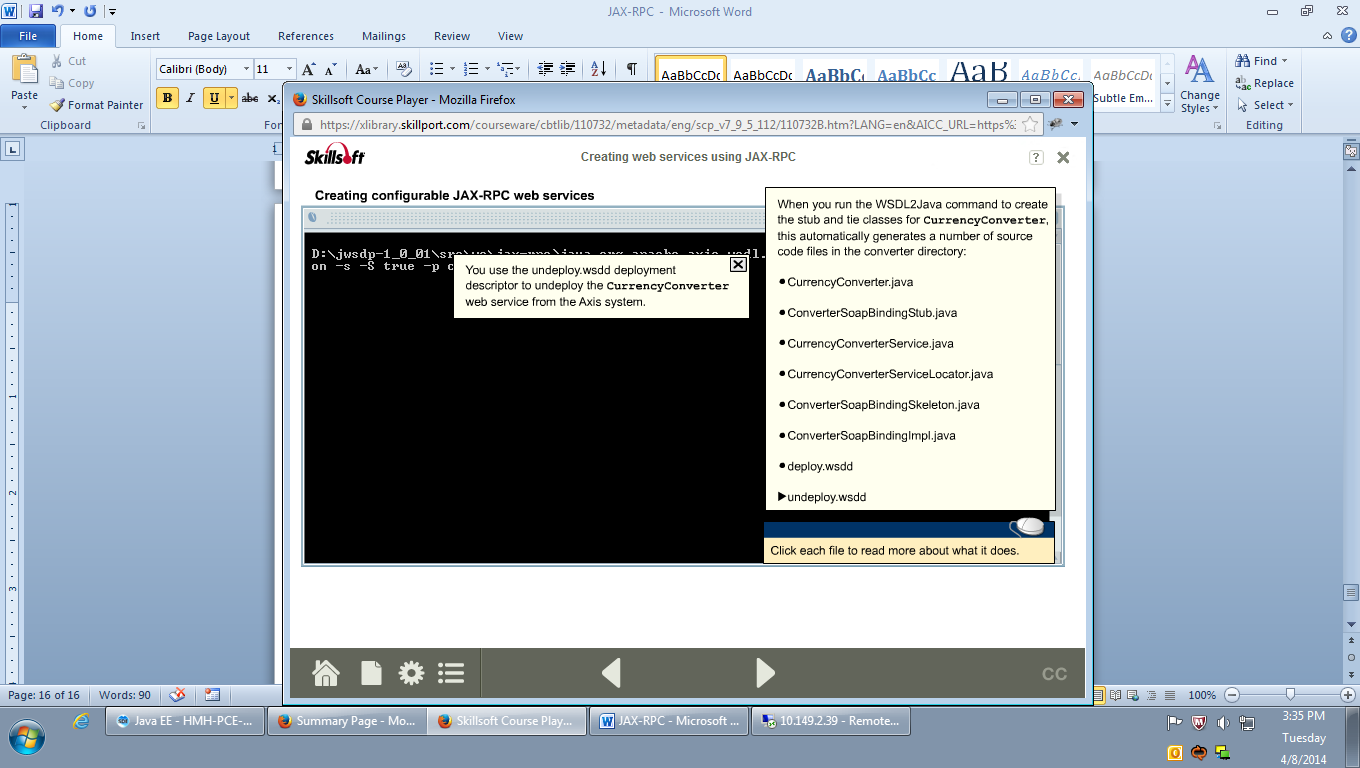


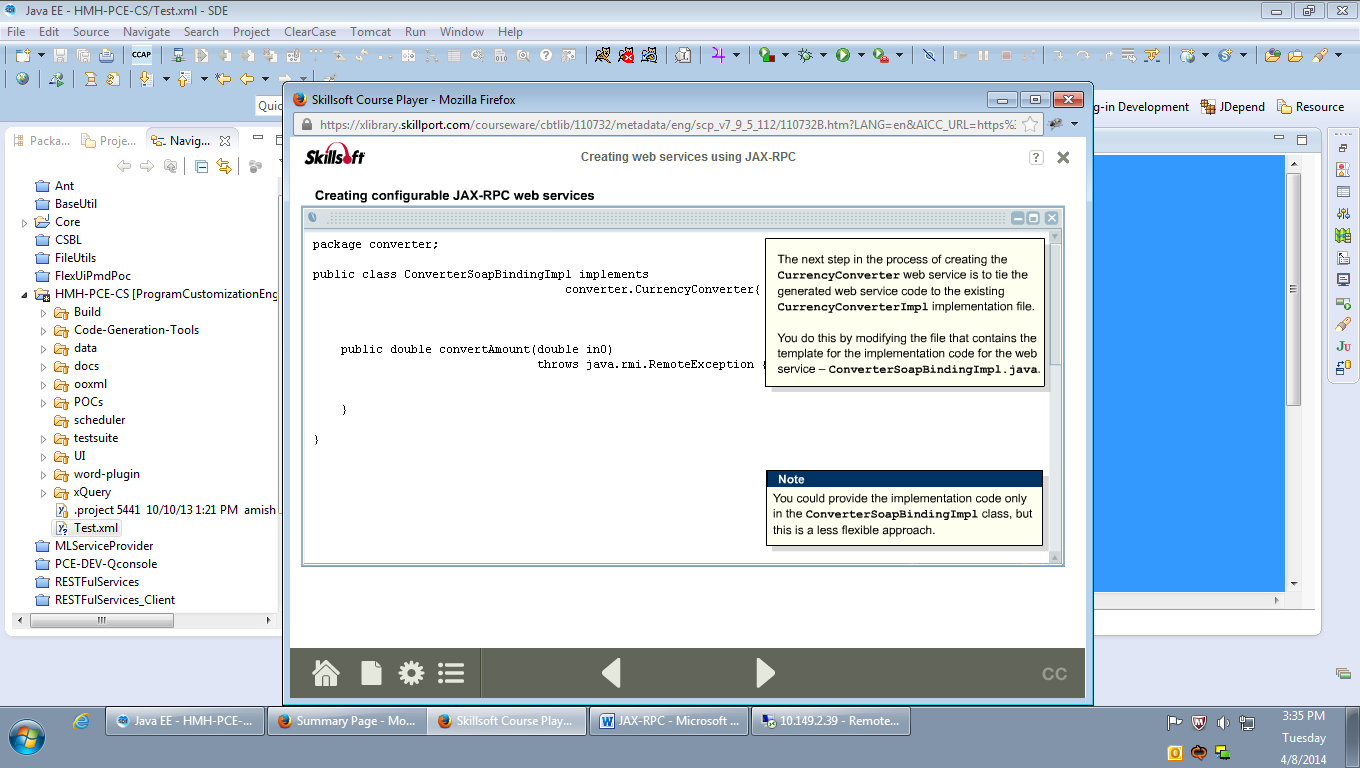


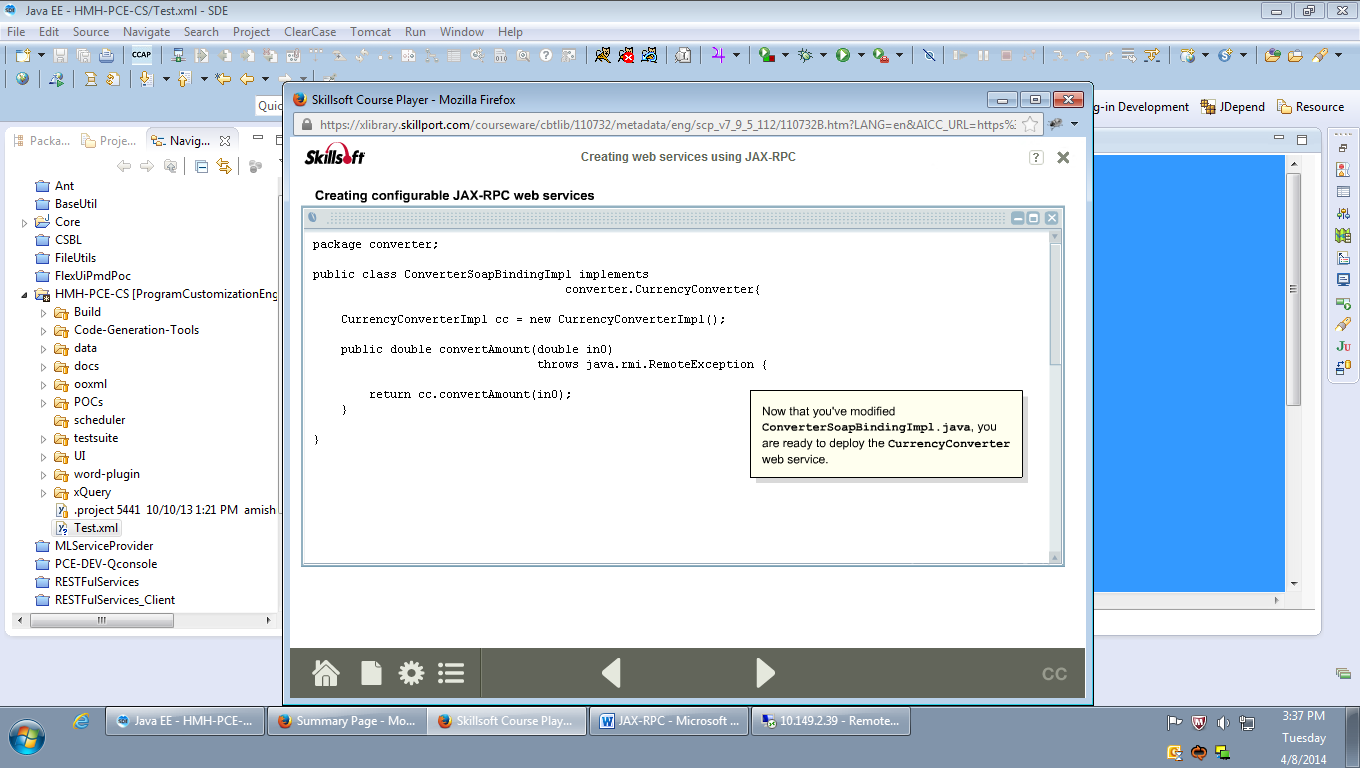




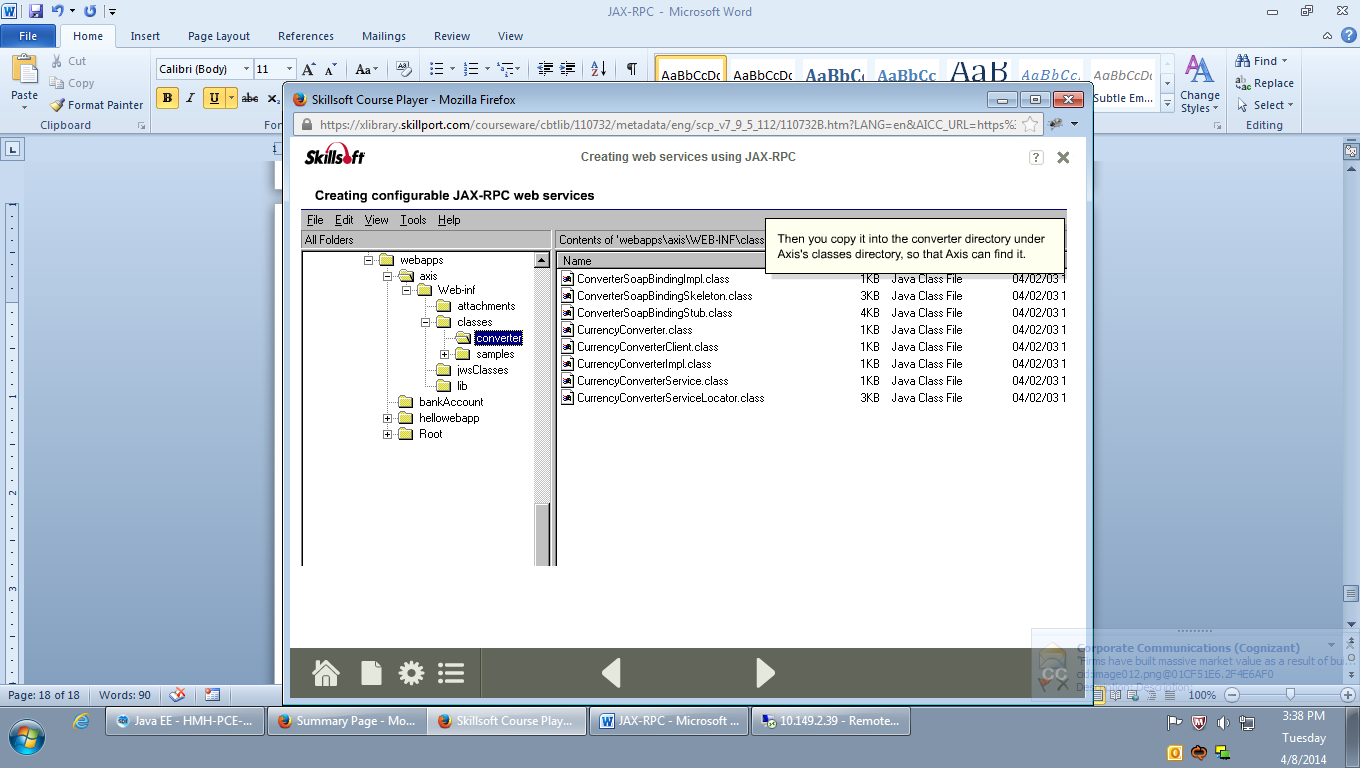


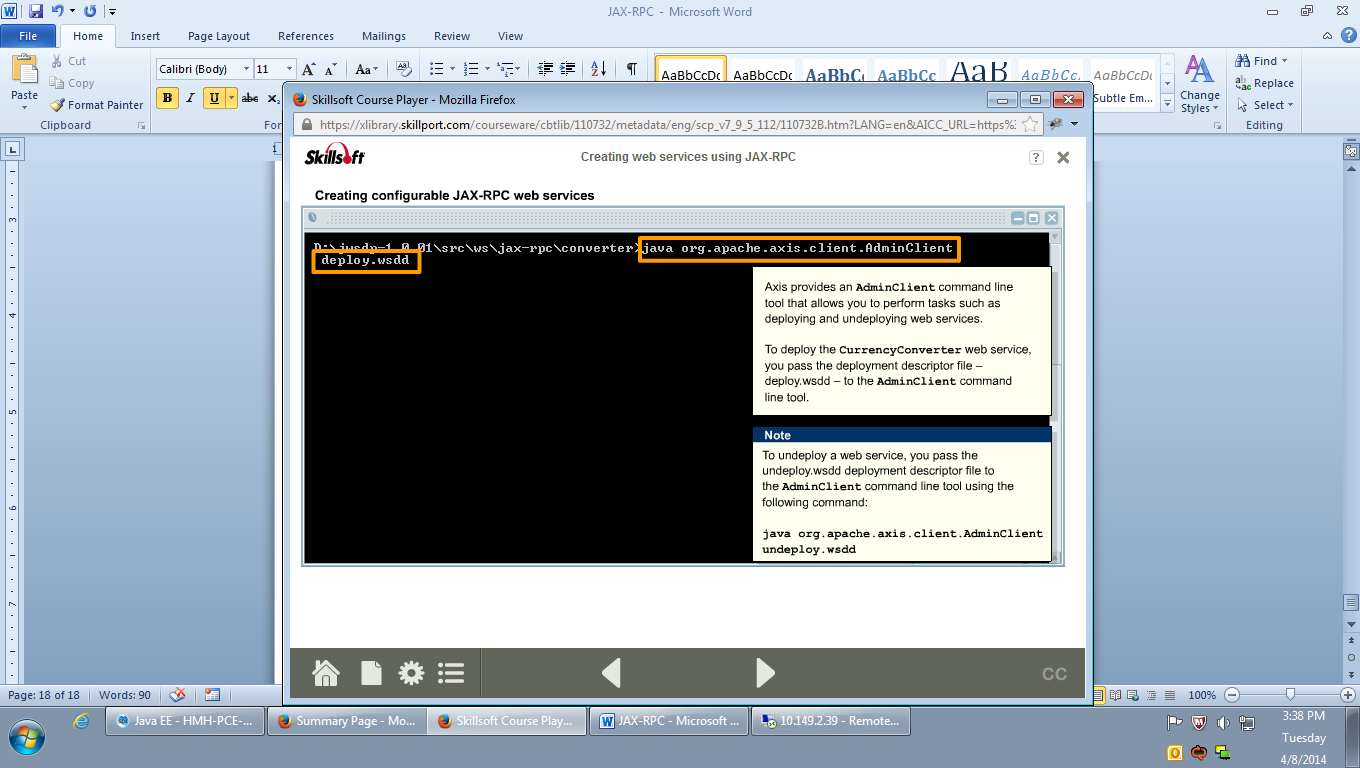












**deploy.wsdd**

<!-- Use this file to deploy some handlers/chains and services -->

<!-- Two ways to do this: -->

<!-- java org.apache.axis.client.AdminClient deploy.wsdd -->

<!-- after the axis server is running -->

<!-- or -->

<!-- java org.apache.axis.utils.Admin client|server deploy.wsdd -->

<!-- from the same directory that the Axis engine runs -->

<deployment

xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<!-- Services from CurrencyConverterService WSDL service -->

<service name="converter" provider="java:RPC">

<parameter name="wsdlTargetNamespace" value="urn:converter"/>

<parameter name="wsdlServiceElement"

value="CurrencyConverterService"/>

<parameter name="wsdlServicePort" value="converter"/>

<parameter name="className"

value="converter.ConverterSoapBindingSkeleton"/>

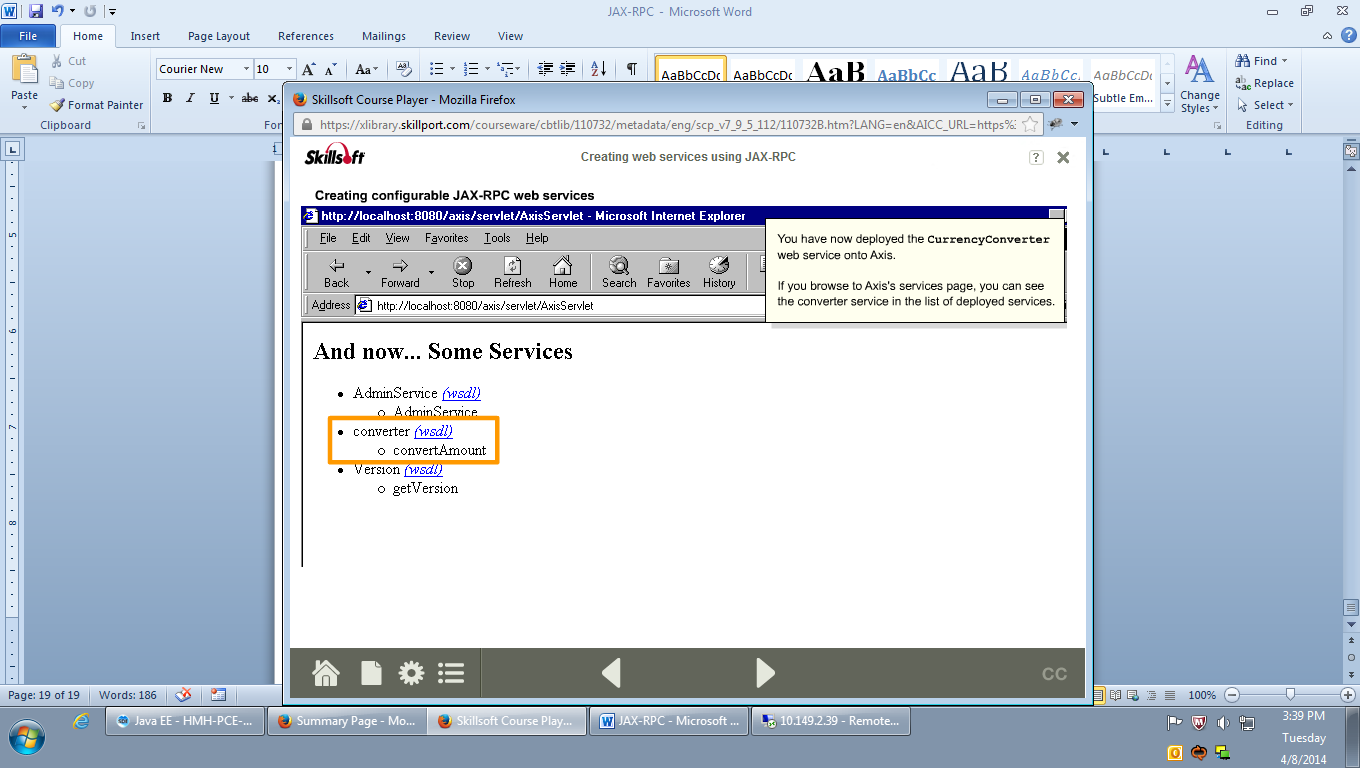
<parameter name="wsdlPortType" value="CurrencyConverter"/>

<parameter name="allowedMethods" value="\*"/>

<parameter name="scope" value="Session"/>

</service>

</deployment>



# Running the Currency Converter web service on Apache Axis

The files provided with this object include the pre-generated stub and skeletons classes for the Currency Converter web service for you to examine if you do not want to run the Apache Axis tools. If you want to run the tools and generate the necessary files from scratch, please follow the process outlined below.

The original interface and implementation class are:

* CurrencyConverter1.java
* CurrencyConverterImpl.java

The WSDL file generated from the interface is:

* cc.wsdl

The files generated from the WSDL file are:

* ConverterSoapBindingImpl.java
* ConverterSoapBindingSkeleton.java
* ConverterSoapBindingStub.java
* CurrencyConverter.java
* CurrencyConverterService.java
* CurrencyConverterServiceLocator.java
* deploy.wsdd
* undeploy.wsdd

The test client class is:

* CurrencyConverterClient.java

The utility file to set up your environment is:

* setclasspath.bat

To compile and deploy the currency converter web service using Apache Axis on Tomcat:

1. Copy the source code files CurrencyConverter1.java and CurrencyConverterImpl.java only to a directory called converter in a suitable location – <SRC>\converter, and save CurrencyConverter1.java as CurrencyConverter.java
2. Edit the batch file setclasspath.bat to reflect your own environment and run it from a command prompt.
3. In the same command prompt, compile the code in <SRC>\converter.
4. From the <SRC> directory run the following command to generate the WSDL:   
   java org.apache.axis.wsdl.Java2WSDL -o cc.wsdl -l "http://localhost:8080/axis/services/converter" -n urn:converter -p"converter" urn:converter converter.CurrencyConverter
5. Once the WSDL is created, from the <SRC> directory run the following command to generate the stub classes:   
   java org.apache.axis.wsdl.WSDL2Java -o . -d Session -s -S true -p converter cc.wsdl
6. Replace the generated version of ConverterSoapBindingImpl.java with the version below and compile all the generated class files
7. Copy all the class files to <TOM\_CAT\_INSTALL\_DIR>\webapps\axis\WEB-INF\classes\converter
8. Return to the converter directory and run the following command to deploy the service:   
   java org.apache.axis.client.AdminClient deploy.wsdd

**Note 1:** To check the service is running, browse to the following URL and you should see it listed as 'converter':   
http://localhost:8080/axis/servlet/AxisServlet

**Note 2:** To test the web service, compile and run the client class - CurrencyConverterClient.java from the converter directory.

**Note 3:** to undeploy the service, type the following command:   
java org.apache.axis.client.AdminClient undeploy.wsdd

## Source code

### cc.wsdl

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<wsdl:definitions targetNamespace=*"urn:converter"*

xmlns:wsdl=*"http://schemas.xmlsoap.org/wsdl/"* xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*

xmlns:wsdlsoap=*"http://schemas.xmlsoap.org/wsdl/soap/"* xmlns:intf=*"urn:converter"*

xmlns:impl=*"urn:converter"* xmlns:soapenc=*"http://schemas.xmlsoap.org/soap/encoding/"*

xmlns:apachesoap=*"http://xml.apache.org/xml-soap"* xmlns=*"http://schemas.xmlsoap.org/wsdl/"*>

<wsdl:types />

<wsdl:message name=*"convertAmountResponse"*>

<wsdl:part name=*"convertAmountReturn"* type=*"xsd:double"* />

</wsdl:message>

<wsdl:message name=*"convertAmountRequest"*>

<wsdl:part name=*"in0"* type=*"xsd:double"* />

</wsdl:message>

<wsdl:portType name=*"CurrencyConverter"*>

<wsdl:operation name=*"convertAmount"* parameterOrder=*"in0"*>

<wsdl:input name=*"convertAmountRequest"* message=*"intf:convertAmountRequest"* />

<wsdl:output name=*"convertAmountResponse"* message=*"intf:convertAmountResponse"* />

</wsdl:operation>

</wsdl:portType>

<wsdl:binding name=*"converterSoapBinding"* type=*"intf:CurrencyConverter"*>

<wsdlsoap:binding style=*"rpc"*

transport=*"http://schemas.xmlsoap.org/soap/http"* />

<wsdl:operation name=*"convertAmount"*>

<wsdlsoap:operation soapAction=*""* />

<wsdl:input name=*"convertAmountRequest"*>

<wsdlsoap:body use=*"encoded"*

encodingStyle=*"http://schemas.xmlsoap.org/soap/encoding/"*

namespace=*"urn:converter"* />

</wsdl:input>

<wsdl:output name=*"convertAmountResponse"*>

<wsdlsoap:body use=*"encoded"*

encodingStyle=*"http://schemas.xmlsoap.org/soap/encoding/"*

namespace=*"urn:converter"* />

</wsdl:output>

</wsdl:operation>

</wsdl:binding>

<wsdl:service name=*"CurrencyConverterService"*>

<wsdl:port name=*"converter"* binding=*"intf:converterSoapBinding"*>

<wsdlsoap:address location=*"http://localhost:8080/axis/services/converter"* />

</wsdl:port>

</wsdl:service>

</wsdl:definitions>

### deploy.wsdd

<!-- Use this file to deploy some handlers/chains and services -->

<!-- Two ways to do this: -->

<!-- java org.apache.axis.client.AdminClient deploy.wsdd -->

<!-- after the axis server is running -->

<!-- or -->

<!-- java org.apache.axis.utils.Admin client|server deploy.wsdd -->

<!-- from the same directory that the Axis engine runs -->

<deployment

xmlns="http://xml.apache.org/axis/wsdd/"

xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<!-- Services from CurrencyConverterService WSDL service -->

<service name="converter" provider="java:RPC">

<parameter name="wsdlTargetNamespace" value="urn:converter"/>

<parameter name="wsdlServiceElement"

value="CurrencyConverterService"/>

<parameter name="wsdlServicePort" value="converter"/>

<parameter name="className"

value="converter.ConverterSoapBindingSkeleton"/>

<parameter name="wsdlPortType" value="CurrencyConverter"/>

<parameter name="allowedMethods" value="\*"/>

<parameter name="scope" value="Session"/>

</service>

</deployment>

### ConverterSoapBindingImpl.java

/\*\*

\* ConverterSoapBindingImpl.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public class ConverterSoapBindingImpl implements

converter.CurrencyConverter{

CurrencyConverterImpl cc = new CurrencyConverterImpl();

public double convertAmount(double in0)

throws java.rmi.RemoteException {

return cc.convertAmount(in0);

}

}

### ConverterSoapBindingSkeleton.java

/\*\*

\* ConverterSoapBindingSkeleton.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public class ConverterSoapBindingSkeleton implements converter.CurrencyConverter, org.apache.axis.wsdl.Skeleton {

private converter.CurrencyConverter impl;

private static java.util.Map \_myOperations = new java.util.Hashtable();

private static java.util.Collection \_myOperationsList = new java.util.ArrayList();

/\*\*

\* Returns List of OperationDesc objects with this name

\*/

public static java.util.List getOperationDescByName(java.lang.String methodName) {

return (java.util.List)\_myOperations.get(methodName);

}

/\*\*

\* Returns Collection of OperationDescs

\*/

public static java.util.Collection getOperationDescs() {

return \_myOperationsList;

}

static {

org.apache.axis.description.OperationDesc \_oper;

org.apache.axis.description.ParameterDesc [] \_params;

\_params = new org.apache.axis.description.ParameterDesc [] {

new org.apache.axis.description.ParameterDesc(new javax.xml.namespace.QName("", "in0"), org.apache.axis.description.ParameterDesc.IN, new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "double"), double.class),

};

\_oper = new org.apache.axis.description.OperationDesc("convertAmount", \_params, new javax.xml.namespace.QName("", "convertAmountReturn"));

\_oper.setReturnType(new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "double"));

\_oper.setElementQName(new javax.xml.namespace.QName("urn:converter", "convertAmount"));

\_oper.setSoapAction("");

\_myOperationsList.add(\_oper);

if (\_myOperations.get("convertAmount") == null) {

\_myOperations.put("convertAmount", new java.util.ArrayList());

}

((java.util.List)\_myOperations.get("convertAmount")).add(\_oper);

}

public ConverterSoapBindingSkeleton() {

this.impl = new converter.ConverterSoapBindingImpl();

}

public ConverterSoapBindingSkeleton(converter.CurrencyConverter impl) {

this.impl = impl;

}

public double convertAmount(double in0) throws java.rmi.RemoteException

{

double ret = impl.convertAmount(in0);

return ret;

}

}

### ConverterSoapBindingStub.java

/\*\*

\* ConverterSoapBindingStub.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public class ConverterSoapBindingStub extends org.apache.axis.client.Stub implements converter.CurrencyConverter {

private java.util.Vector cachedSerClasses = new java.util.Vector();

private java.util.Vector cachedSerQNames = new java.util.Vector();

private java.util.Vector cachedSerFactories = new java.util.Vector();

private java.util.Vector cachedDeserFactories = new java.util.Vector();

public ConverterSoapBindingStub() throws org.apache.axis.AxisFault {

this(null);

}

public ConverterSoapBindingStub(java.net.URL endpointURL, javax.xml.rpc.Service service) throws org.apache.axis.AxisFault {

this(service);

super.cachedEndpoint = endpointURL;

}

public ConverterSoapBindingStub(javax.xml.rpc.Service service) throws org.apache.axis.AxisFault {

if (service == null) {

super.service = new org.apache.axis.client.Service();

} else {

super.service = service;

}

}

private org.apache.axis.client.Call createCall() throws java.rmi.RemoteException {

try {

org.apache.axis.client.Call \_call =

(org.apache.axis.client.Call) super.service.createCall();

if (super.maintainSessionSet) {

\_call.setMaintainSession(super.maintainSession);

}

if (super.cachedUsername != null) {

\_call.setUsername(super.cachedUsername);

}

if (super.cachedPassword != null) {

\_call.setPassword(super.cachedPassword);

}

if (super.cachedEndpoint != null) {

\_call.setTargetEndpointAddress(super.cachedEndpoint);

}

if (super.cachedTimeout != null) {

\_call.setTimeout(super.cachedTimeout);

}

if (super.cachedPortName != null) {

\_call.setPortName(super.cachedPortName);

}

java.util.Enumeration keys = super.cachedProperties.keys();

while (keys.hasMoreElements()) {

java.lang.String key = (java.lang.String) keys.nextElement();

if(\_call.isPropertySupported(key))

\_call.setProperty(key, super.cachedProperties.get(key));

else

\_call.setScopedProperty(key, super.cachedProperties.get(key));

}

return \_call;

}

catch (java.lang.Throwable t) {

throw new org.apache.axis.AxisFault("Failure trying to get the Call object", t);

}

}

public double convertAmount(double in0) throws java.rmi.RemoteException {

if (super.cachedEndpoint == null) {

throw new org.apache.axis.NoEndPointException();

}

org.apache.axis.client.Call \_call = createCall();

\_call.addParameter(new javax.xml.namespace.QName("", "in0"), new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "double"), double.class, javax.xml.rpc.ParameterMode.IN);

\_call.setReturnType(new javax.xml.namespace.QName("http://www.w3.org/2001/XMLSchema", "double"), double.class);

\_call.setUseSOAPAction(true);

\_call.setSOAPActionURI("");

\_call.setOperationStyle("rpc");

\_call.setOperationName(new javax.xml.namespace.QName("urn:converter", "convertAmount"));

java.lang.Object \_resp = \_call.invoke(new java.lang.Object[] {new java.lang.Double(in0)});

if (\_resp instanceof java.rmi.RemoteException) {

throw (java.rmi.RemoteException)\_resp;

}

else {

try {

return ((java.lang.Double) \_resp).doubleValue();

} catch (java.lang.Exception \_exception) {

return ((java.lang.Double) org.apache.axis.utils.JavaUtils.convert(\_resp, double.class)).doubleValue();

}

}

}

}

### CurrencyConverter.java

/\*\*

\* CurrencyConverter.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public interface CurrencyConverter extends java.rmi.Remote {

public double convertAmount(double in0) throws java.rmi.RemoteException;

}

### CurrencyConverter1.java

/\*\*

\* A simple interface for a class that converts

\* US Dollar amounts into Euros.

\*/

package converter;

public interface CurrencyConverter {

/\*\*

\* Method to calculate the converted amount

\* @param double the converted amount

\*/

public double convertAmount(double amount) ;

}

### CurrencyConverterClient.java

package converter;

/\*

\* Client class that calls the currency converter web service

\* using the generated stubs.

\*/

public class CurrencyConverterClient {

public static void main(String [] args) throws Exception {

// Create a service

CurrencyConverterService service =

new CurrencyConverterServiceLocator();

// Retrieve a stub to the service

CurrencyConverter cc = service.getconverter();

// Make a call to the service

System.out.println("Convert 10.00 = "

+ cc.convertAmount(10.00));

}

}

### CurrencyConverterImpl.java

/\*

\* CurrencyConverter.java

\*

\* Created on 04 Feb 2003, 15:51

\*/

package converter;

/\*\*

\* A simple class to convert US Dollar amounts into Euros.

\*/

public class CurrencyConverterImpl {

private double amount = 0;

//conversion rate as of 04 Feb, 2003

private final static double rate = 0.925252;

/\*\*

\* Method to calculate the converted amount

\* @param double the converted amount

\*/

public double convertAmount(double amnt) {

amount = amnt \* rate;

return amount;

}

}

### CurrencyConverterService.java

/\*\*

\* CurrencyConverterService.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public interface CurrencyConverterService extends javax.xml.rpc.Service {

public java.lang.String getconverterAddress();

public converter.CurrencyConverter getconverter() throws javax.xml.rpc.ServiceException;

public converter.CurrencyConverter getconverter(java.net.URL portAddress) throws javax.xml.rpc.ServiceException;

}

### CurrencyConverterServiceLocator

/\*\*

\* CurrencyConverterServiceLocator.java

\*

\* This file was auto-generated from WSDL

\* by the Apache Axis WSDL2Java emitter.

\*/

package converter;

public class CurrencyConverterServiceLocator extends org.apache.axis.client.Service implements converter.CurrencyConverterService {

// Use to get a proxy class for converter

private final java.lang.String converter\_address = "http://localhost:8080/axis/services/converter";

public java.lang.String getconverterAddress() {

return converter\_address;

}

// The WSDD service name defaults to the port name.

private java.lang.String converterWSDDServiceName = "converter";

public java.lang.String getconverterWSDDServiceName() {

return converterWSDDServiceName;

}

public void setconverterWSDDServiceName(java.lang.String name) {

converterWSDDServiceName = name;

}

public converter.CurrencyConverter getconverter() throws javax.xml.rpc.ServiceException {

java.net.URL endpoint;

try {

endpoint = new java.net.URL(converter\_address);

}

catch (java.net.MalformedURLException e) {

return null; // unlikely as URL was validated in WSDL2Java

}

return getconverter(endpoint);

}

public converter.CurrencyConverter getconverter(java.net.URL portAddress) throws javax.xml.rpc.ServiceException {

try {

converter.ConverterSoapBindingStub \_stub = new converter.ConverterSoapBindingStub(portAddress, this);

\_stub.setPortName(getconverterWSDDServiceName());

return \_stub;

}

catch (org.apache.axis.AxisFault e) {

return null;

}

}

/\*\*

\* For the given interface, get the stub implementation.

\* If this service has no port for the given interface,

\* then ServiceException is thrown.

\*/

public java.rmi.Remote getPort(Class serviceEndpointInterface) throws javax.xml.rpc.ServiceException {

try {

if (converter.CurrencyConverter.class.isAssignableFrom(serviceEndpointInterface)) {

converter.ConverterSoapBindingStub \_stub = new converter.ConverterSoapBindingStub(new java.net.URL(converter\_address), this);

\_stub.setPortName(getconverterWSDDServiceName());

return \_stub;

}

}

catch (java.lang.Throwable t) {

throw new javax.xml.rpc.ServiceException(t);

}

throw new javax.xml.rpc.ServiceException("There is no stub implementation for the interface: " + (serviceEndpointInterface == null ? "null" : serviceEndpointInterface.getName()));

}

/\*\*

\* For the given interface, get the stub implementation.

\* If this service has no port for the given interface,

\* then ServiceException is thrown.

\*/

public java.rmi.Remote getPort(javax.xml.namespace.QName portName, Class serviceEndpointInterface) throws javax.xml.rpc.ServiceException {

java.rmi.Remote \_stub = getPort(serviceEndpointInterface);

((org.apache.axis.client.Stub) \_stub).setPortName(portName);

return \_stub;

}

public javax.xml.namespace.QName getServiceName() {

return new javax.xml.namespace.QName("urn:converter", "CurrencyConverterService");

}

private java.util.HashSet ports = null;

public java.util.Iterator getPorts() {

if (ports == null) {

ports = new java.util.HashSet();

ports.add(new javax.xml.namespace.QName("converter"));

}

return ports.iterator();

}

}

### setclasspath.bat

@rem axis must be a subdirectory of MAIN\_DIR

set MAIN\_DIR=D:\jwsdp-1\_0\_01\webapps

set CLASSPATH=%CLASSPATH%;%MAIN\_DIR%\axis\WEB-INF\lib\axis.jar;%MAIN\_DIR%\axis\WEB-INF\lib\jaxrpc.jar;%MAIN\_DIR%\axis\WEB-INF\lib\commons-logging.jar;%MAIN\_DIR%\axis\WEB-INF\lib\wsdl4j.jar;%MAIN\_DIR%\axis\WEB-INF\lib\saaj.jar;%MAIN\_DIR%\axis;%MAIN\_DIR%\axis\WEB-INF\lib\commons-discovery.jar;<path to your class files>;

### undeploy.wsdd

<!-- Use this file to undeploy some handlers/chains and services -->

<!-- Two ways to do this: -->

<!-- java org.apache.axis.client.AdminClient undeploy.wsdd -->

<!-- after the axis server is running -->

<!-- or -->

<!-- java org.apache.axis.utils.Admin client|server undeploy.wsdd -->

<!-- from the same directory that the Axis engine runs -->

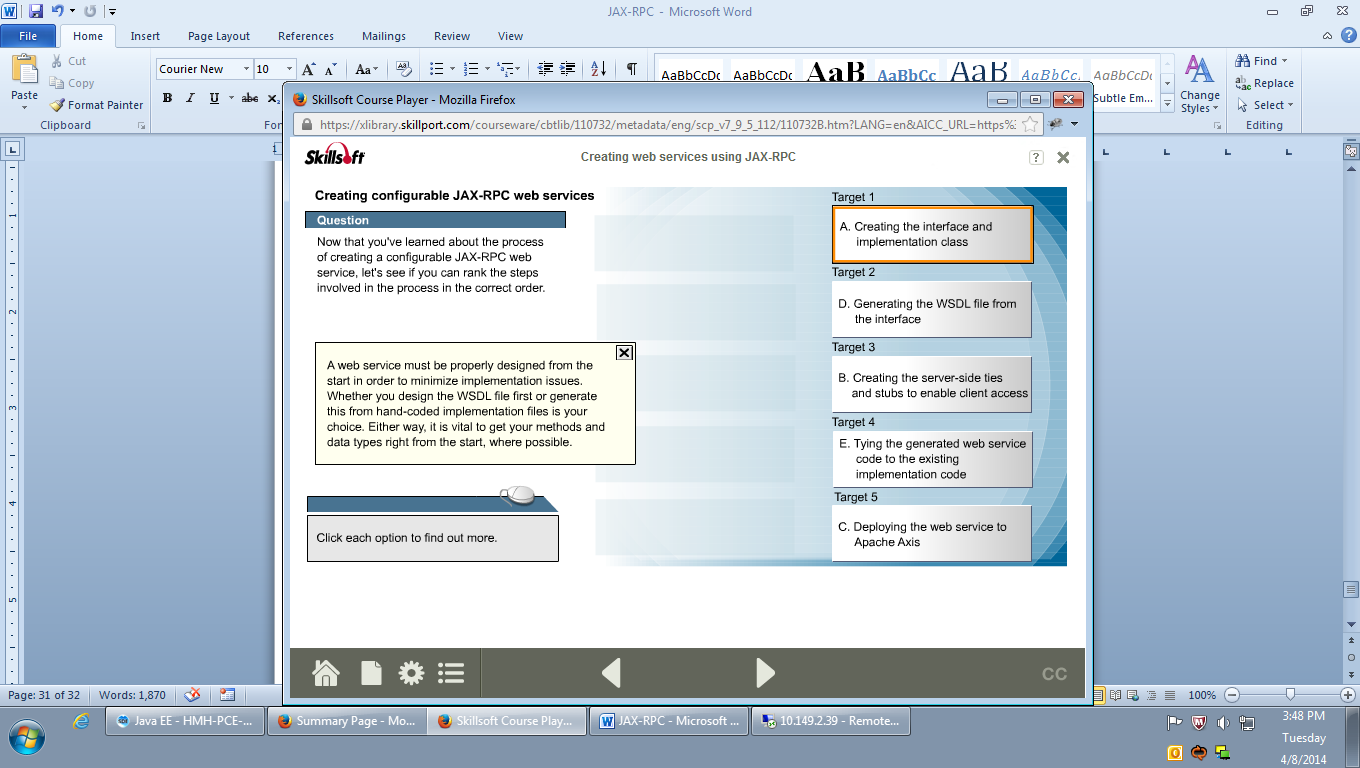
<undeployment

xmlns="http://xml.apache.org/axis/wsdd/">

<!-- Services from CurrencyConverterService WSDL service -->

<service name="converter"/>

</undeployment>



# The Java API for XML Messaging (JAXM)

## Abstract

This article discusses the components and functions of the Java API for XML Messaging (JAXM), examines the different types of JAXM clients, and looks at how JAXM works with other JAX APIs.

## What is JAXM?

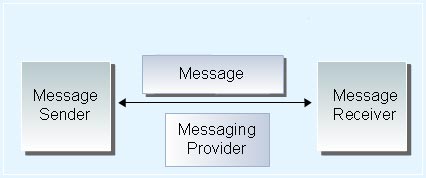
The Java API for XML Messaging (JAXM) is a standard that enables applications to send and receive business documents across the Internet using a pure Java API. Business-to-business (B2B) XML-based messaging can be implemented using JAXM as the data exchange mechanism. JAXM achieves this by providing a standard specification for all document-oriented XML message communication between two web service endpoints. JAXM is designed to exchange XML-based messages using SOAP (Simple Object Access Protocol). So JAXM is based on the SOAP and SOAP with Attachments specifications that define the format of a SOAP message. JAXM also aims to provide support for higher-level standards-based messaging protocols, such as ebXML.

Messaging is a component within the web services architecture that involves moving substantial quantities of data between interacting businesses. Because timely data delivery is often critical, B2B messaging must function smoothly and applications must interact seamlessly.

Typically, businesses use a messaging provider service to transport and route messages between endpoints. When JAXM is used in conjunction with a messaging provider it provides guaranteed message delivery and allows asynchronous – or one way – messaging along with the ability to route messages to multiple receivers.

## The components of JAXM

There are typically four components in any messaging system – the messaging provider, the message, the message sender, and the message receiver. Equally, the JAXM model typically contains four elements, with each handling a part of the messaging process.



The messaging system

The JAXM messaging provider carries the JAXM message between the message sender and receiver. Messaging providers exchange messages into two ways – asynchronously and synchronously, which means one-way or or request-response exchange, respectively. With an asynchronous exchange, the sender continues processing after transmitting the message and does not need to wait for a response. When the message is received, the receiver replies if necessary. With a synchronous exchange, the receiver must process the message and acknowledge the message before the sender can continue processing.

The messaging provider provides all the behind-the-scenes processing to transport and route messages. When a business uses a messaging provider, JAXM messages go through it and the messaging provider stores and tracks each message and assigns it a message identifier. If a message fails, the messaging provider can attempt to resend it.

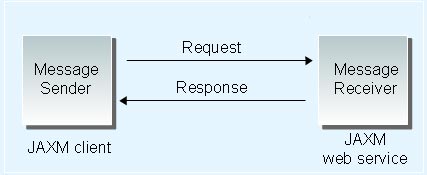
In the JAXM model, messages can be sent by a standalone JAXM client directly to the receiver, as well as by a JAXM client interacting with a messaging provider. The typical scenario is to use a messaging provider. In either case, the JAXM client converts the XML message to a JAXM message. JAXM messages must conform to the SOAP standard and the model provides default messages that conform to the SOAP standards and can be customized as necessary. A JAXM message may also contain attachments.

Messages are ultimately received by a JAXM web service, which processes the message and handles it in the appropriate way.

## JAXM clients

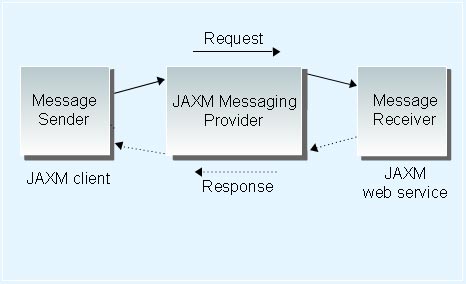
**There are two types of client that can send JAXM messages:**

* a standalone JAXM Client
* a JAXM Client that uses a messaging provider



**Standalone JAXM client**

A standalone JAXM Client connects to a web service and sends messages directly to its URL. It is typically implemented as a servlet. Standalone clients can only send JAXM messages, but they can't act as a message receiver as well. And without using a messaging provider, the JAXM client can only engage in synchronous messaging so the JAXM services they access must support this style of exchange.



**Client using a JAXM messaging provider**

Messaging providers give JAXM clients greater reliability and flexibility, allowing synchronous and asynchronous messaging. This type of client can also act as a message receiver. These clients should be deployed either in a J2EE web container where the SOAP is protocol bound to HTTP or in an Enterprise JavaBeans container.

Whichever container is used, it must be supplied with the name of your message provider when the message is being sent. This is achieved using the Java Naming and Directory Interface (JNDI). You must register the message provider with JNDI, allowing the client to establish a connection to it. Once the connection is established, the communications infrastructure provided by the messaging provider is entirely transparent to the client.

## JAXM and SAAJ

From what we have seen, JAXM contains two types of functionality – low-level functionality centered on the ability to exchange XML documents as SOAP messages, and high-level asynchronous messaging functionality and the ability to work with high-level emerging protocols such as ebXML. The low-level functionality is contained in the SOAP with Attachments API for Java (SAAJ) API, which was split from the previous version of the JAXM API to form a separate distinct package with the latest release of JAXM (1.1).

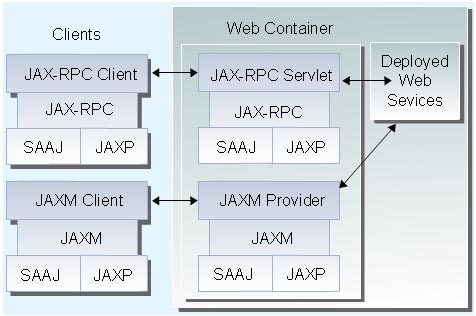
SAAJ allows developers to create messages that conform to the SOAP 1.1 and SOAP with Attachments specifications. So now basic XML messaging and attachment functionality resides in a separate package, which other JAX APIs, such as JAX-RPC, can now use without being dependent on JAXM.

## The difference between JAXM and JAX-RPC

The JAX-RPC API supports making XML-based RPC calls over SOAP. Unlike JAXM, which provides document-based messaging, JAX-RPC is an RPC-based messaging model that sends method calls encapsulated in XML.

JAX-RPC's functionality overlaps with JAXM as implementations of JAX-RPC such as Apache Axis can also handle document-based messaging. Although both models support SOAP messaging, JAX-RPC handles basic SOAP messaging behind the scenes, whereas with JAXM SOAP messages must be explicitly created. JAXM does provide asynchronous messaging which JAX-RPC does not. However, it is possible to send SOAP messages using the asynchronous messaging functionality of the existing Java Message Service (JMS) API instead of using HTTP. For these reasons some commentators have pointed out that JAXM's functionality is somewhat redundant and it hasn't been included in the latest release of the J2EE – J2EE 1.4.

## How the JAX APIs work together



The JAX APIs

All the JAX APIs use established SOAP-based standards and work together to support web services. As you can see in the diagram, both JAXM and JAX-RPC use JAXP and SAAJ. So both styles of communication – JAXM's document-oriented and JAX-RPC's RPC-based – use the SAAJ and JAXP APIs.

JAXM uses the SAAJ and JAXP APIs explicitly, as developers have to make calls using these APIs in their clients and web services. In contrast, JAX-RPC does this behind the scenes allowing the developers to make standard method calls.

## Summary

The Java API for XML Messaging (JAXM) is a standard that enables applications to send and receive business documents across the Internet using a pure Java API. The JAXM model contains four elements – the JAXM messaging provider, the JAXM client, the JAXM message, and the JAXM service. Clients can send JAXM messages directly to a JAXM web service endpoint or they can route them through a messaging provider if more sophisticated messaging is required.

The SOAP with Attachments API for Java (SAAJ) was once part of JAXM but has since been removed to enable web service developers to use the SAAJ package without relying on JAXM. JAXM resembles JAX-RPC in that both provide web service communication mechanisms. JAXM is document-oriented, whereas JAX-RPC is an RPC-based messaging model that sends method calls encapsulated in XML. All the JAX APIs use established SOAP-based standards and work together to support web services.

# The Java API for XML Registries (JAXR)

## Abstract

This article describes how the Java API for XML Registries (JAXR) provides a standard mechanism for accessing registry services from the Java language, outlines its components, and describes how you use it to register a business and search a registry.

## What is a registry?

A business registry is a third-party structure containing information about a business that enables other businesses to discover and access it. It acts like an electronic version of the Yellow Pages. In terms of web services, registries are implemented in XML and facilitate the registration, deployment, and discovery of web services. XML business registries are based on agreed standards, allowing businesses to build business-to-business (B2B) communications over the Internet. By their nature, XML registries facilitate dynamic and loosely coupled interactions between businesses.

Businesses seeking a B2B partner can register with a registry, so that prospective partners can easily establish communication channels. Businesses can also submit shared resources to a registry or search the registry for potential partners.

There are 2 main types of XML registries:

* the ebXML Registry and Repository standard
* the Universal Description, Discovery, and Integration (UDDI) standard.

ebXML specifies an open standards registry that is sponsored by the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport (UN/CEFACT).

The UDDI project is backed by an industry consortium and allows businesses to register under a variety of categories. It specifies a searchable database, which can be queried by businesses seeking partners in a specific category.

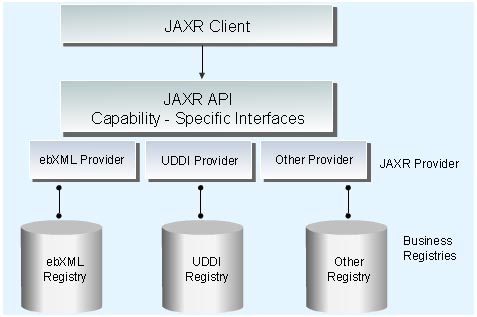
## JAXR and registries

The Java API for XML Registries (JAXR) is a standard mechanism for accessing XML-based business registries from the Java language.

JAXR allows web service endpoints to create entries in public registries to advertise their services, to store shared information in the registries, or to create private registries. And it allows web service clients to search registries for businesses.

JAXR provides developers with a user-friendly abstraction API, which allows them to work with any standards-based XML registry. The two types of registry currently supported are ebXML registries and UDDI registries. The JAXR information model defines how content and metadata is structured within XML registries. This provides developers with a basis for creating client programs that are portable across different registries.

JAXR also supports pluggable JAXR providers, so developers can build applications without being restricted to a specific vendor.



A JAXR client communicates with XML-based business registries via the JAXR API

## JAXR Goals

The Java Specification Requests (JSR) 93 group has developed JAXR to fulfill several stated objectives, including support for

* industry-standard XML registry functionality
* registration of member organizations and enterprises
* submission and storing of arbitrary registry content
* lifecycle management of XML and non-XML registry content
* user-defined associations between registry content
* user-defined multilevel classification of registry content along multiple user defined facets
* registry content querying based on defined classification schemes
* registry content querying based on complex adhoc queries
* registry content querying based on keyword based search
* sharing of web services
* sharing of business process between partners
* sharing of schemas between partners
* sharing of business documents between partners
* trading partner agreement assembly and negotiation
* schema assembly
* heterogeneous distributed registries
* enabling publish/subscribe XML messaging between parties

## How JAXR works

The JAXR architecture consists of

* JAXR clients
* JAXR providers

A JAXR client is any application that uses the JAXR API to communicate with an XML business registry.

A JAXR provider is an interface to an XML business registry that has been created using the JAXR API. A JAXR provider may give access to a single registry or a class of registry providers. Clients access registries through JAXR providers.

A JAXR provider is created by implementing two packages:

* javax.xml.registry.infomodel
* javax.xml.registry

The javax.xml.registry.infomodel package contains the RegistryObject interface, which defines the information schema for the particular registry and how the registry objects relate to one another. The javax.xml.registry package contains the RegistryService and Connection interfaces, which define access to the particular registry. These interfaces specify how client connections are created.

The javax.xml.registry package also contains the BusinessQueryManager interface – which allows the client application to search the registry for data – and the BusinessLifeCycleManager interface – which allows the client application to edit registry information.

## Registering a business using JAXR

To register with a business registry, a business can use JAXR to supply its name, business description, and business classification. In this example, a company called Imagenie Online Music Store uses JAXR to register its name, description, and North American Industry Classification System (NAICS) category in a business registry.

RegistryService regService =   
        connection.getRegistryService();  
  
BusinessLifeCycleManager blcMan =  
         regService.getBusinessLifeCycleManager();  
  
BusinessQueryManager bqMan =   
        regService.getBusinessQueryManager();  
  
Organization org = blcMan.createOrganization  
            ("Imagenie Online Music Store");  
  
InternationalString desc=blcMan.createInternationalString  
        ("An entertainment and music education company");  
        org.setDescription(desc);  
  
ClassificationScheme classScheme = bqMan.  
    findClassificationSchemeByName(null, "ntis-gov:naics");  
  
Classification clas =   
    (Classification)blcMan.createClassification(classScheme,  
        "Prerecorded Tape, Compact Disc, and Record Stores",  
        "451220");  
  
Collection classifications = new ArrayList();  
classifications.add(clas);  
  
org.addClassifications(classifications);  
  
Collection orgs = new ArrayList();  
orgs.add(org);  
  
blcMan.saveOrganizations(orgs);

First, the RegistryService object is created, followed by the BusinessLifeCycleManager object, and then the BusinessQueryManager object. Imagenie's name and description and classification are added to the Organization object, which is then added to a Collection object. This Collection object is then saved by the BusinessLifeCycleManager.

## Searching a business registry using JAXR

To search a business registry, a business can use JAXR to specify its search criteria.

The search functionality is provided by JAXR's BusinessQueryManager interface. In this example, a business is using a BusinessQueryManager object, to search for online music businesses.

BusinessQueryManager bqMan =   
            regService.getBusinessQueryManager();  
  
ClassificationScheme classScheme = bqMan.  
    findClassificationSchemeByName(null, "ntis-gov:naics");  
  
Classification clas =   
    (Classification)blcMan.createClassification(classScheme,  
        "Prerecorded Tape, Compact Disc, and Record Stores",  
        "451220");  
  
Collection classifications = new ArrayList();  
classifications.add(clas);  
  
Collection qualifiers = new ArrayList();  
qualifiers.add(FindQualifier.CASE\_SENSITIVE\_MATCH);  
  
Collection patterns = new ArrayList();  
patterns.add("%Music%");  
patterns.add("%Online%");  
  
BulkResponse bulkResp = bqMan.findOrganizations  
    (qualifiers, patterns, null, classifications, null,   
    null);  
          
Collection orgs = bulkResp.getCollection();

The BusinessQueryManager object invokes the method findOrganizations, to which it can pass six possible search parameters. In this example, three search criteria that specify the classification of the business and that its name should contain the words 'Music' and 'Online' are passed to the function. No criteria are specified for the other three parameters.

## Summary

An XML registry is a third-party structure that contains electronic information about businesses seeking to establish business-to-business (B2B) web service communications. There are two main standards for XML business registries – the ebXML Registry and Repository standard and the Universal Description, Discovery, and Integration (UDDI) project. The Java API for XML Registries (JAXR) is a standard mechanism that allows Java developers to build client applications that can access both of these types of business registries.

The JAXR architecture consists of JAXR clients – applications that use the JAXR API to communicate with an XML business registry – and JAXR providers, which provide the interface for this communication. Using the packages contained in the JAXR API, a business can submit its details to a registry or it can search a registry by specifying a series of search criteria.