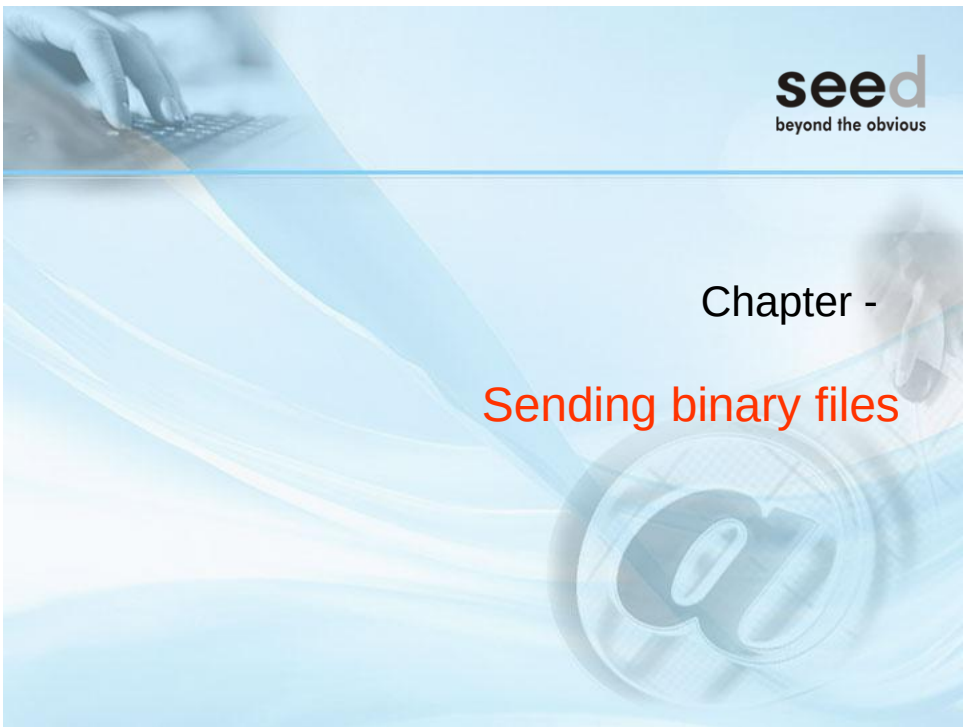


Objectives

- At the end of this chapter you will be able to understand
  - How to receive and return binary files in our web service.

Providing the image of a product

- Suppose that we'd like to have a web service to allow people to upload the image (jpeg) of a product (identified by a product id).
- The SOAP message may be like:

```
<Envelope>
  <Body>
    <uploadImage>
      <productId>p01</productId>
      <image>kdubn87kamlndy...</image>
    </uploadImage>
  </Body>
</Envelope>
```

Typically binary data such as the image is encoded using the base64 encoding

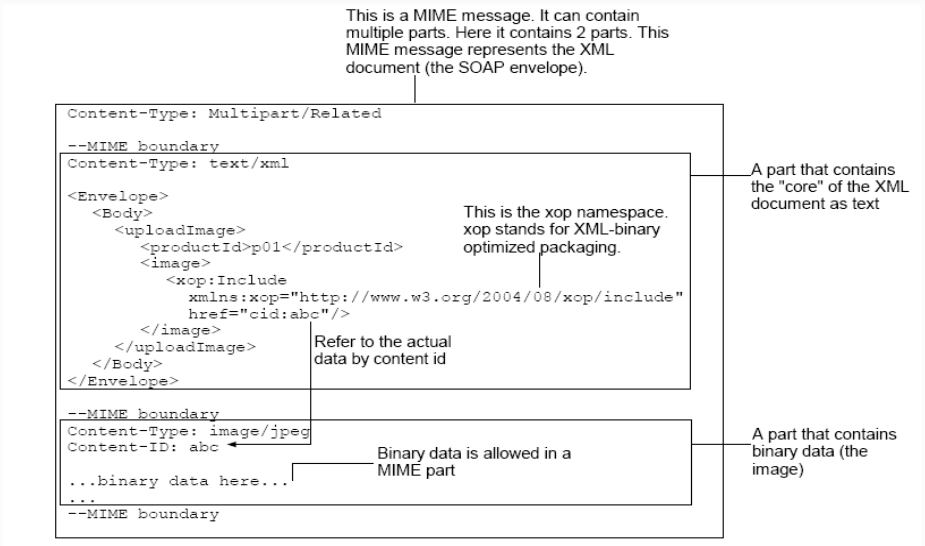
seed
beyond the obvious

Providing the image of a product

- The problem is that the base64 encoded data will be much larger than the binary version. This wastes *processing time*, *network bandwidth* and *transmission time*. In fact, if the image is huge, then many XML parsers may not be able to handle it properly.
- To solve this problem, instead of always representing an XML document as text, it will be better if it can be represented as a MIME message.
- For example, the above XML document (SOAP envelope) can be represented as below without changing its meaning:

seed
beyond the obvious

Providing the image of a product



Providing the image of a product

- To implement this idea, create a new project named *ImageService* as usual.
- Modify the WSDL file:

Providing the image of a product - WSDL file

Use a urn as the target namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="urn:ttdev.com:service/img"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="ImageService"
   targetNamespace="urn:ttdev.com:service/img">
   <wsdl:types>
     <xsd:schema targetNamespace="urn:ttdev.com:service/img"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="uploadImage">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="productId" type="xsd:string" />
               <xsd:element name="image" type="xsd:base64Binary" />
             </xsd:sequence>
           </xsd:complexType>
        </xsd:element>
     </xsd:schema>
   </wsdl:types>
```

It will contain binary data. It is basically to be encoded using base64. Later you will tell Axis to use XOP for it.

seed
beyond the obvious

7

Providing the image of a product - WSDL file

```
   <wsdl:message name="uploadImageRequest">
     <wsdl:part name="parameters" element="tns:uploadImage" />
   </wsdl:message>
   <wsdl:portType name="ImageService">
     <wsdl:operation name="uploadImage">
        <wsdl:input message="tns:uploadImageRequest" />
     </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="ImageServiceSOAP" type="tns:ImageService">
     <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
     <wsdl:operation name="uploadImage">
        <soap:operation
          soapAction="urn:ttdev.com:service/img/uploadImage" />
        <wsdl:input>
          <soap:body use="literal" />
        </wsdl:input>
     </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ImageService">
     <wsdl:port binding="tns:ImageServiceSOAP"
        name="ImageServiceSOAP">
        <soap:address
          location="http://localhost:8080/axis2/services/ImageService" />
     </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

The operation doesn't return anything, so there is no output message.

seed
beyond the obvious

8

Providing the image of a product - build file

- Although this is not required, it uses the wrapped convention. Next, update *build.xml*:

seed
beyond the obvious

Providing the image of a product - build file

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="jar.server">
  ...
  <property name="name" value="ImageService" />
  ...
  <target name="generate-service">
    <wsdl2code
      wsdlfilename="${name}.wsdl"
      serverside="true"
      generateservicexml="true"
      skipbuildxml="true"
      serversideinterface="true"
      namespacetopackages="urn:ttdev.com:service/img=com.ttdev.image"
      targetsourcefolderlocation="src"
      targetresourcesfolderlocation="src/META-INF"
      overwrite="true"
      unwrap="true" />
    <replaceregexp
      file="src/META-INF/services.xml"
      match="${name}Skeleton"
      replace="${name}Impl" />
  </target>
  <target name="generate-client">
    <wsdl2code
      wsdlfilename="${name}.wsdl"
      skipbuildxml="true"
      namespacetopackages="urn:ttdev.com:service/img=com.ttdev.image.client"
      targetsourcefolderlocation="src"
      overwrite="true"
      unwrap="true" />
  </target>
</project>
```


seed
beyond the obvious

Providing the image of a product

- Generate the service stub and client stub. Check the implementation class:

```
public class ImageServiceSkeleton implements ImageServiceSkeletonInterface {
    public void uploadImage(
        java.lang.String productId1,
        javax.activation.DataHandler image2) {
    }
}
```

- Note that the binary image data is presented as a *DataHandler* object.
- To read the data from it, create an *ImageServiceImpl* class:

seed
beyond the obvious

Providing the image of a product – Implementation class

This is how you get the content type from a DataHandler

A DataHandler represents a MIME part above: It has a content type and some data (bytes).

This is how you get the data from a DataHandler

```
public class ImageServiceImpl implements ImageServiceSkeletonInterface {

   public void uploadImage(String productId, DataHandler image) {
      System.out.println(image.getContentType());
      try {
         InputStream in = image.getInputStream();
         String imageDir = "c:/tmp";
         FileOutputStream out = new FileOutputStream(new File(imageDir,
                  productId));
         try {
            byte buf[] = new byte[1024];
            for (;;) {
               int noBytesRead = in.read(buf);
               out.write(buf, 0, noBytesRead);
               if (noBytesRead < buf.length) {
                  break;
               }
            }
         } finally {
            out.close();
         }
      } catch (IOException e) {
         throw new RuntimeException(e);
      }
   }
}
```

Copy the jpeg file data into c:\tmp. The file is named after the product id (e.g., c:\tmp\p01).

seed
beyond the obvious

Providing the image of a product

- Create an *ImageClient.java* file in the client package:

seed
beyond the obvious

Providing the image of a product - Client

```
import javax.activation.DataHandler;
import javax.activation.DataSource;
import javax.activation.FileDataSource;

public class ImageClient {
   public static void main(String[] args) throws RemoteException {
      ImageServiceStub service = new ImageServiceStub();
      service._getServiceClient().getOptions().setProperty(
              Constants.Configuration.ENABLE_MTOM, "true");
      DataSource source = new FileDataSource("c:/axis/docs/xdocs/1_3/images/axis.jpg");
      DataHandler handler = new DataHandler(source);
      service.uploadImage("p01", handler);
      System.out.println("Done!");
   }
}
```

Critical point: Enable MTOM. MTOM stands for message transmission optimization mechanism. It means the same thing as XOP when it is applied to SOAP messages. The effect is, whenever it needs to send base64 encoded data, it will send it using XOP.

You need to make sure this file exists

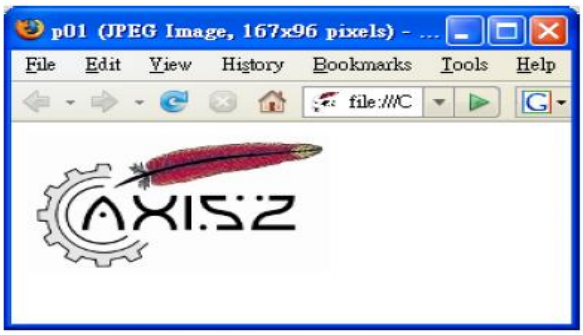
Create a DataHandler object that reads that DataSource object

Create a DataSource object that will read the data from the file. It will also find out the MIME type (image/jpeg in this case) from the file extension (.jpg).

seed
beyond the obvious

Providing the image of a product - Client

- We should find a new file p01 in c:\temp .
- We can verify that it's a copy of *axis.jpg* by opening it in a browser:



seed
beyond the obvious

Providing the image of a product – TCP monitor

```
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_6D8E7B2093DFD9FC5B1195966468539;
SOAPAction: "urn:ttdev.com:service/img/uploadImage"
User-Agent: Axis2
Host: 127.0.0.1:1234
Transfer-Encoding: chunked

2339
--MIMEBoundaryurn_uuid_6D8E7B2093DFD9FC5B1195966468539
Content-Type: application/xop+xml; charset=UTF-8; type="text/xml"
Content-Transfer-Encoding: binary
Content-ID: <0.urn:uuid:6D8E7B2093DFD9FC5B1195966468540@apache.org>
   <?xml version='1.0' encoding='UTF-8'?>
      <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
         <soapenv:Body>
            <ns1:uploadImage xmlns:ns1="urn:ttdev.com:service/img">
               <productId>p01</productId>
               <image>
                  <xop:Include href="cid:1.urn:uuid:6D8E7B2093DFD9FC5B1195966468650@apache.org"
               </image>
            </ns1:uploadImage>
         </soapenv:Body>
      </soapenv:Envelope>
--MIMEBoundaryurn_uuid_6D8E7B2093DFD9FC5B1195966468539
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <1.urn:uuid:6D8E7B2093DFD9FC5B1195966468650@apache.org>
ÿØÿà  JFIF     H H  ÿÛ C                                                    ÿÛ C
 ÿÄ µ               }        !1A  Qa "q 2□'¡ #B±Á RÑð$3br,
```

MIME message (multipart/related)

Refer to the binary data using cid (content id)

The binary data

seed
beyond the obvious

Enabling MTOM in the service

- For the moment, it is our client that needs to send a file. If it was our web service that needed to do that, we would need to enable MTOM in the service.
- To do that, modify *services.xml*:

seed
beyond the obvious

Enabling MTOM in the service - descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
    <service name="ImageService">
        <messageReceivers>
            <messageReceiver mep="http://www.w3.org/ns/wsdl/in-only"... />
        </messageReceivers>
        <parameter name="ServiceClass">
          com.ttdev.image.ImageServiceImpl
        </parameter>
        <parameter name="useOriginalwsdl">true</parameter>
        <parameter name="modifyUserWSDLPortAddress">true</parameter>
        <parameter name="enableMTOM">true</parameter>
        <operation name="uploadImage" mep="http://www.w3.org/ns/wsdl/in-only">
            <actionMapping>urn:ttdev.com:service/img/uploadImage</actionMapping>
        </operation>
    </service>
</serviceGroup>
```


seed
beyond the obvious

Interoperability

- If we need to send binary files to others, make sure the other side supports MTOM.
- For example, for .NET, MTOM is supported with WSE (Web Services Enhancements) 3.0 or later.

Quick Recap . . .

- XOP stores XML elements that is of the type xsd:base64Binary as MIME parts and represents the whole XML document as a MIME message. When the XML document is a SOAP envelope, it is called MTOM.
- To receive a binary file using MTOM, if the receiver is written with Axis2, for maximum interoperability, it can always handle incoming messages using MTOM without any configuration
- To send a binary file using MTOM, enable MTOM in the sender.