Lab 3: XML Review

In this lab, you will review the basic XML concepts required for understanding Web services.

Objectives:

- To work with well formed XML
- To write an XML Schema for valid XML

Part 1: Working with Well Formed XML

One of the benefits of XML is that it lets you define content that can be transformed into various forms. In this lab, the fictional Cogswell Company would like to make their travel policy guidelines available on their intranet as well as printed in an employee handbook.

In this part, you will convert a text document that describes the travel policy for Cogswell into well formed XML.

Steps:

_1. Ask your instructor for the following information:

Lab Installation directory: _____

RAD Installation directory: _____

If your machine has the standard lab setup on Windows XP, the information will be something like:

Setting	Value
Lab Installation directory:	c:\websrvclass
RAD Installation directory	C:\Program Files\IBM\SDP

If your computer is running Windows Vista, Windows 7 or later, the standard directories are:

Setting	Value
Lab Installation directory:	c:\Users\username\websrvclass
RAD Installation directory	C:\Program Files\IBM\SDP

Please substitute your actual user name, e.g. *libtrain1* for the "username" above.

_2. Start RAD from by selecting *IBM Software Development Platform - Rational Application Developer Standard Edition* the Windows Start menu.

Note: If your computer has Windows Vista, Windows 7 or later, right-click on the RAD Start menu entry and choose *Run As Administrator*, then OK the access.

When prompted for the *Workspace Location*, enter the **{Lab Installation Directory}\workspace**, for example, on Windows XP:

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

```
c:\websrvclass\workspace
```

On Windows Vista, Windows 7 or later:

```
c:\Users\username\websrvclass\workspace
```

Please substitute your actual user name, e.g. *libtrain1* for the "username" above.

You can close the *Rational Software Development Platform* welcome window if you like by clicking the 'X' in its title-bar tab.

_3. Your next job is to create a *project* for the lab. Follow these steps:

a. Choose **File - New - Project** to start the wizard.

b. On the first wizard page, choose **General - Project**, then press Next.

c. Enter **lab03** for the *Project name*, then press Finish.

_4. Next, you will import a text file containing the "raw" travel policy that you will convert to XML. Follow these steps carefully:

a. In the Project Explorer pane, right-click on the new project and choose *Import...* to start the import wizard.

b. On the first wizard page, choose *General - File system*, then press Next.

c. On the next wizard page, for the *From directory*, press the Browse button and navigate to the **{Lab Installation Directory}/starters/lab03** folder, then press OK. If your computer has the standard lab setup on Windows XP, the proper directory is:

```
c:\websrvclass\starters\lab03
```

If your computer is running Windows Vista or later, the standard directory is:

```
c:\Users\username\websrvclass\starters\lab03
```

Substitute your actual user name, e.g. *libtrain1* for the "username" above.

In the left pane, click on *lab03* to highlight it, then put a checkmark next to the the *exam.xml*, *exam.xsd* and *policy.txt* entries in the right pane. The dialog should look like the one in Figure 1:

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

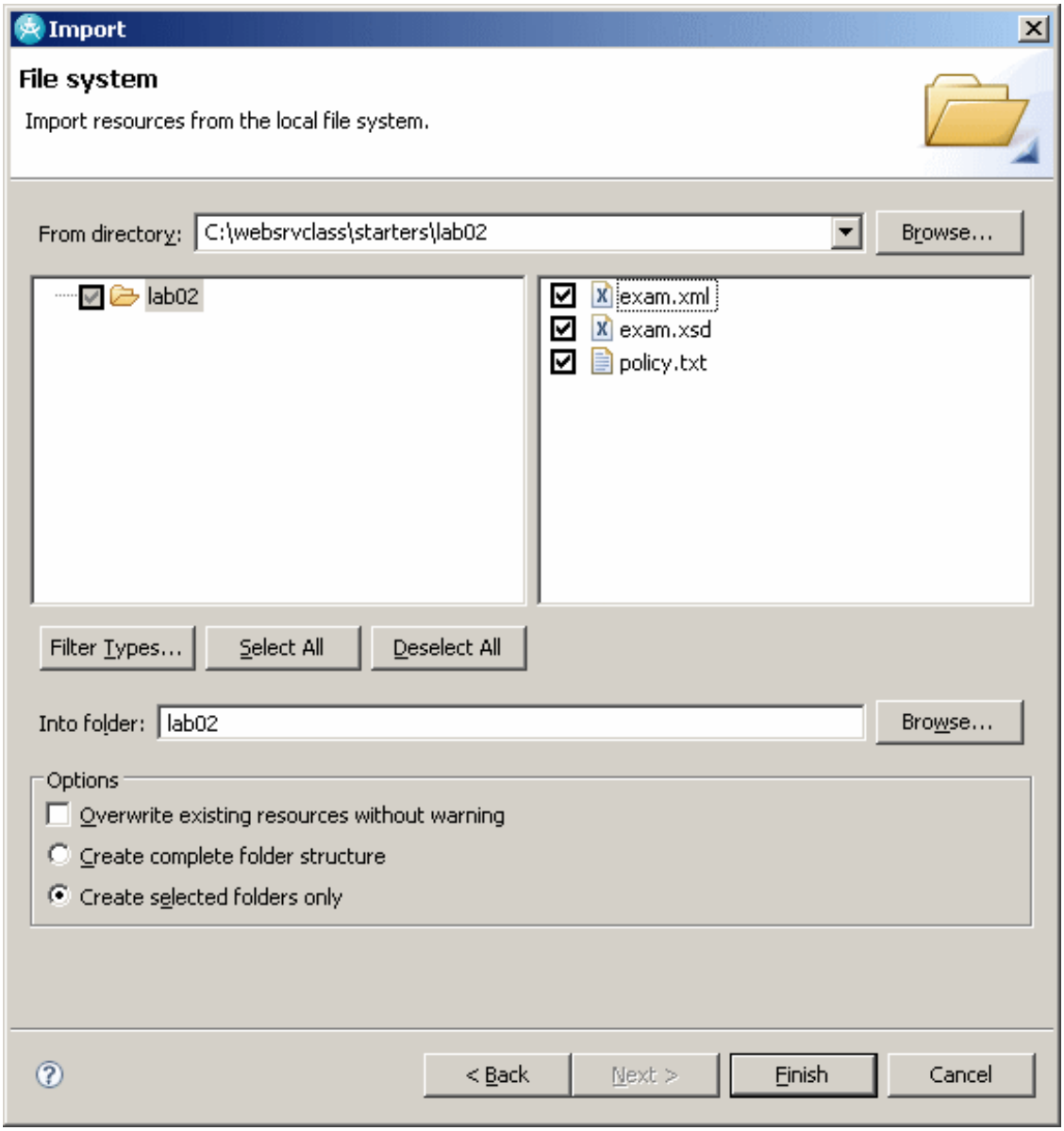


Figure 1: Import Dialog

Press Finish to complete the import. Back in the Project Explorer pane, you can expand the lab03 project to see the imported files.

_5. Open the *policy.txt* file into the text editor by double-clicking on it.

Note the structure of the document: it contains a title, a set of definitions and then the set of policy rules that govern travel for Cogswell employees. Your first job is to visualize the document as a tree of content -- that will make the transition to XML easier. Sketch the tree structure of the document below:

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

_6. Copy the file to an XML file using this procedure:

   a. In the Project Explorer, right-click on *policy.txt* and choose **Copy**.

   b. Right-click on the *lab03* project and choose **Paste** -- enter **policy.xml** for the name of the new file.

   c. Double-click on the XML file to open it into the XML editor.

   d. Press the *Source* tab at the bottom of the editor to see the raw text.

_7. In the Project Explorer, right-click on the policy.xml file and choose *Validate* -- you should see an error indicating that the file does not contain well formed XML.

_8. Your job is to add XML markup to the file so it becomes well formed, self-describing data.

   Start by adding the *XML declaration* at the top of the file.

   As you add markup, you can right-click on the file and choose *Validate*, which will cause RAD to check it for well-formdness and show any problems by drawing a red X in the editor pane's left margin. At this point, the XML is not yet well formed.

_9. Using your tree sketch from above, separate the text content into XML elements. Be sure to close each element! Here are some hints:

   - The entire document should be contained in a **policy** element
   - It's a good idea to create *aggregating elements* that hold child elements. For example, it would be good to create an enclosing **definitions** element that contains **definition** elements and a **rules** element enclosing **rule** elements. Remember that it's OK and common to have more than one element with the same name!
   - You can write a **heading** element as a child of the **policy** element that contains the text *General Travel Policy*.

   Note: We recommend that you indent the elements and text to make the XML more readable.

_10. As you finish elements, use the Validate XML File menu to check the document for well formedness. When you are finished, it should report that the file is well formed.

   You should also be able to see the tree structure of your XML by looking in the *Outline* pane.

_11. Compare your travel-policy XML against the version stored in the **{Lab Installation Directory}/solutions/lab03** folder.

Part 2: Writing an XML Schema

In this part, you will write an XML Schema to validate XML content that represents an exam.

Steps:

_1. Edit the *exam.xml* that you imported in the last part and familiarize yourself with this *instance document*. Note that it contains no namespaces and does not reference a schema.

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

_2. Next, edit the starter schema named **exam.xsd** that you imported in an earlier part.

Note how it defines the root element and required namespace information for a schema.

_3. Edit *exam.xml* and modify the *test* start tag so it references the schema:

```
<test id="XM30-XML"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="exam.xsd">
```

In the Project Explorer, right-click on the XML file and choose *Validate XML File* -- you will see an error due to the fact that you have not completed the schema. Your job is to complete the schema so that it validates this instance document.

_4. Edit the schema. Start by defining globally scoped *element* definitions for the simple PCDATA elements in the content:

- **description**
- **questionText**
- **choiceText**
- **correct**

For example:

```
<xs:element name="description" type="xs:string"/>
```

As you edit the schema, save it, then go back to the instance document and use the Validate XML File menu.

You should see no errors on the lines that use the definitions you just created.

_5. Next, define the "aggregator" elements as *complex types*, referencing the elements you defined in the last step. Use *minOccurs* and *maxOccurs* as appropriate. Do not worry about attributes yet. Note that you can use the *Outline* pane to help you visualize the source XML's tree structure.

- **choice**
- **choices**
- **question**
- **testQuestions**
- **test**

For example:

```
<xs:element name="choice">
  <xs:complexType>
    <xs:all>
      <xs:element ref="correct"/>
      <xs:element ref="choiceText"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```


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

Note: To specify an unlimited number of child elements, set *maxOccurs* to *unbounded*.

At this point, you should only have errors in *exam.xml* that refer to attributes. If you have any other errors, fix them before continuing.

_6. Now it's time to define the attributes. Define *xs:string* attributes for each of the elements that require them. For example, for the *choice* element:

```
<xs:attribute name="id" type="xs:string"/>
```

_7. Then test your validation as before -- you should have no errors!

_8. From the RAD main menu, choose *File* - *Close All* to close all of the editing windows.

Part 3: XML Namespaces

In this part, you will demonstrate your mastery of XML namespaces, which is a required skill for understanding Web services.

Steps:

_1. The following is a well-formed XML document. Please study it:

```
<env:Envelope  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <env:Body   env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <m:validate    xmlns:m="http://example.com/Lab05">
   <purchase     xmlns:n1="java:creditcard"
     xsi:type="n1:Purchase">
    <amount
       xsi:type="xsd:double">20.2</amount>
    <cardNumber
      xsi:type="xsd:string">sample string</cardNumber>
    <expiryDate
      xsi:type="xsd:dateTime">2005-07-03T16:52:11.218-05:00</expiryDate>
   </purchase>
  </m:validate>
 </env:Body>
</env:Envelope>
```

_2. Answer the following questions:

a. What is the namespace URI of the *Envelope* root element?

_____

b. What is the namespace URI of the *Body* element?

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

_____

c. What is the namespace URI of the *validate* element?

_____

d. What is the namespace URI of the *purchase* element?

_____

e. What is the namespace URI of the *encodingStyle* attribute within the *Body* element?

_____

f. What is the namespace URI of the *type* attribute within the *purchase* element?

_____

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.