Lab 9: Document Style Messaging with Attachments

In this lab, you will create a Web service that accepts a PDF file as an attachment. You will then write a client program that sends the PDF.

Objectives:

- To work with non-well-formed data in Web services

Part 1: Writing the Service

In this part, you will create a bottom-up Web service that accepts an at

Steps:

_1. Create a new Dynamic Web project named **Lab09Web** associated with *lab09EAR*.

_2. In the new project, create a class named **ProcessPDF** in a new package named **com.goliath.pdf** package.

_3. Complete the class:

a. Write @WebService and @BindingType annotations on the class so that it uses SOAP 1.1 and MTOM.

b. Write a method named **processPDF** that accepts an argument of type String for a filename, and an argument of type *javax.activation.DataHandler* for the PDF. The method should return a String.

c. Annotate the second argument with **@XmlMimeType**:

```
public String processPDF(String fname,
        @XmlMimeType("application/pdf") DataHandler src)
{
}
```

This will cause the generated WSDL to specify "application/pdf" as the *xmime:expectedContentTypes* for the parameter. Since the parameter's type is DataHandler, the schema type will be *xs:base64Binary*. The combination of these inform any client exactly what the service expects.

d. Check the content-type of the request and if it's not PDF, return a message to that effect:

```
String contentType = src.getContentType();
if (!contentType.equals("application/pdf"))
{
    return "Input is not PDF: " + contentType;
}
```

e. Write code to copy the input PDF stream into a file. Your code might look something like (exception handling omitted):

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
File outFile = new File(fname);
System.out.println("Output file: " + outFile.getAbsolutePath());
FileOutputStream fos = new FileOutputStream(outFile);
InputStream is = src.getInputStream();
int abyte = is.read();
while(abyte != -1)
{
    fos.write(abyte);
    abyte = is.read();
}

is.close();
fos.close();
```

You can use the *Source - Surround With - Try/Catch Block* menu to generate exception-handling code.

f. Return the string "Success".

_4. Right-click on the WebSphere server and add the project to the server. You can remove any older projects.

_5. Open an external browser, and enter the following URL to view the service's WSDL:

```
http://localhost:9080/Lab09Web/ProcessPDFService?wsdl
```

Then examine the service's WSDL and schema. The schema's URL is:

```
http://localhost:9080/Lab09Web/ProcessPDFService/
   ProcessPDFService_schema1.xsd
```

Looking at the schema, find the *processPDF* complexType and verify that it specifies a type of *base64Binary* and expectedContentTypes of *application/pdf*.

Part 2: Writing the Client

In this part, you will write a client that sends a PDF to the service.

Steps:

_1. Create a new Java project named **lab09Client**.

_2. As you have done in earlier labs, generate client stubs into the client project.

_3. Use Windows Explorer to find a PDF file on your hard disk (if you can't find one, you can use the *serverStartup.pdf* file in the *lab08* "starters" folder).

Use Windows Explorer to copy the PDF file to the **root** directory of the project (not the *src* folder).

Then, back in RAD, refresh the project to see the PDF file.

_4. Create a new class named **SendPDF** in a new package named *client*. The class should have a *main* method.

Complete the *main* method:

a. Create a service object and retrieve a proxy for the service as you have done in earlier labs.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

b. Create a File object that represents the PDF:

```
File pdfFile = new File("xxxxxx.pdf");
```

Substitute your PDF file's name for the xxxxxx.

c. Create an object that associates a MIME type with a file extension:

```
MimetypesFileTypeMap mftm = new MimetypesFileTypeMap();
mftm.addMimeTypes("application/pdf pdf PDF");
```

d. Create a *java.activation.DataSource* that references the file:

```
FileDataSource ds = new FileDataSource(pdfFile);
ds.setFileTypeMap(mftm);
DataHandler dh = new DataHandler(ds);
```

e. As a debugging aid, output the file's content-type and size:

```
System.out.println("PDF length: " + pdfFile.length());
System.out.println("Content type: " + dh.getContentType());
```

f. Enable MTOM:

```
BindingProvider bp = (BindingProvider)proxy;
SOAPBinding sb = (SOAPBinding)bp.getBinding();

sb.setMTOMEnabled(true);
```

Note: The SOAPBinding type is from the *javax.xml.ws.soap* package.

g. Invoke the service method and display the returned String to the console.

_5. Run the client as a Java application and note the returned String.

Look in the Console to find the full pathname of the PDF "on the server", then use Windows Explorer to verify that the PDF was copied.

_6. As in earlier labs, ensure that the TCP/IP monitor is running, then add code to the client so that it sends requests via the monitor.

Run the client and then look in the monitor and examine the request message. Note how the attachment part follows the Envelope.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.