IBM Software
WebSphere

An IBM Proof of Technology

JAXWS Web Service

Contents

Lab 7 JAXWS Web Service

In this lab we will learn how to use JAXWS web service via a sample application demonstrating seven different patterns:

1. A web service stub client
2. A web service dynamic client
3. A POJO based endpoint
4. Handler chain
5. Web service message context
6. Catalog support
7. MTOM support

Please refer to the following table for file and resource location references on different operating systems.

Location Ref.	OS	Absolute Path
{LAB_HOME}	Windows	`C:\WLP_<version>`
	Linux	`~/WLP_<version>`
	Mac OSX	

7.1 Prerequisites

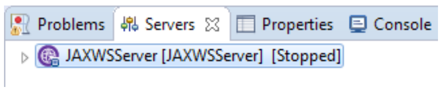
The following preparation must be completed prior to beginning this lab:

1. Complete the Getting Started lab to set up the lab environment, and learn how to create a server using Eclipse with WebSphere Developer Tools (WDT).
2. Optional: complete the Simple Development lab if you need a refresher on how to use Eclipse and WDT.

7.2 Create a new server

__1. Start Eclipse.

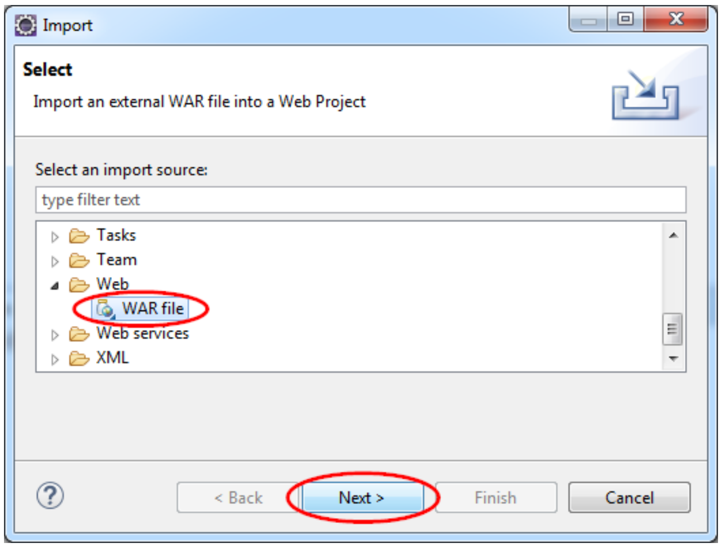
__2. Using the procedure learned in the Getting Started lab, create a new Liberty profile application server called **JAXWSServer**.



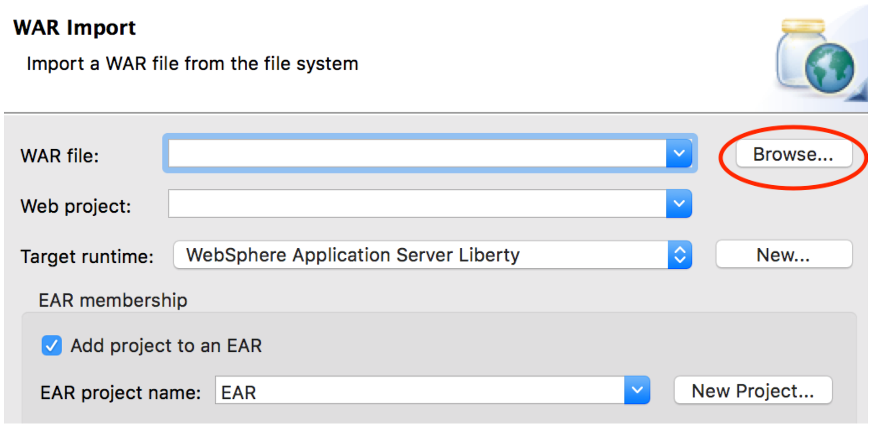
JAXWSServer [JAXWSServer] [Stopped]

7.3 Import the JAXWS sample application

__3. Click **File > Import** from the Eclipse menu. Expand **Web** and select **WAR file** on the Import window, then click **Next**.



__4. Click **Browse** on the WAR Import view.

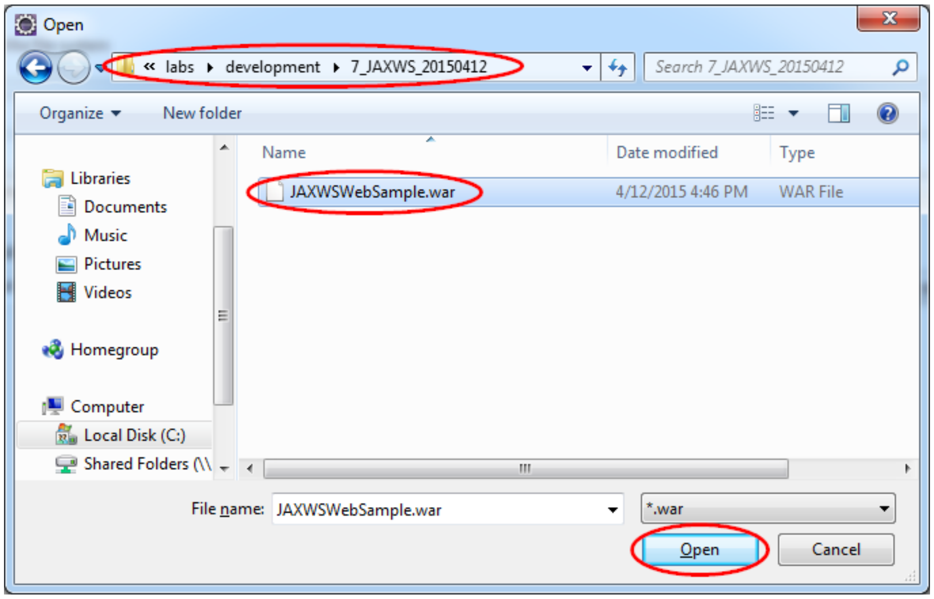


Target runtime: WebSphere Application Server Liberty

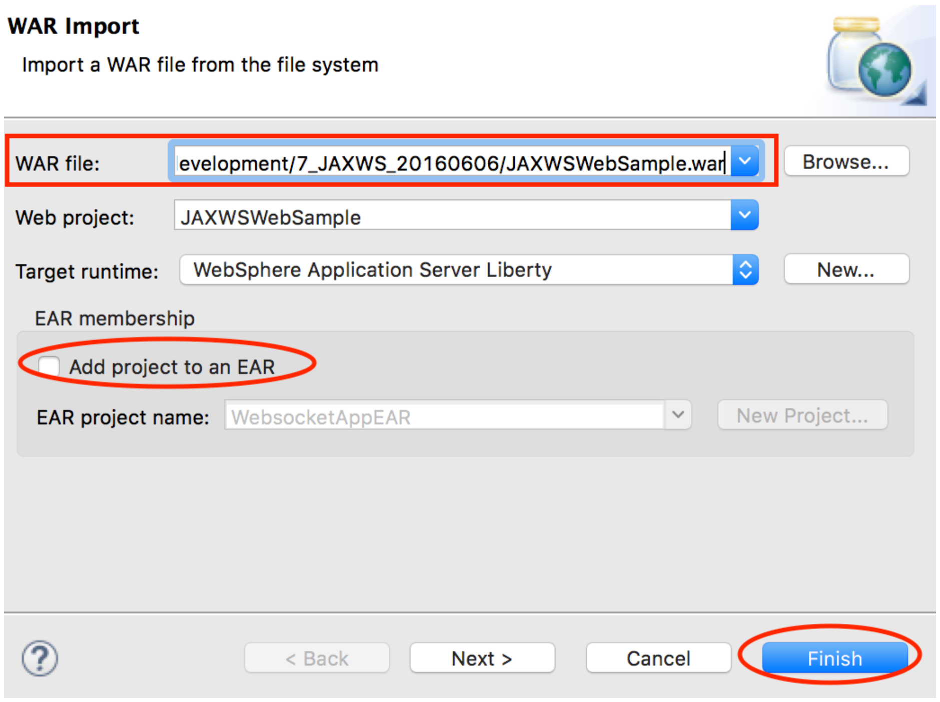
Add project to an EAR: ✓

EAR project name: EAR

__5. Use the Open dialog to locate the `{LAB_HOME}\labs\development\7_JAXWS_<timestamp>\ JAXWSWebSample.war` file and click **Open**.



__6. Clear the **Add project to an EAR** checkbox, and verify the **Target runtime** is set to **WebSphere Application Server Liberty**, and click **Finish**.


WAR Import

Import a WAR file from the file system

WAR file: evelopment/7_JAXWS_20160606/JAXWSWebSample.war

Web project: JAXWSWebSample

Target runtime: WebSphere Application Server Liberty

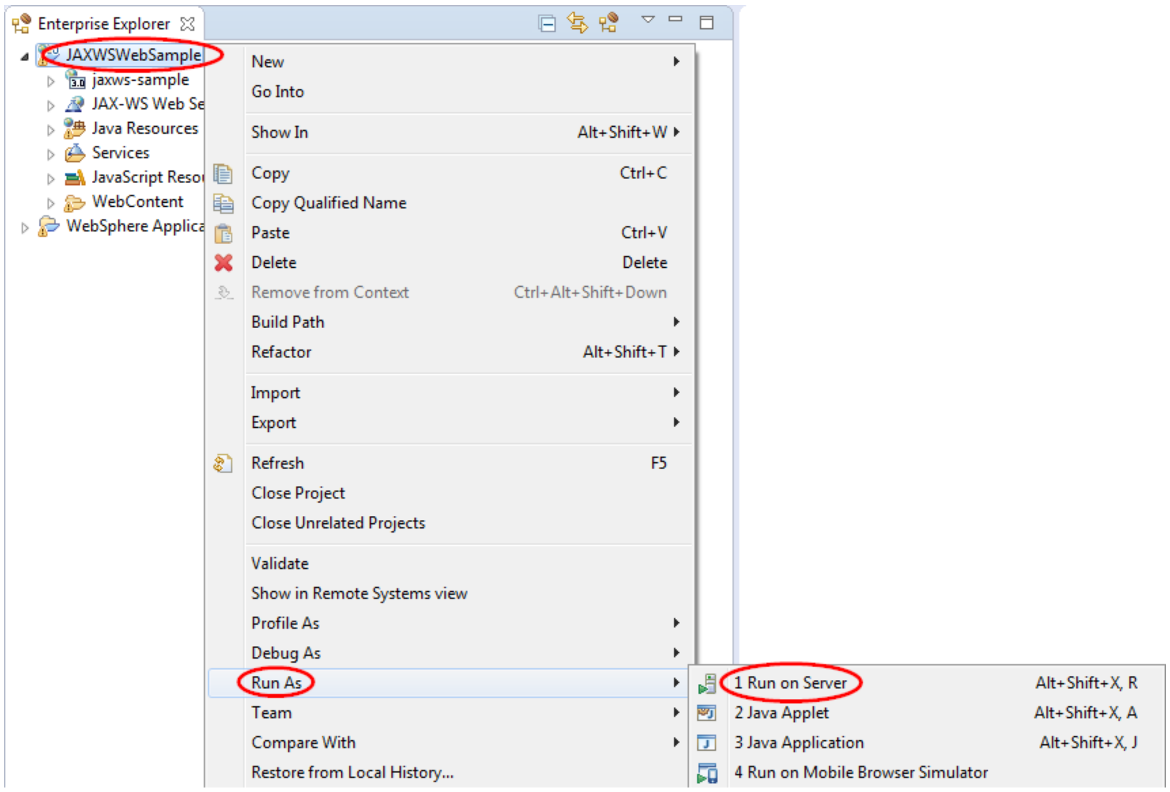
EAR membership

☐ Add project to an EAR

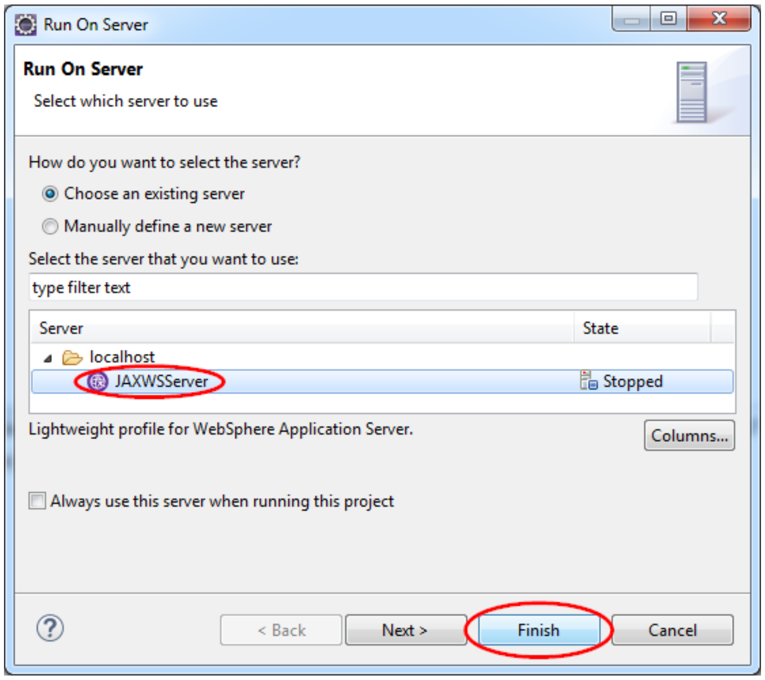
EAR project name: WebsocketAppEAR

7.4 Run the sample application

__1. In the Enterprise Explorer view, right click the **JAXWSWebSample** project, then click **Run As > Run on server**.



__2. On the Run On Server window, select **JAXWSServer** from the **Server** list, then click **Finish**.


Run On Server

Run On Server

Select which server to use

How do you want to select the server?

(●) Choose an existing server

( ) Manually define a new server

Select the server that you want to use:

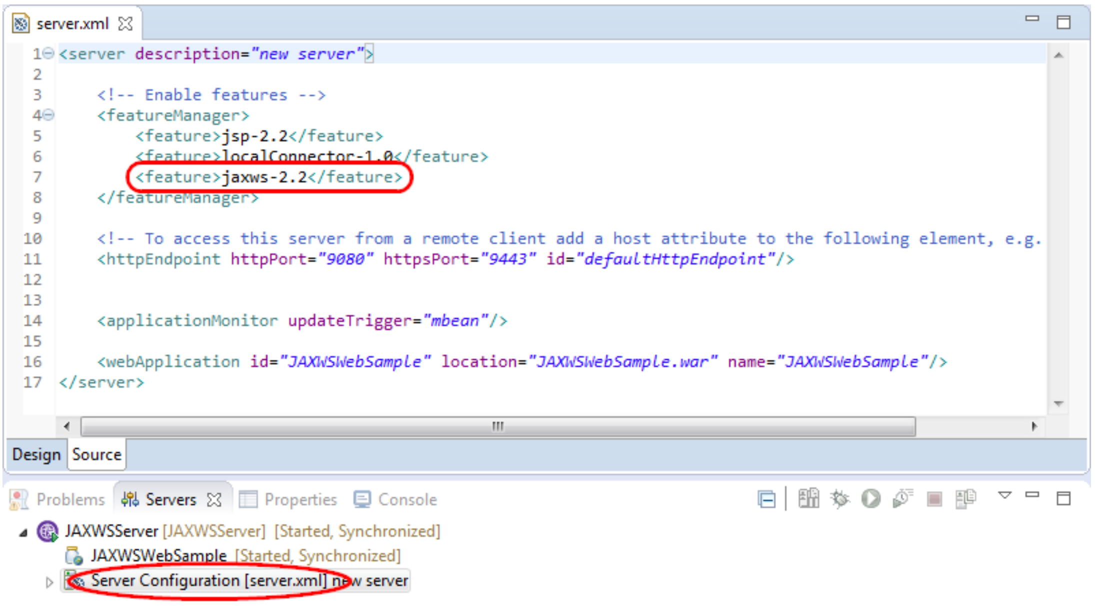
type filter text

Server	State
localhost	
JAXWSServer	Stopped

Lightweight profile for WebSphere Application Server.

[ ] Always use this server when running this project

__3. Expand **JAXWSServer** in the Server view and double click **Server Configuration**. Inspect `server.xml` and ensure the `jaxws-2.2` feature has been added automatically by WDT.

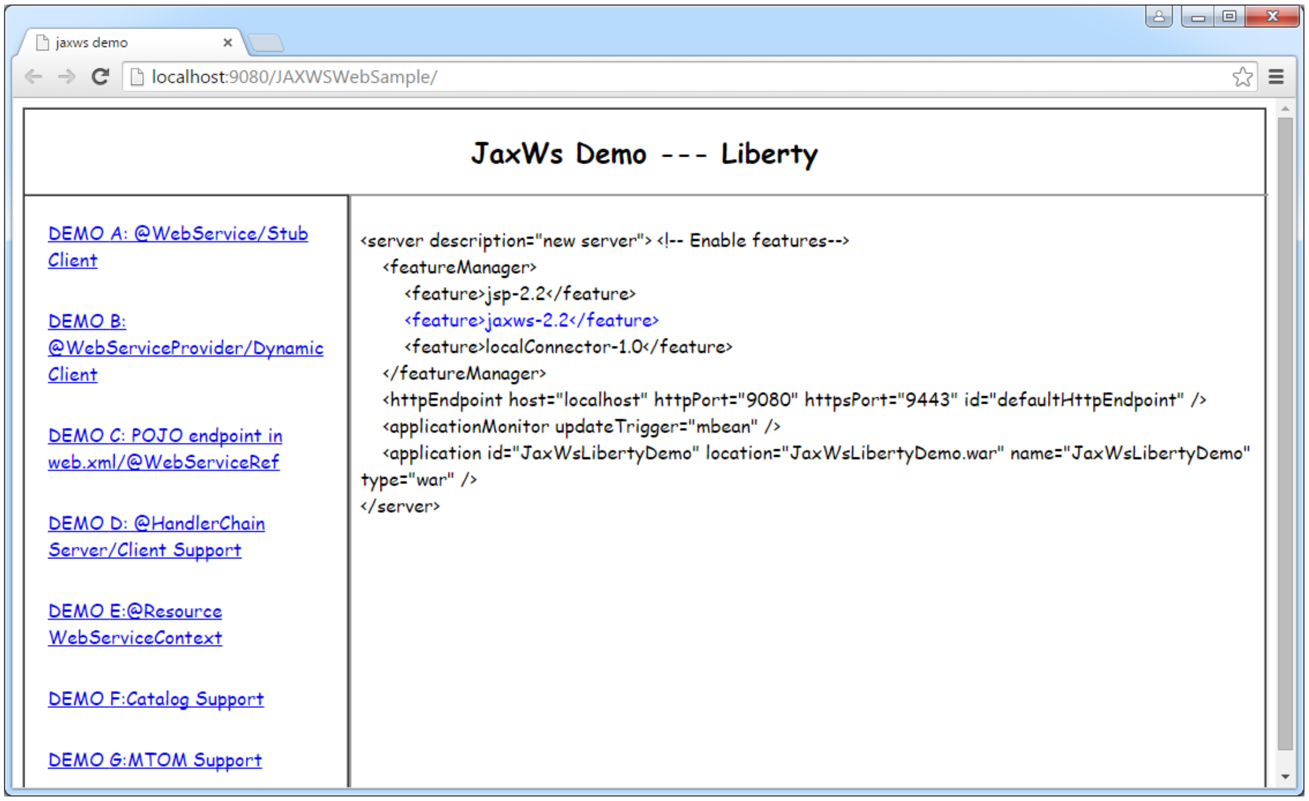


```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>localConnector-1.0</feature>
        <feature>jaxws-2.2</feature>
    </featureManager>

    <!-- To access this server from a remote client add a host attribute to the following element, e.g.
    <httpEndpoint httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint"/>


    <applicationMonitor updateTrigger="mbean"/>

    <webApplication id="JAXWSWebSample" location="JAXWSWebSample.war" name="JAXWSWebSample"/>
</server>
```

```
JAXWSServer [JAXWSServer]  [Started, Synchronized]
    JAXWSWebSample  [Started, Synchronized]
  Server Configuration [server.xml] new server
```

__4. After the server starts running, Eclipse may automatically bring up a browser for you to interact with the application. You may also start your browser and go to the URL location http://localhost:9080/JAXWSWebSample/.

__5. Interact with each of the seven scenarios per instructions provided on the web page.



JaxWs Demo --- Liberty

DEMO A: @WebService/Stub Client

DEMO B: @WebServiceProvider/Dynamic Client

DEMO C: POJO endpoint in web.xml/@WebServiceRef

DEMO D: @HandlerChain Server/Client Support

DEMO E:@Resource WebServiceContext

DEMO F:Catalog Support

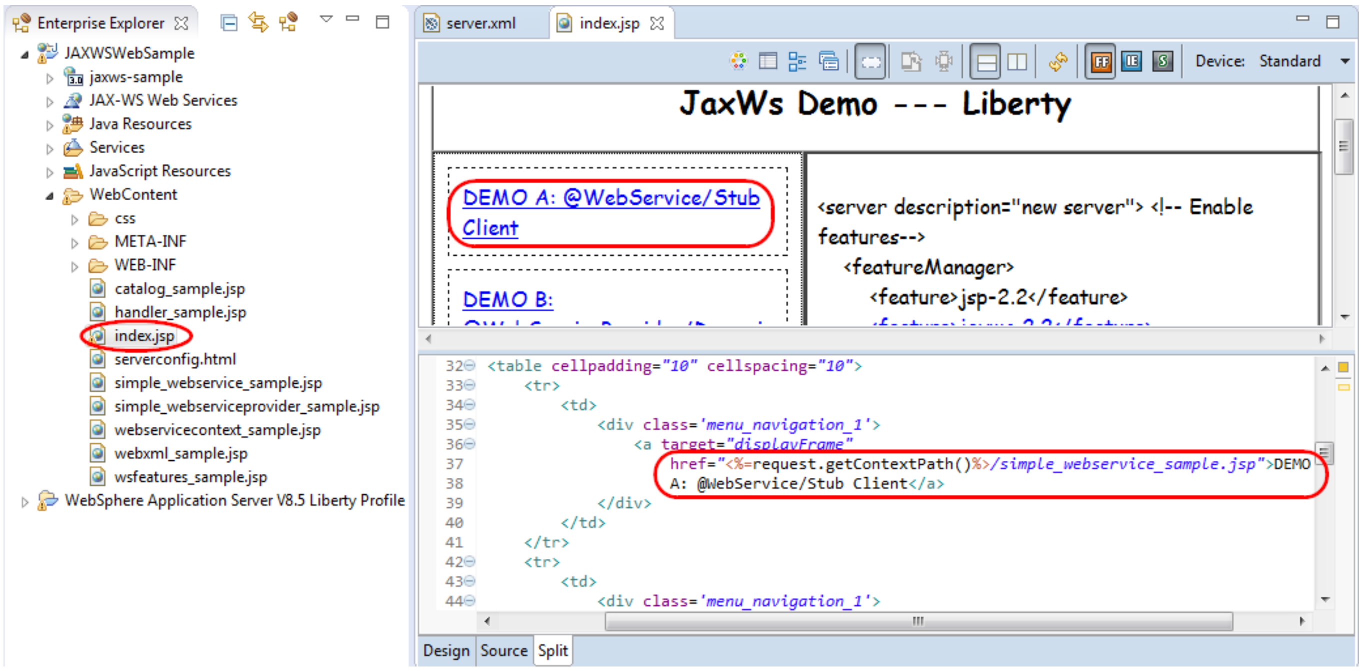
DEMO G:MTOM Support

```
<server description="new server"> <!-- Enable features-->
   <featureManager>
      <feature>jsp-2.2</feature>
      <feature>jaxws-2.2</feature>
      <feature>localConnector-1.0</feature>
   </featureManager>
   <httpEndpoint host="localhost" httpPort="9080" httpsPort="9443" id="defaultHttpEndpoint" />
   <applicationMonitor updateTrigger="mbean" />
   <application id="JaxWsLibertyDemo" location="JaxWsLibertyDemo.war" name="JaxWsLibertyDemo"
type="war" />
</server>
```

7.5 Code walk-through

7.5.1 `index.jsp`

The `index.jsp` is the main entry point to the sample application. In the Enterprise Explorer view, navigate to **JAXWSWebSample > WebContent** and double click on `index.jsp`. This displays the contents of `index.jsp` in Eclipse. Note that the code creates a table for the seven scenarios, with a separate link to each. The highlight below shows the link to the first scenario.



```
<server description="new server"> <!-- Enable
features-->
    <featureManager>
        <feature>jsp-2.2</feature>
```

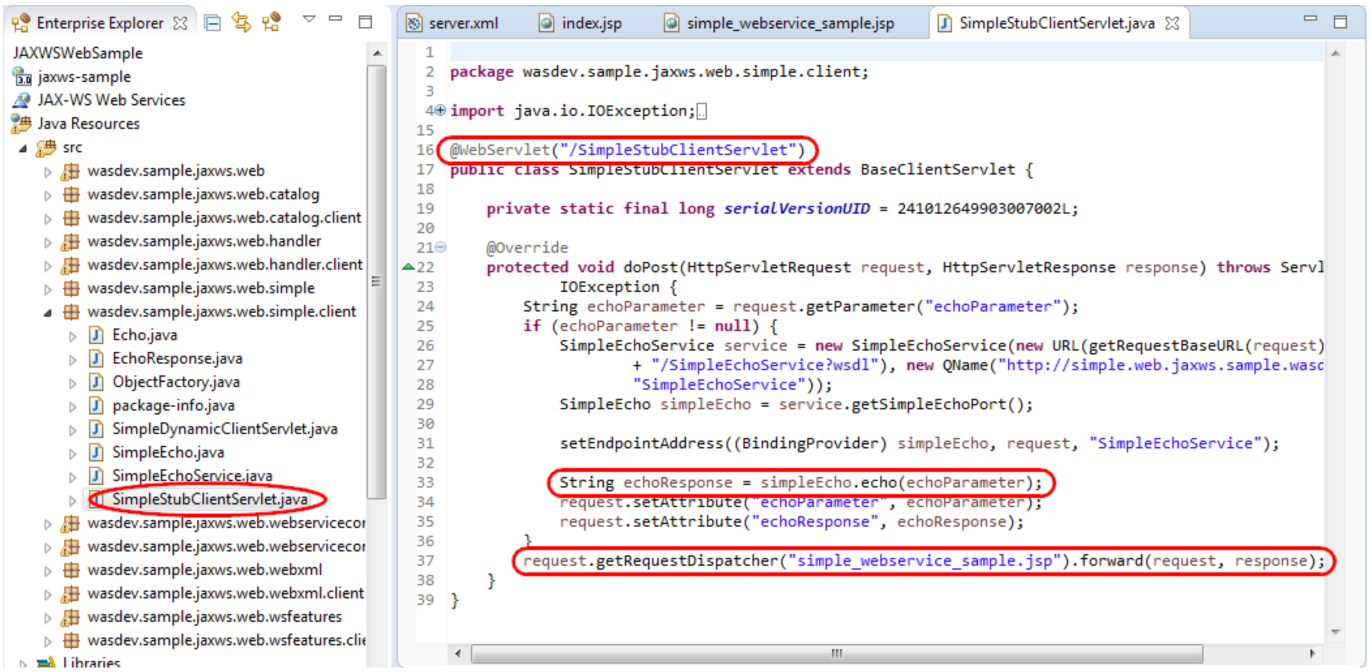
```
<table cellpadding="10" cellspacing="10">
    <tr>
        <td>
            <div class='menu_navigation_1'>
                <a target="displayFrame"
                    href="<%=request.getContextPath()%>/simple_webservice_sample.jsp">DEMO
                    A: @WebService/Stub Client</a>
            </div>
        </td>
    </tr>
    <tr>
        <td>
            <div class='menu_navigation_1'>
```

7.5.2 WebService stub client

Open `simple_webservice_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `SimpleStubClientServlet`:

```
<form action="<%=request.getContextPath()%>/SimpleStubClientServlet"
    target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.simple.client.SimpleStubClientServlet.java` and examine its contents. Note how it respond to the servlet path `/SimpleStubClientServlet`, invokes the web service via `SimpleEcho` stub, and forwards the results back to `simple_websiervice_sample.jsp`.


Enterprise Explorer
JAXWSWebSample
jaxws-sample
JAX-WS Web Services
Java Resources
src
wasdev.sample.jaxws.web
wasdev.sample.jaxws.web.catalog
wasdev.sample.jaxws.web.catalog.client
wasdev.sample.jaxws.web.handler
wasdev.sample.jaxws.web.handler.client
wasdev.sample.jaxws.web.simple
wasdev.sample.jaxws.web.simple.client
Echo.java
EchoResponse.java
ObjectFactory.java
package-info.java
SimpleDynamicClientServlet.java
SimpleEcho.java
SimpleEchoService.java
SimpleStubClientServlet.java
wasdev.sample.jaxws.web.webservicecor
wasdev.sample.jaxws.web.webservicecor
wasdev.sample.jaxws.web.webxml
wasdev.sample.jaxws.web.webxml.client
wasdev.sample.jaxws.web.wsfeatures
wasdev.sample.jaxws.web.wsfeatures.clie
Libraries
server.xml
index.jsp
simple_webservice_sample.jsp
SimpleStubClientServlet.java

```

package wasdev.sample.jaxws.web.simple.client;

import java.io.IOException;

@WebServlet("/SimpleStubClientServlet")
public class SimpleStubClientServlet extends BaseClientServlet {

    private static final long serialVersionUID = 241012649903007002L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws Servl
            IOException {
        String echoParameter = request.getParameter("echoParameter");
        if (echoParameter != null) {
            SimpleEchoService service = new SimpleEchoService(new URL(getRequestBaseURL(request)
                    + "/SimpleEchoService?wsdl"), new QName("http://simple.web.jaxws.sample.wasc
                    "SimpleEchoService"));
            SimpleEcho simpleEcho = service.getSimpleEchoPort();

            setEndpointAddress((BindingProvider) simpleEcho, request, "SimpleEchoService");

            String echoResponse = simpleEcho.echo(echoParameter);
            request.setAttribute("echoParameter", echoParameter);
            request.setAttribute("echoResponse", echoResponse);
        }
        request.getRequestDispatcher("simple_webservice_sample.jsp").forward(request, response);
    }
}
```

Examine the file `wasdev.sample.jaxws.web.simple.client.SimpleEcho.java`. Note that it defines the client web service interface, with the `echo` method. This is used by the `SimpleStubClientServlet` to make a web service call.

Examine the file `wasdev.sample.jaxws.web.simple.SimpleEcho.java` and note the web service implementation that merely echoes its input.

```
package wasdev.sample.jaxws.web.simple;

import javax.jws.WebService;

@WebService
public class SimpleEcho {

    public String echo(String value) {
        return "Echo Response [" + value + "]";
    }
}
```

7.5.3 WebService dynamic client

Open `simple_webserviceprovider_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `SimpleDynamicClientServlet`:

```
<form action="<%=request.getContextPath()%>/SimpleDynamicClient"
   target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.simple.client.SimpleDynamicClientServlet.java`, and note how it uses a `MessageFactory` to dynamically create a message, and the `Service` class to create a web service client reference. No actual Java client interface is required.

Examine the file `wasdev.sample.jaxws.web.simple.SimpleEchoProvider.java`, which is the server side implementation of the service. Note how it uses the `MessageFactory` to retrieve the message and process the message. No actual Java interface is needed.

7.5.4 POJO based endpoint

Open `webxml_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `/simpleHelloWorldWebXmLClientServlet`:

```
<form
   action="<%=request.getContextPath()%>/SimpleHelloWorldWebXmlClientServlet"
   target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.webxml.client.SimpleHelloWorldWebXmlClientServlet`, and note how it demonstrates using four different ways to get a web service client:

1. Declaring a variable of the type `Service` with the `@WebServiceRef` annotation that maps it to the `SimpleHelloWorldWebXmlService` class.

   ```
   @WebServiceRef(value = SimpleHelloWorldWebXmlService.class)
   private Service service;
   ```

2. Declaring a variable of the type `SimpleHelloWorldWebXmlService` with the `@WebServiceRef` annotation.

   ```
   @WebServiceRef
   private SimpleHelloWorldWebXmlService simpleHelloWorldWebXmlService;
   ```

3. Declaring a variable of the web service interface type `SimpleHelloWorldWebXml`, with the `@WebServiceRef` annotation that maps it to the `SimpleHelloWorldWebXmlService` class.

   ```
   @WebServiceRef(value = SimpleHelloWorldWebXmlService.class)
   private SimpleHelloWorldWebXml simpleHelloWorldWebXml;
   ```

4. Declaring a variable of the type `SimpleHelloWorldWebXmlService` with the `@Resource` annotation.

    ```
    @Resource
    private SimpleHelloWorldWebXmlService simpleHelloWorldWebXmlService2;
    ```

Open `wasdev.sample.jaxws.web.webxml.client.SimpleHelloWorldWebXmlService`, the implementation of the client service. Note the `@WebServiceClient` annotation:

```
@WebServiceClient(name = "SimpleHelloWorldWebXmlService", targetNamespace =
"http://webxml.web.jaxws.sample.wasdev/", wsdlLocation = "WEB-
INF/wsdl/SimpleHelloWorldWebXmlService.wsdl")
```

Note that the web service client is connecting to the path `/CustomizedHelloworld`:

```
url = new URL(baseUrl, "http://localhost:9080/JaxWsLibertyDemo/CustomizedHelloWorld?wsdl");
```

Open `web.xml` and note that `/CustomizedHelloWorld` is mapped to a servlet `SimpleHelloWorld`:

```
<servlet>
   <display-name>SimpleHelloWorld</display-name>
   <servlet-name>SimpleHelloWorld</servlet-name>
   <servlet-class>wasdev.sample.jaxws.web.webxml.SimpleHelloWorldWebXml</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>SimpleHelloWorld</servlet-name>
   <url-pattern>/CustomizedHelloWorld</url-pattern>
</servlet-mapping>
```

Open `wasdev.sample.jaxws.web.webxml.SimpleHelloWorldWebXml.java`, and note that it is just a POJO that implements the web service.

7.5.5 Handler Chain

Open `handler_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `/HandlerClientServlet`:

```
<form action="<%=request.getContextPath()%>/HandlerClientServlet"
   target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.handler.client.HandlerClientServlet`, and note that it declares and client handler chain, and uses a client service `RouteTrackerService`:

```
@HandlerChain(file = "handler-client.xml")

….
@WebServiceRef(value = RouteTrackerService.class)
private RouteTracker routeTracker;
```

Open `wasdev.sample.jaxws.web.handler.client.handler-client.xml`, and note that it declares two handlers:

```
<handler-chain>
   <handler>
      <handler-name>LogicalHandler</handler-name>
      <handler-class>wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler
      </handler-class>
      <init-param>
         <param-name>arg0</param-name>
         <param-value>testInitParam</param-value>
      </init-param>
   </handler>
   <handler>
      <handler-name>SOAPHandler</handler-name>
      <handler-class>wasdev.sample.jaxws.web.handler.client.TestClientSOAPHandler
      </handler-class>
   </handler>
</handler-chain>
```

Open `wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler` and note that it contains a minimal implementation of a `LogicalHandler`.

Open `wasdev.sample.jaxws.web.handler.client.TestClienSOAPHandler` and note that it contains a minimal implementation of a `SOAPHandler`.

Open `wasdev.sample.jaxws.web.handler.client.RouteTrackerService` and note that points to the `RouteTrackerService` server side implementation:

```
url = new URL(baseUrl,
"http://localhost:9080/JaxWsLibertyDemo/RouteTrackerService?wsdl");
```

Open `wasdev.sample.jaxws.web.handler.RouteTracker` and note it implements the `RouteTracker` service, and uses handler chain defined in `handler-test.xml`:

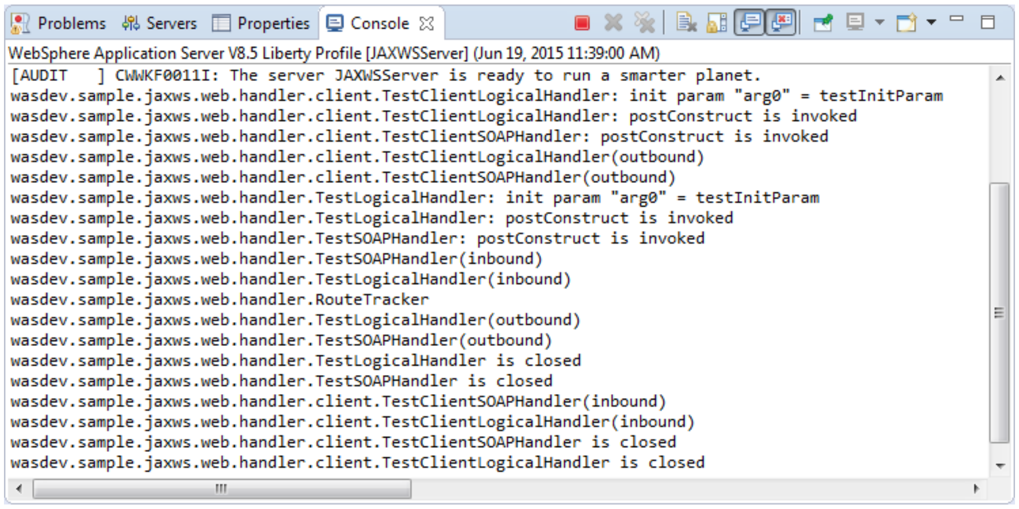
```
@WebService(name = "RouteTracker", serviceName = "RouteTrackerService", portName =
   "RouteTrackerPort", targetNamespace = "http://web.jaxws.sample.wasdev/")
@HandlerChain(file = "handler-test.xml")
public class RouteTracker {

   @SuppressWarnings({ "rawtypes", "unchecked" })
   public String track(@WebParam(name = "message") String message) {

      System.out.println(getClass().getName());
      return "response [" + message + "] Please check the outputs on the console";
   }
}
```

Bring up `wasdev.sample.jaxws.web.handler.handler-test.xml` and note it uses two handlers:

```
<handler-chain>
   <handler>
      <handler-name>LogicalHandler</handler-name>
      <handler-class>wasdev.sample.jaxws.web.handler.TestLogicalHandler
      </handler-class>
      <init-param>
         <param-name>arg0</param-name>
         <param-value>testInitParam</param-value>
      </init-param>
   </handler>
   <handler>
      <handler-name>SOAPHandler</handler-name>
      <handler-class>wasdev.sample.jaxws.web.handler.TestSOAPHandler
      </handler-class>
   </handler>
</handler-chain>
```

The implementation of the server side handler is similar to client side handler. Therefore, we will not discuss them here.

After you submit the query for this scenario in the browser, check the Console view in Eclipse to ensure that the handlers are called. It'll look like this:



```
WebSphere Application Server V8.5 Liberty Profile [JAXWSServer] (Jun 19, 2015 11:39:00 AM)
[AUDIT   ] CWWKF0011I: The server JAXWSServer is ready to run a smarter planet.
wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler: init param "arg0" = testInitParam
wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler: postConstruct is invoked
wasdev.sample.jaxws.web.handler.client.TestClientSOAPHandler: postConstruct is invoked
wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler(outbound)
wasdev.sample.jaxws.web.handler.client.TestClientSOAPHandler(outbound)
wasdev.sample.jaxws.web.handler.TestLogicalHandler: init param "arg0" = testInitParam
wasdev.sample.jaxws.web.handler.TestLogicalHandler: postConstruct is invoked
wasdev.sample.jaxws.web.handler.TestSOAPHandler: postConstruct is invoked
wasdev.sample.jaxws.web.handler.TestSOAPHandler(inbound)
wasdev.sample.jaxws.web.handler.TestLogicalHandler(inbound)
wasdev.sample.jaxws.web.handler.RouteTracker
wasdev.sample.jaxws.web.handler.TestLogicalHandler(outbound)
wasdev.sample.jaxws.web.handler.TestSOAPHandler(outbound)
wasdev.sample.jaxws.web.handler.TestLogicalHandler is closed
wasdev.sample.jaxws.web.handler.TestSOAPHandler is closed
wasdev.sample.jaxws.web.handler.client.TestClientSOAPHandler(inbound)
wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler(inbound)
wasdev.sample.jaxws.web.handler.client.TestClientSOAPHandler is closed
wasdev.sample.jaxws.web.handler.client.TestClientLogicalHandler is closed
```

7.5.6 Web Service message context

Open `webservicecontext_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `/WebServiceContextServlet`:

```
<form action="<%=request.getContextPath()%>/WebServiceContextServlet"
   target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.webservicecontext.client.WebServiceContextServlet`, and note that it defines a variable of type `WebServiceContextQueryService` with the `@WebServiceRef` annotation, and it calls the `query` method on the web service to get a string:

```
@WebServiceRef(value = WebServiceContextQueryService.class)
private WebServiceContextQuery contextQuery;

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    String echoParameter = request.getParameter("submit");
    if (echoParameter != null) {
        setEndpointAddress((BindingProvider) contextQuery, request,
            "WebServiceContextQueryService");
        String responseMessage = contextQuery.query();
        request.setAttribute("output", responseMessage);
    }
    request.getRequestDispatcher("webservicecontext_sample.jsp").forward(request, response);
}
```

Open `wasdev.sample.jaxws.web.webservicecontext.client.WebServiceContextQueryService`, and note that it connects to a URL path at `/WebserviceContextQueryService`:

```
url = new URL(baseUrl,
"http://localhost:9080/JaxWsLibertyDemo/WebServiceContextQueryService?wsdl");
```

Open `wasdev.sample.jaxws.web.webservicecontext.client.WebServiceContextQuery.java`, the service interface, and note that it is defined as a web service with one method `query`.

Open `wasdev.sample.jaxws.web.webservicecontext.WebServiceContextQuery.java`, and note that it simply returns all the properties in the `MessageContext` as a string.

7.5.7 Catalog

Open `catalog_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `/CatalogclientServlet`:

```
<form action="<%=request.getContextPath()%>/CatalogClientServlet"
    target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.catalog.client.CatalogServlet.java`, and note that it defines a variable `calculatorPortTypes` with the @WebServiceRef annotation mapping to the service endpoint interface `Calculator`:

```
@WebServiceRef(value = Calculator.class)
private CalculatorPortType calculatorPortType;
```

Also note the call to `setEndpointAddress` in the base class uses `/Calculator` as path:

```
setEndpointAddress((BindingProvider) calculatorPortType, request, "Calculator");
```

Open `wasdev.sample.jaxws.web.catalog.client.Calculator.java`, and note that it is defined with the @WebServiceClient annotation, with a non-existent `wsdlLocation` of `http://foo.org/calculator.wsdl`.

```
@WebServiceClient(name = "Calculator",
   targetNamespace = "http://catalog.web.jaxws.sample.wasdev",
   wsdlLocation = "http://foo.org/calculator.wsdl")
public class Calculator extends Service
```

Open `WEB-INF/jax-ws-catalog.xml`. Note it redefines the location of the WSDL file to be at `wsdl/cauculator.wsdl`.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
   prefer="system">
   <system systemId="http://foo.org/calculator.wsdl" uri="wsdl/calculator.wsdl" />
</catalog>
```

Double check that `WEB-INF/wsdl/calculator.wsdl` exists.

With the WSDL available, the client is able to make a request to the server. Open `wasdev.sample.jaxws.web.catalog.Calculator.java` and note it defines a web service endpoint interface whose name is `CalculatorPortType`.

Open `wasdev.sample.jaxws.web.catalog.CalculatorService.java` and note that it implements the web service endpoint's method to add two numbers.

7.5.8 MTOM support

MTOM stands for Message Transmission and Optimization Mechanism, a way to send binary data. The MTOM sample transmits binary image between client and server, with or without MTOM enabled.

Run the sample with MTOM enabled, and note that the message is a MIME multi-part message with XOP (XML-Binary Optimized Packaging), similar to this:

```
Request Message: Accept : [*/*] Cache-Control : [no-cache] connection : [keep-alive]
Content-Length : [1270] content-type : [multipart/related; type="application/xop+xml";
boundary="uuid:859cdb17-ddc3-4536-9a70-313c789b592f"; start="<root.message@cxf.apache.org>";
start-info="text/xml"] Host : [localhost:9080] Pragma : [no-cache] SOAPAction : [""] User-
Agent : [Apache CXF 2.6.2-ibm-s20130829-0230] ------=_Part_0_1837574876.1434745215329
Content-Type: text/xml; charset=utf-8 <soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Header><Action
xmlns="http://www.w3.org/2005/08/addressing">http://jaxws.service/ImageServiceImpl/uploadIma
geRequest</Action><MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:5eb16644-
5d17-4f21-be54-3a3d2c0d8e84</MessageID><To
xmlns="http://www.w3.org/2005/08/addressing">http://localhost:9080/JAXWSWebSample/ImageServi
ceImplService</To><ReplyTo
xmlns="http://www.w3.org/2005/08/addressing"><Address>http://www.w3.org/2005/08/addressing/a
```

```
nonymous</Address></ReplyTo></soap:Header><soap:Body><a:uploadImage
xmlns:a="http://jaxws.service/"><arg0>111</arg0><arg1><xop:Include
xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:2b3bd2bb-c38a-4781-9822-
cafe8cbde00b-1@cxf.apache.org"/></arg1></a:uploadImage></soap:Body></soap:Envelope> ------
=_Part_0_1837574876.1434745215329 Content-Type: application/octet-stream Content-Transfer-
Encoding: binary Content-ID: <2b3bd2bb-c38a-4781-9822-cafe8cbde00b-1@cxf.apache.org>
———————————————————————————— ------=_Part_0_1837574876.1434745215329--
```

Run the sample with MTOM disabled, and note that the message is a regular SOAP message, similar to this:

```
Request Message: Accept : [*/*] Cache-Control : [no-cache] connection : [keep-alive]
Content-Length : [710] content-type : [text/xml; charset=UTF-8] Host : [localhost:9080]
Pragma : [no-cache] SOAPAction : [""] User-Agent : [Apache CXF 2.6.2-ibm-s20130829-0230]
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Header><Action
xmlns="http://www.w3.org/2005/08/addressing">http://jaxws.service/ImageServiceImpl/uploadIma
geRequest</Action><MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:c8f12ba7-
c2f6-48e1-9d79-335926607071</MessageID><To
xmlns="http://www.w3.org/2005/08/addressing">http://localhost:9080/JAXWSWebSample/ImageServi
ceImplService</To><ReplyTo
xmlns="http://www.w3.org/2005/08/addressing"><Address>http://www.w3.org/2005/08/addressing/a
nonymous</Address></ReplyTo></soap:Header><soap:Body><a:uploadImage
xmlns:a="http://jaxws.service/"><arg0>111</arg0><arg1>AAECAw==</arg1></a:uploadImage></soap:
Body></soap:Envelope>
```

Open `wsfeatures_sample.jsp` and note how it displays the instructions for the demo in the `.jsp`, and then routes the request to `/ImageClientServlet`:

```
<form action="<%=request.getContextPath()%>/ImageClientServlet"
    target='_self' method='POST'>
```

Open `wasdev.sample.jaxws.web.wsfeatures.client.ImageClientServlet.java`, and note that it defines a variable called `MOCK_IMAGE_BYTES` that contains the bytes to be transmitted. It also defines two `@WebServiceRef`, where `mtomEnabledImageService` transmits the image with MTOM enabled via `@MTOM` annotation, while `mtomDisabledImageService` transmits the image with MTOM disabled:

```
private static final byte[] MOCK_IMAGE_BYTES = { 0, 1, 2, 3 };

@MTOM
@WebServiceRef(value = ImageServiceImplService.class)
private ImageServiceImpl mtomEnabledImageService;

@WebServiceRef(value = ImageServiceImplService.class)
private ImageServiceImpl mtomDisabledImageService;
```

Open `wasdev.sample.jaxws.web.wsfeatures.client.ImageServiceImplService.java`, and note that it defines a web service client, with URL to access the server.

Open `wasdev.sample.jaxws.web.wsfeatures.client.ImageServiceImpl.java`, and note that it defines a web service endpoint with a single method to upload an array of bytes, and to receive an array of bytes in return:

```
@WebMethod
@WebResult(targetNamespace = "")
@RequestWrapper(localName = "uploadImage",
   targetNamespace = "http://jaxws.service/",
   className = "wasdev.sample.jaxws.web.wsfeatures.client.UploadImage")
@ResponseWrapper(localName = "uploadImageResponse",
   targetNamespace = "http://jaxws.service/",
   className = "wasdev.sample.jaxws.web.wsfeatures.client.UploadImageResponse")
public byte[] uploadImage(
      @WebParam(name = "arg0", targetNamespace = "") String arg0,
      @WebParam(name = "arg1", targetNamespace = "") byte[] arg1);
```

For the server side, open `wasdev.sample.jaxws.web.wsfeatures.ImageServiceImpl.java`, and note that it just returns whatever the handler places on the message context. The content differs depending on whether or not the message is sent via MTOM:

```
return ((String) webServiceContext.getMessageContext().get("request.message")).getBytes();
```

Open `wasdev.sample.jaxws.web.wsfeatures.MessageInfoHandler.java`, and note that for incoming request, it retrieves all the HTTP headers and places them on the message context.

7.6 Cleanup

__1. Stop the server **JAXWSServer** from Eclipse.

Appendix A. Notices

Appendix B. Trademarks and copyrights

NOTES

NOTES

WebSphere® software