Lab 2 Message Models and Working with XML Messages

2.1 Overview

In this lab, the IntroMessageFlow will be modified to identify the parser (XMLNSC) to be used to process the message.

The steps are very simple.

The properties of the Input node will be modified.

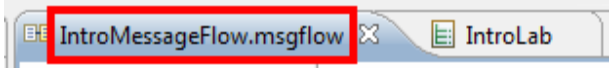
The Test Client will be used to run another test.

The trace file contents will be viewed to see the difference.

2.2 Using the XML Parser

__1. Return to the IBM Integration Toolkit.

__2. Click on the **IntroMessageFlow** tab to bring the message flow into view.



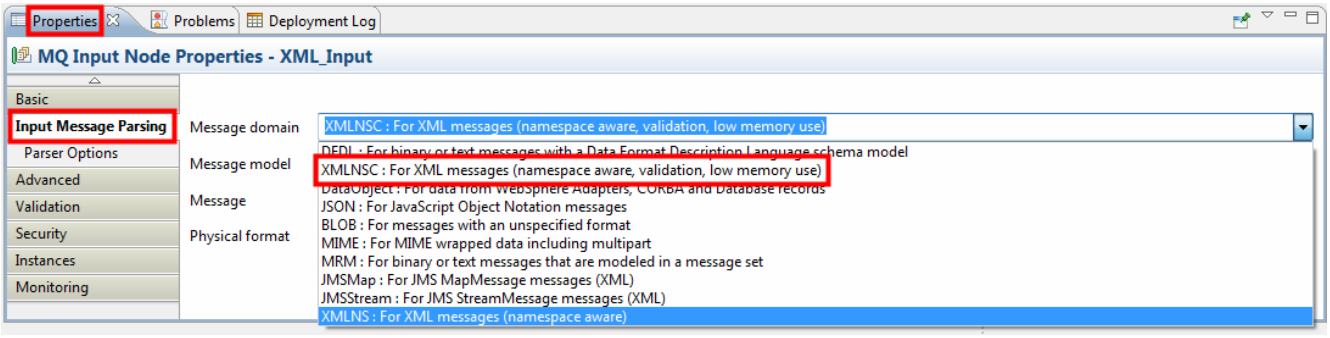
__3. Click on the **XML_Input** node to bring its properties into view.

The message flow will be modified so that it uses the XMLNSC parser to process the input message.

__4. On the **Properties** view at the bottom of the screen, click the **Input Message Parsing** tab. Since nothing was specified when the node was added, the Message domain (i.e. the parser) defaults to **BLOB** – which you saw in the trace.

__5. Click the pull-down for the **Message domain**. The various parsers are listed along with a short description. Depending on the Message domain selection, the other fields may be enabled or disabled.

__6. Select the **XMLNSC** parser. The **XMLNSC** parser that supports Namespaces (the NS part) and builds a more efficient or compact tree (the C part). The compact tree uses less memory.



__7. 

Save the message flow (**Ctrl+S**).

<u>Key Idea: Parsers and Message Domains</u>

IBM Integration Bus supplies a range of parsers to parse and write messages in different formats.

A parser is called when the bit stream that represents an input message must be converted to the format that is used internally by the broker; this process is known as parsing. The input is a bit stream, and the output is a logical tree representation of the message.

A serializer is called when a logical tree structure must be converted into a bit stream (for example on an Output node). This process is known as serializing.

Each parser is suited to a particular class of messages, known as a message domain. The following list contains some examples of the message domains used in IBM Integration Bus:

- XMLNSC – for XML documents
- DFDL – for general text or binary data streams including industry standards
- JSON – for JSON documents
- DataObject – for data without a stream representation (used by adapters)

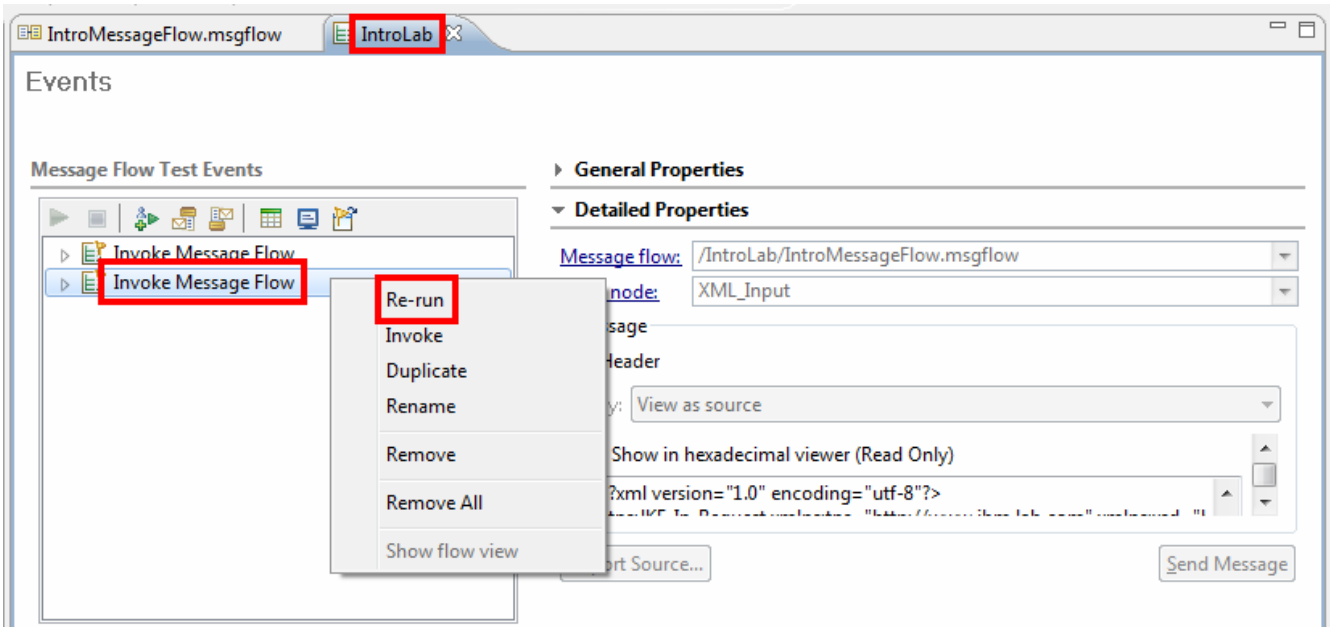
Now, let's re-run the Test Client.

__8. Switch back to the Test Client (**IntroLab** tab).

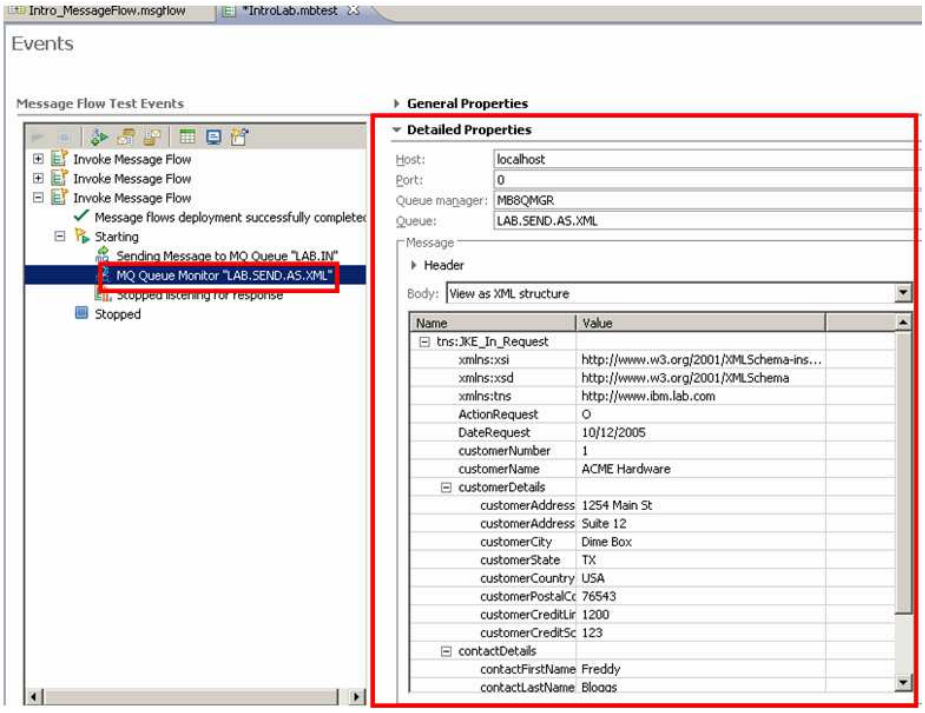
__9. Select one of the **Invoke Message Flow** items.

__10. Press the right mouse button.

__11. Select **Re-run** from the menu.



You again see the same output message from the Test Client.

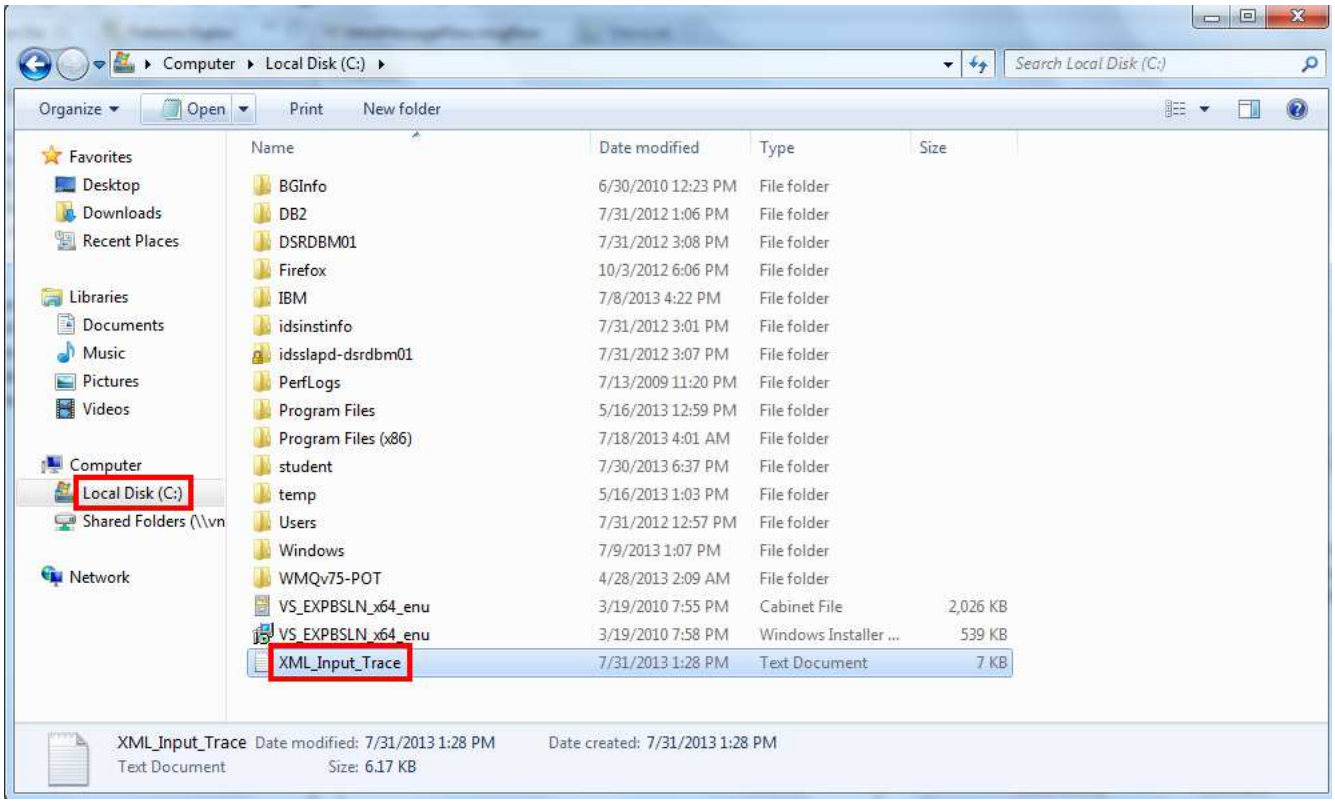


This is only what the output message looks like *after* it arrives on the output queue. Let's see what the message looked like *while* it was passing through the message flow.

__12. Return to Windows Explorer.

__13. Navigate to the file at **C:\XML_Input_Trace.txt**.

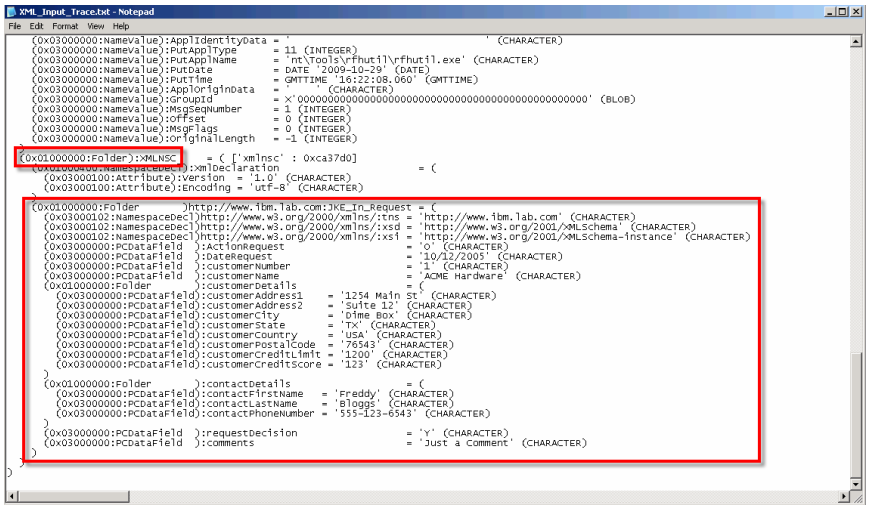
__14. Double click **on XML_Input_Trace.txt** file.



__15. Scroll to the end of the file (**Ctrl + End**).

Trace output is placed at the end of any existing content in a file so scroll down to the bottom of the file and view the results. Much more pleasing…here is a nicely formatted message tree that will allow you to conveniently access the fields in the XML message by name. Notice:

- The **XMLNSC** Domain name (which is what we set on the input node).
- All of the element types are String represented by the (CHARACTER)! Why is that? The answer requires understanding both Parsers and Message Models.



__16. Close the Notepad window.

__17. Minimize Windows Explorer.

2.3 Creating a Message Model from an XSD

In this portion of the lab, we will use a message model to parse the XML message.

Key Idea: Message Models

Much of the business world relies on the exchange of information between applications. This information is contained in messages that have a defined structure that is known and agreed by the sender and the receiver.

Applications typically use a combination of message formats, including those message formats that are defined by the following structures or standards:

- Comma Separated Values (CSV)
- COBOL, C, PL1, and other language data structures
- Industry standards such as SWIFT, X12 or HL7
- XML including SOAP

You can model a wide variety of message formats so that they can be understood by IBM Integration Bus message flows. When the message format is known, the broker can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow.

Some message formats are self-defining and can be parsed without reference to a model. However, most message formats are not self-defining, and a parser must have access to a predefined model that describes the message if it is to parse the message correctly.

An example of a self-defining message format is XML. In XML, the message itself contains metadata in addition to data values, and it is this metadata that enables an XML parser to understand an XML message even if no model is available. Another example of a self-defining format is JSON.

Examples of messages that do not have a self-defining message format are CSV text messages, binary messages that originate from a COBOL program, and SWIFT formatted text messages. None of these message formats contain sufficient information to enable a parser to fully understand the message. In these cases, a model is required to describe them.

Even if your messages are self-defining, and do not require modeling, message modeling has the following advantages:

- Runtime validation of messages. Without a message model, a parser cannot check whether input and output messages have the correct structure and data values.
- Enhanced parsing of XML messages. Although XML is self-defining, all data values are treated as strings if a message model is not used. If a message model is used, the parser is provided with the data type of data values, and can cast the data accordingly.
- Code completion assistance when coding transformation. When you are creating ESQL programs for your message flows, the ESQL editor can use message models to provide code completion assistance.
- Graphical mapping. Without message models, you cannot use the Message Mapping editor.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.
- Generation of documentation.

- Provision of version control and access control for message models by storing them in a central repository.

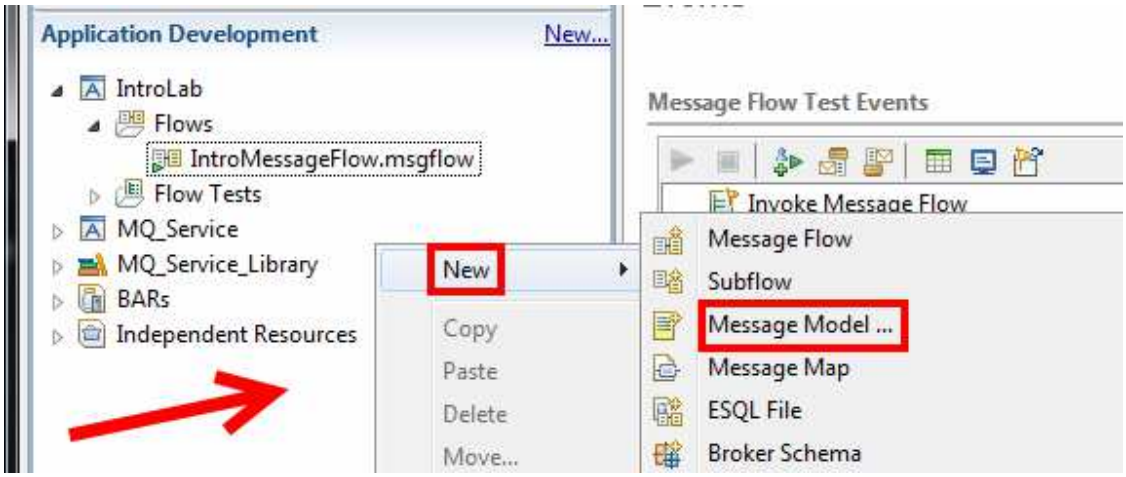
Message models allow the full use of the facilities that are offered by IBM Integration Bus.

To speed up the creation of message models, importers are provided to read metadata such as C header files, COBOL copybooks, and EIS (Enterprise Information System, such as SAP®) metadata, and to create message models from that metadata. Additionally, predefined models are available for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG.

The XML Parser was run in *programmatic* mode where it parsed the XML message, so it assumed everything was a string. By parsing with a model, we can get a message with typed elements and one that is subject to constraints (such as required fields, max field lengths, etc.). The toolkit provides wizards to import your existing models (such as WSDLs, XSDs, copybooks, etc.)

__1. In the Application Development view (project navigator) on the left, right click the whitespace.

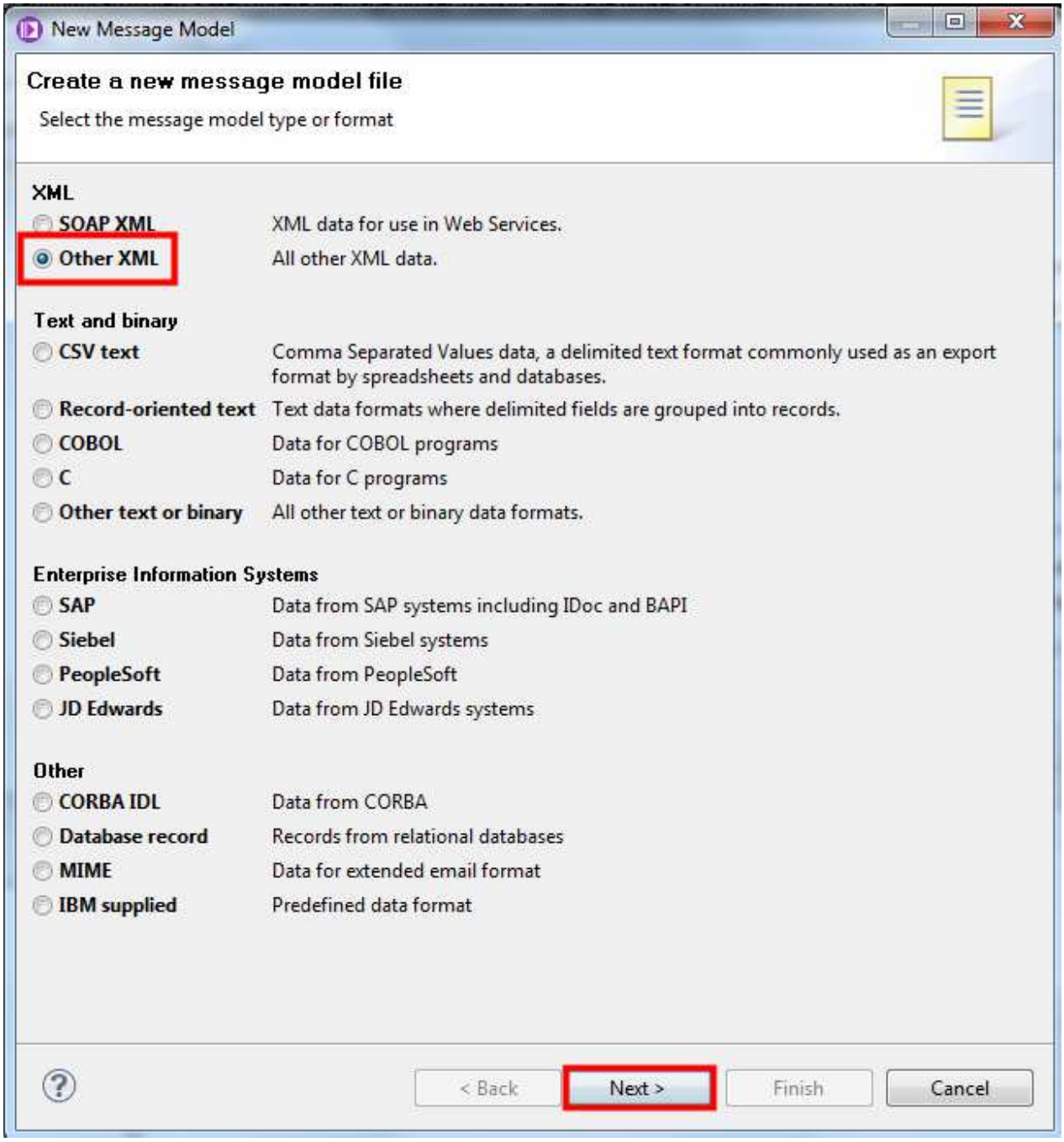
__2. Select **New→Message Model…** from the menu.



__3. Select the **Other XML** radio button (under **XML**).

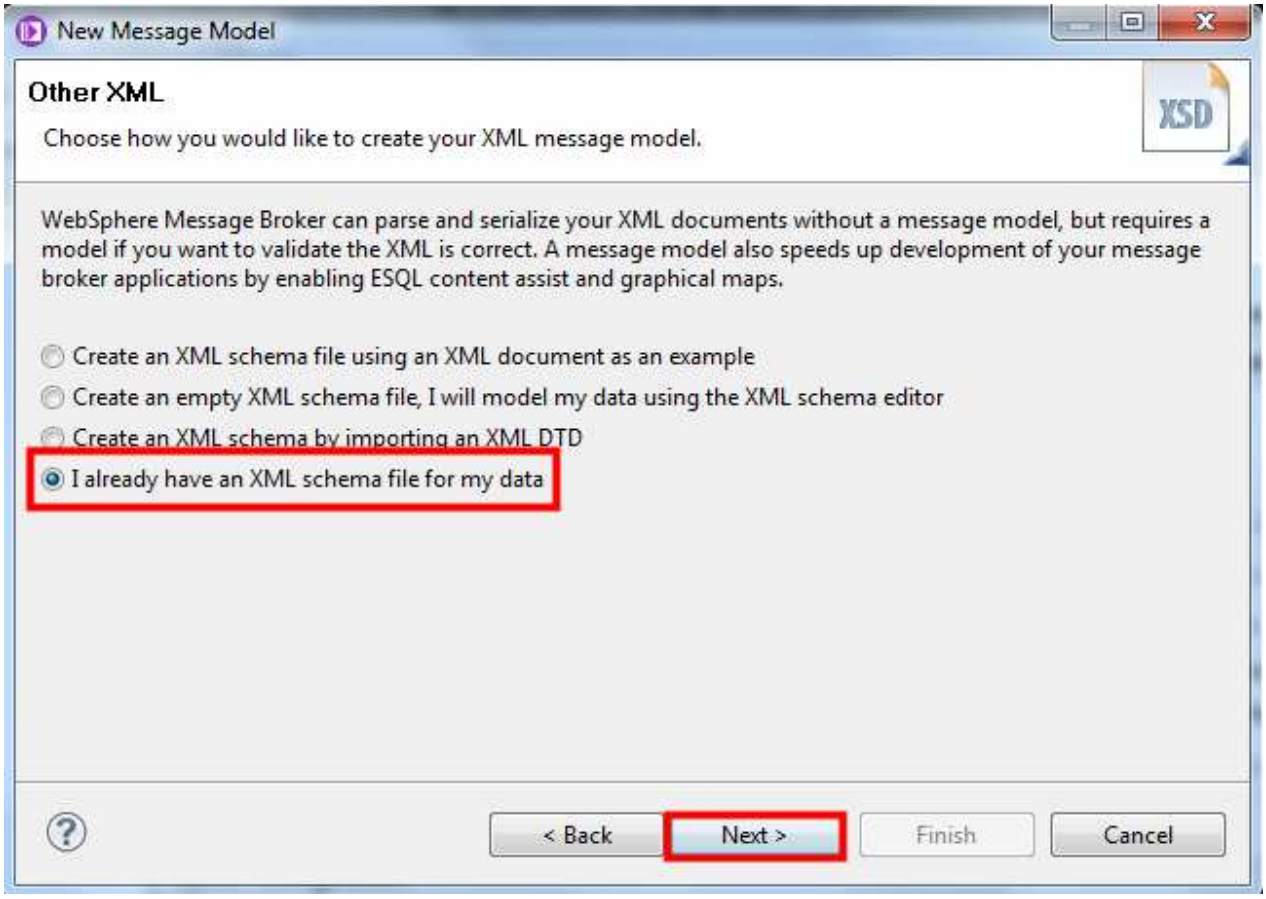
__4. Check out some of the other options for which there are import wizards.

__5. Click **Next**.

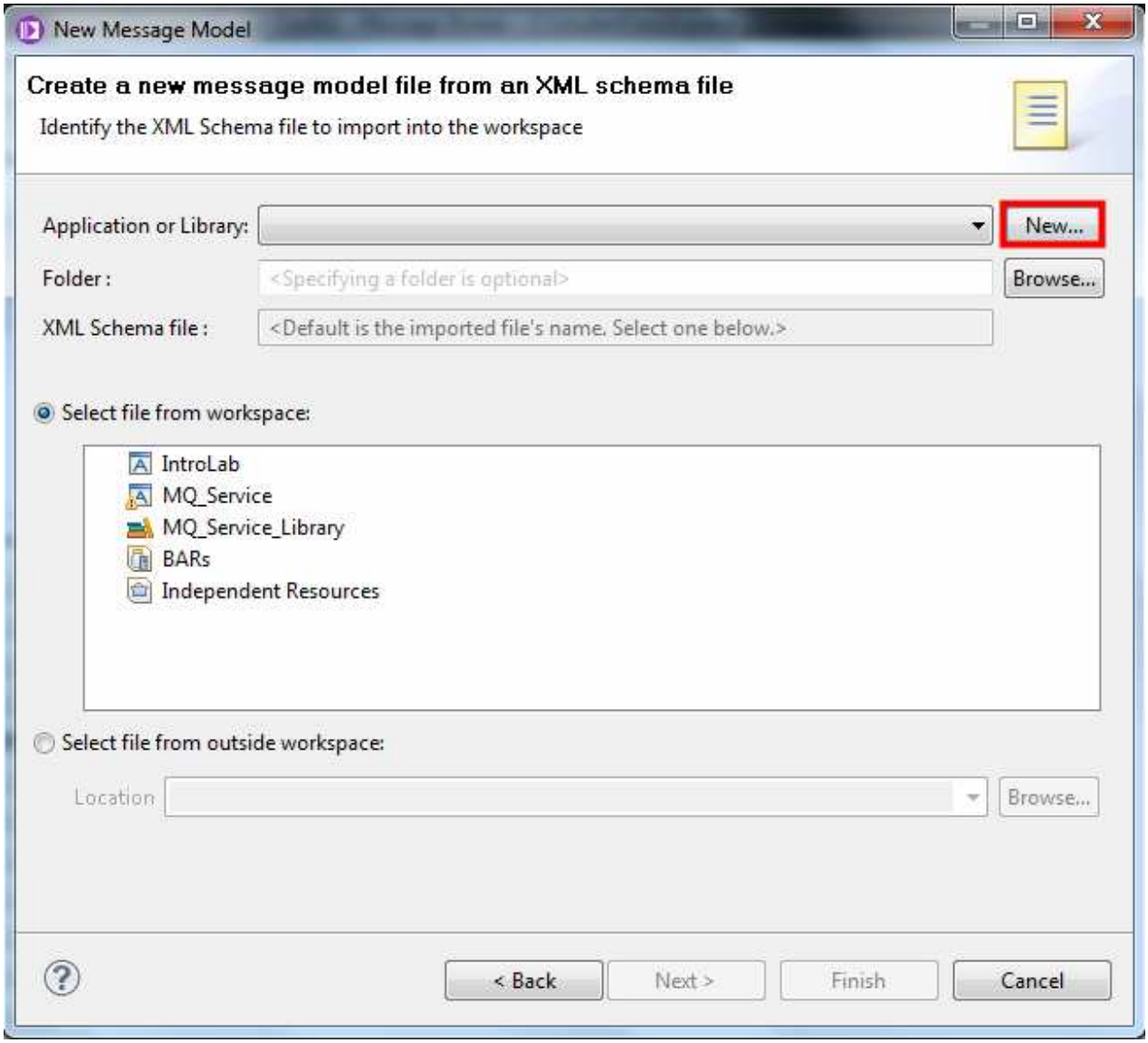


__6. Select the **I already have an XML schema for my data** radio button.

__7. Click **Next**.

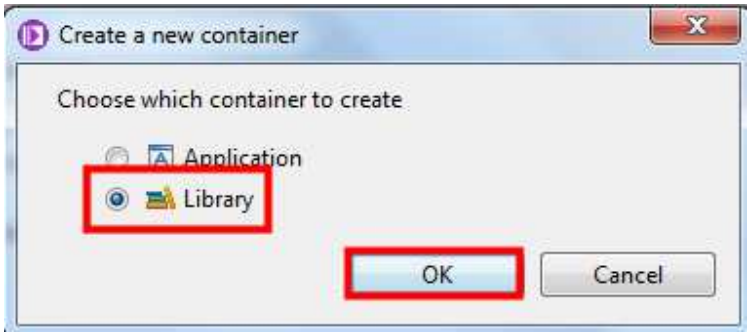


__8. Click the **New..** button next to the **Application or Library** field.



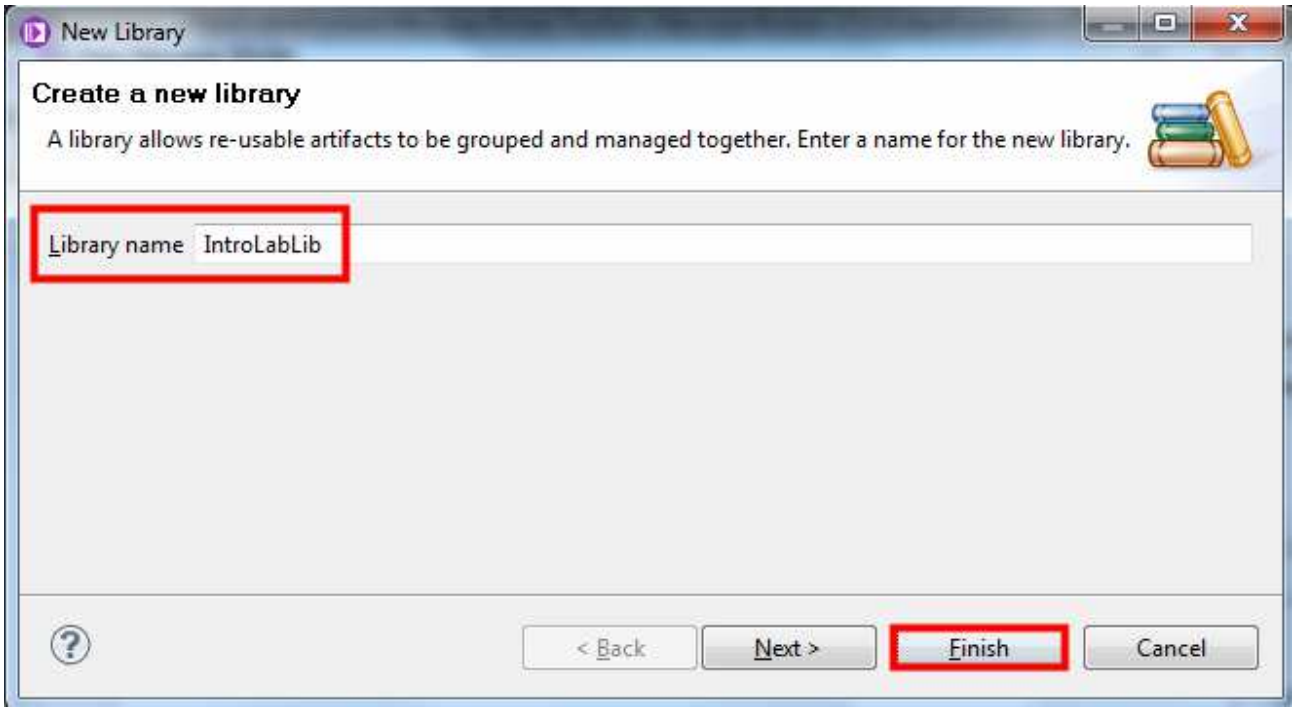
__9. In the popup dialog, select **Library**.

__10. Click **OK**.



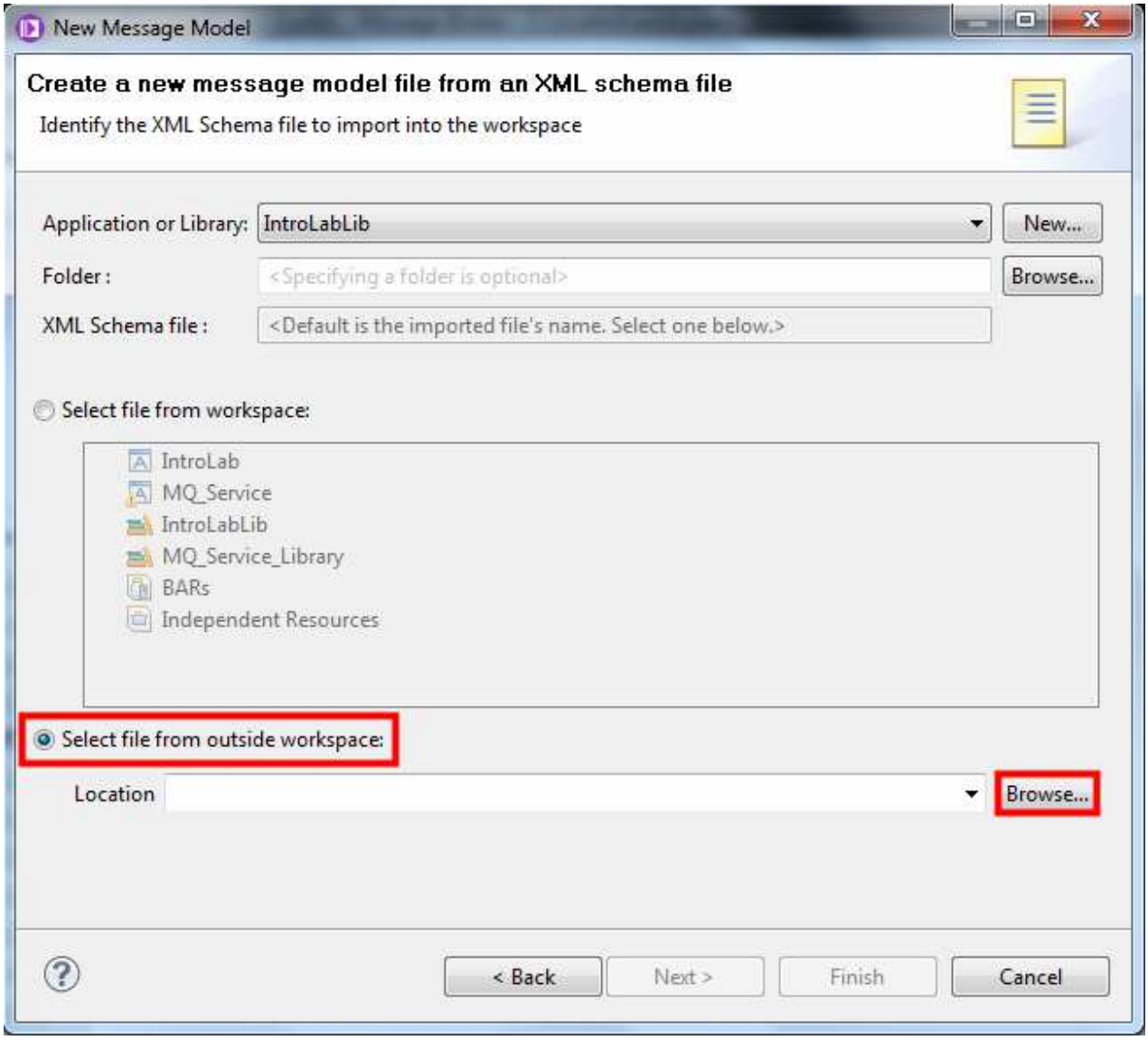
__11. In the popup dialog, type **IntroLabLib** as the **Library name**.

__12. Click **Finish**.



__13. Back in the Message Model wizard, check the radio button **Select file from outside workspace**.

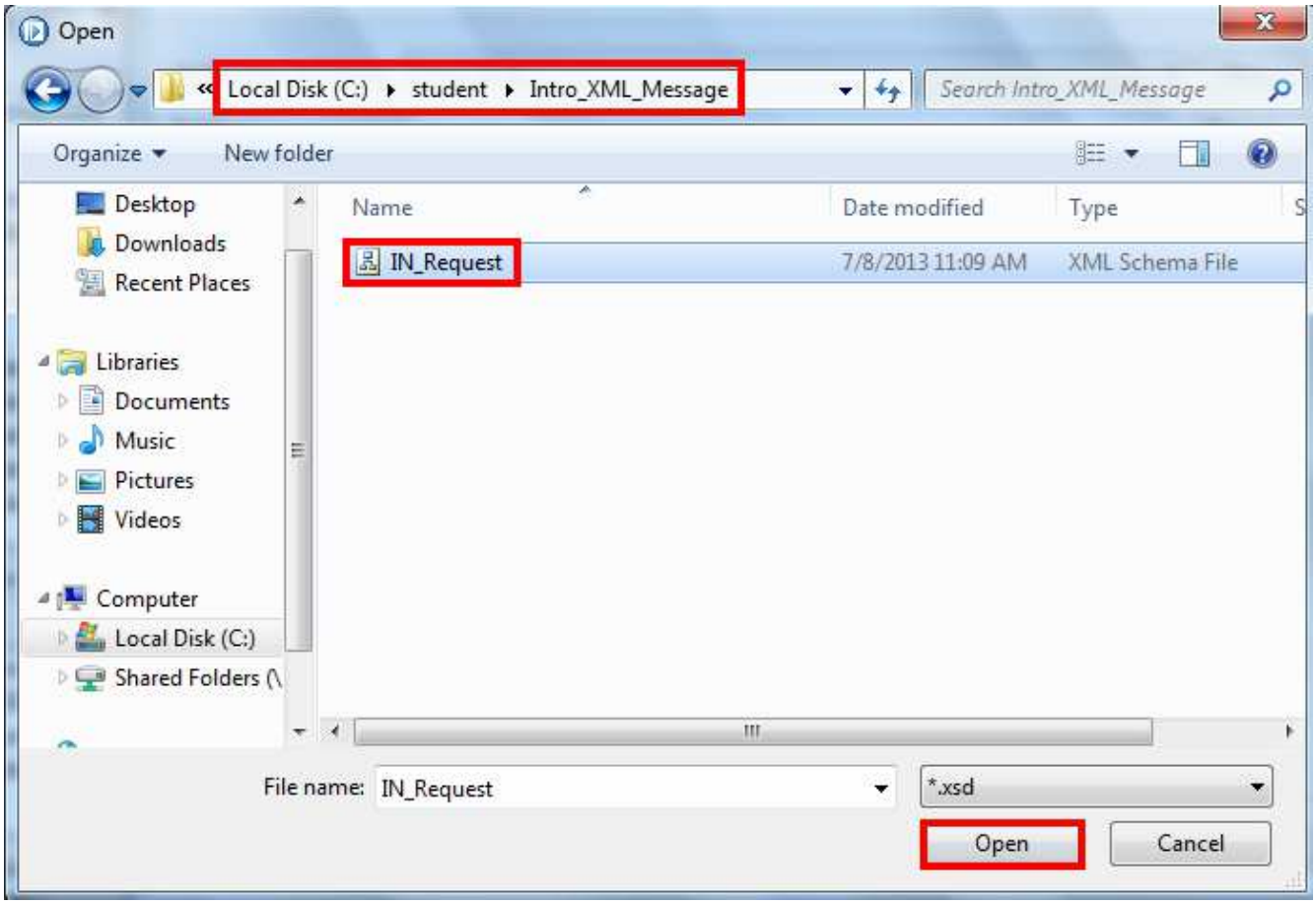
__14. Click **Browse…**



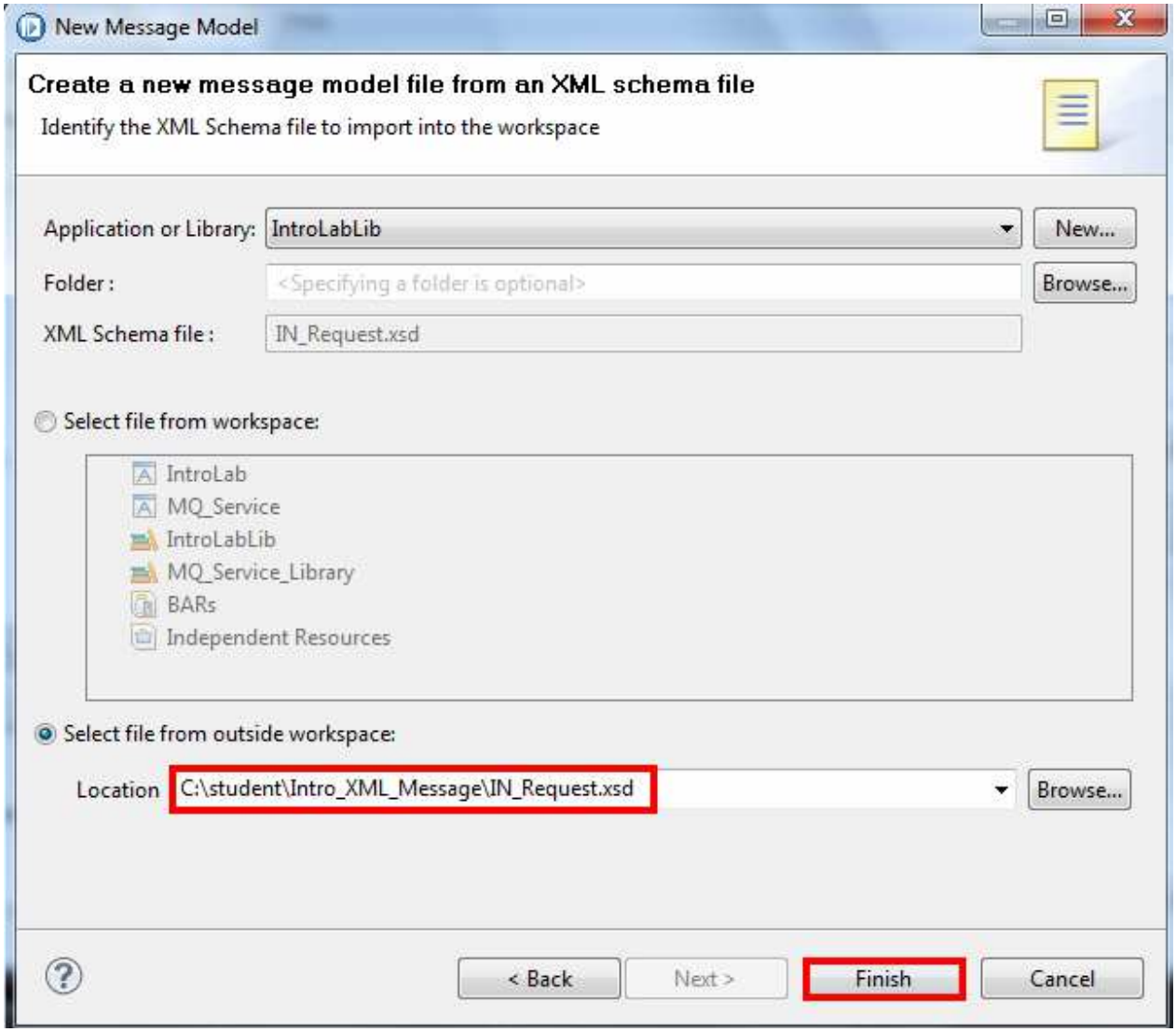
__15. Navigate to **C:\Student\Intro_XML_Message** folder.

__16. Select **IN_Request.xsd**.

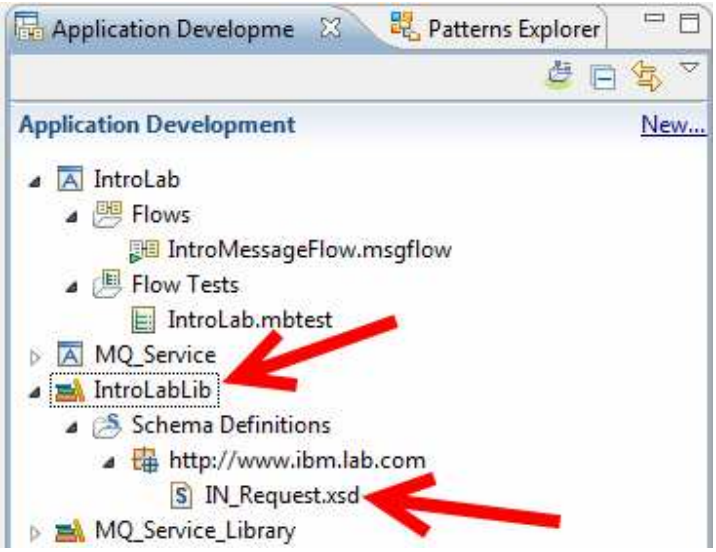
__17. Click **Open**.



__18. Back in the Message Model Wizard, click **Finish.**



You now have a Library project with the XML message model for the input message.



Key Idea: Library Projects

Applications and libraries are deployable containers of resources, such as message flows, message definitions (DFDL, XSD files), JAR files, XSL style sheets, and WebSphere Adapters files.

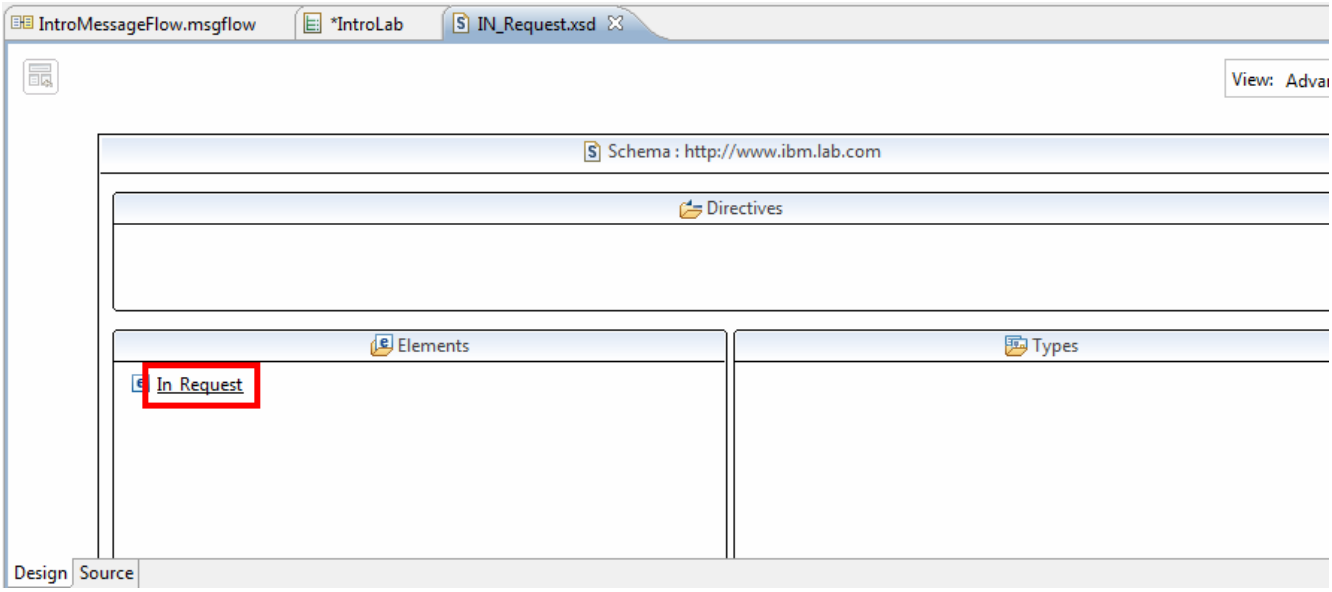
A library is a logical grouping of related code, data, or both. A library contains references to reusable resources, such as a message model or map. A library can refer to a resource that is contained in another library. Libraries are optional. They are typically used to reuse resources. Libraries can be either embedded in an application (private) or obtained by a message flow (that is not part of an application) dynamically at run time (execution group level). Use multiple libraries to group related resources (for example, by type or function).

Consider using libraries if you want to share routines and definitions across multiple teams, projects, or brokers. Libraries are also useful if you need to use different versions of a coherent set of routines and definitions.

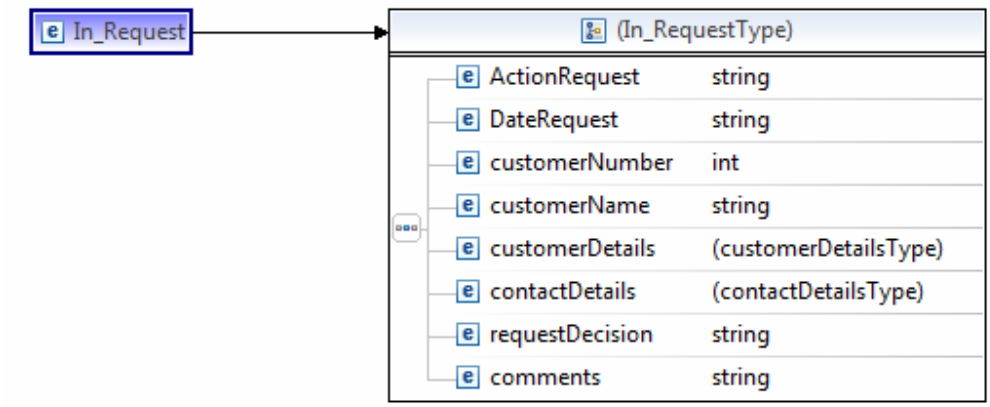
Using a library is typically not necessary if you do not need to regularly reuse IBM Integration Bus routines or definitions.

Notice that the XSD is opened for you after import and is visible using the XML Schema Editor, an editor in the toolkit which shows you both a GUI representation of your XML schema as well as the source. **In_Request** is the only Global element. If you double click on it, you can drill down into its structure.

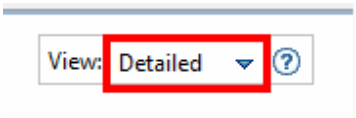
__19. Double click on the **In_Request** element to view the message elements.



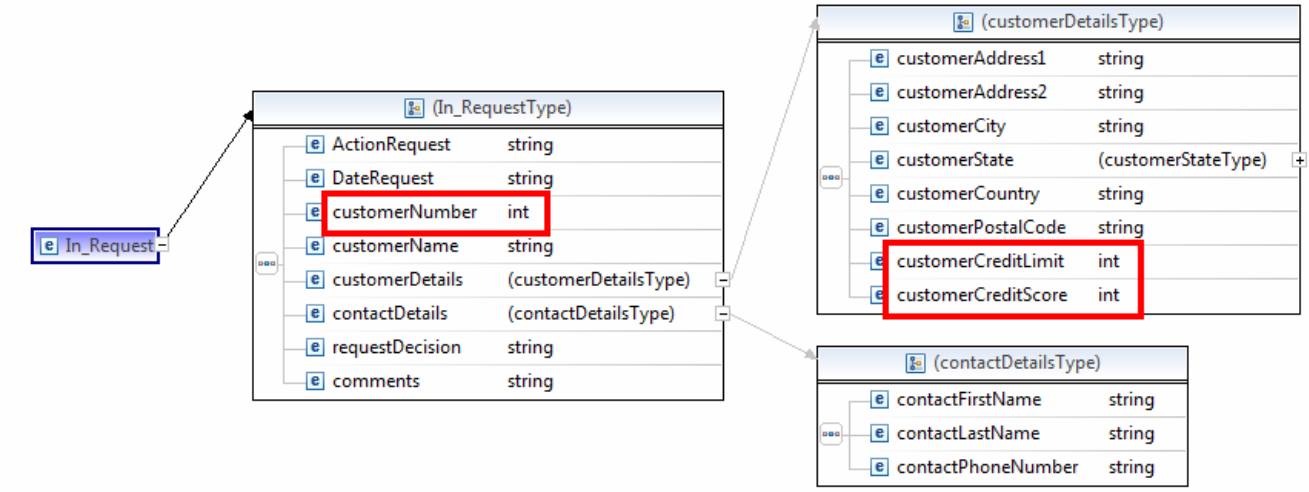
The message model should now be visible.



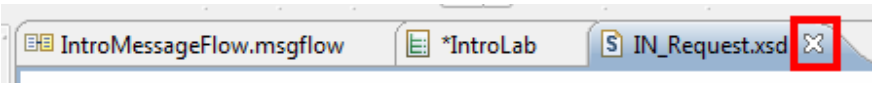
__20. Select the **Detailed** view.



Some elements, such as customerNumber and customerCreditScore, are integers (ints) and not strings.



__21. Close the **IN_Request.xsd** tab.

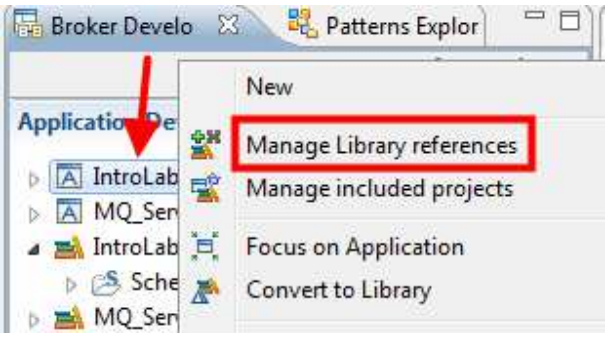


Now, let's update the message flow to use the message model when parsing incoming messages.

__22. In the project view on the left, select the **IntroLab** application.

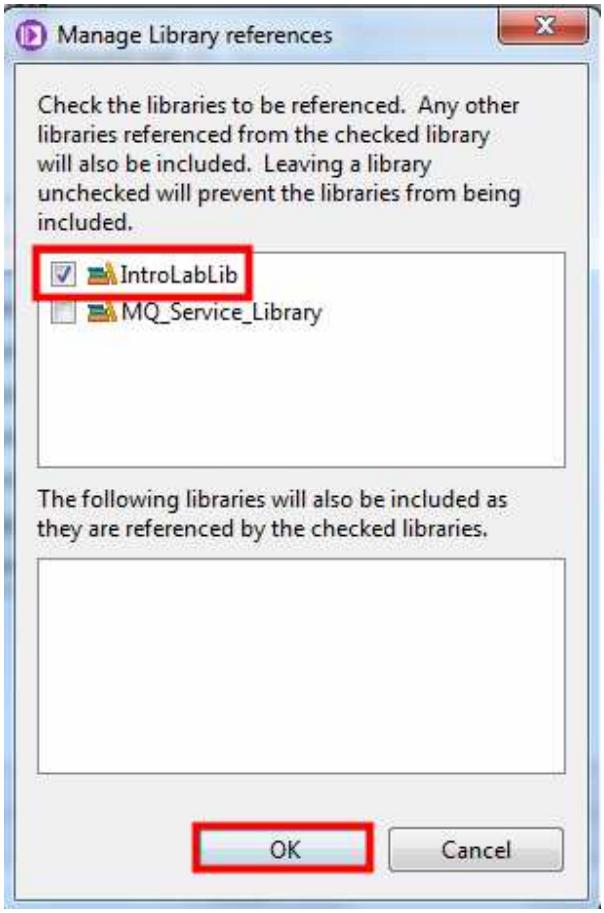
__23. Press the right mouse button.

__24. Select **Manage Library references**.



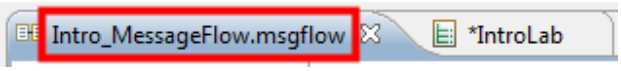
__25. Select the **IntroLabLib** check box.

__26. Click **OK**.



We need to tell the parser to run in *schema-driven* mode, rather than operate in *programmatic* mode.

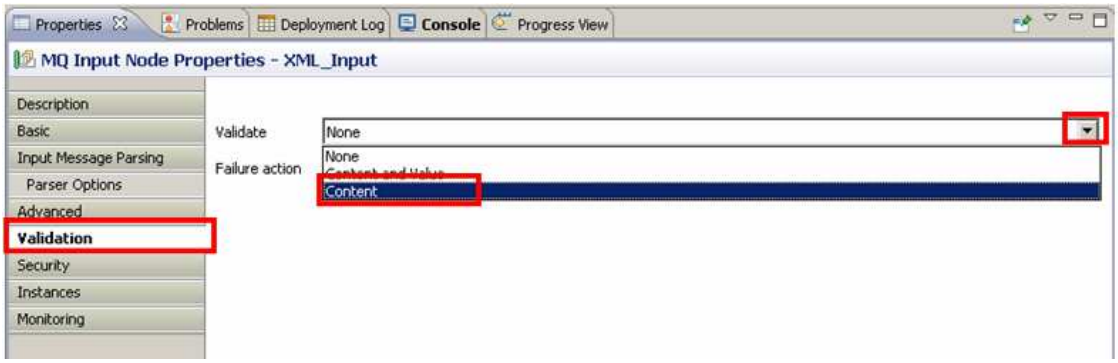
__27. Click on the **Intro_MessageFlow.msgflow** tab to return to the message flow editor.



__28. Single click on the **XML_Input** node, in order to edit its properties.

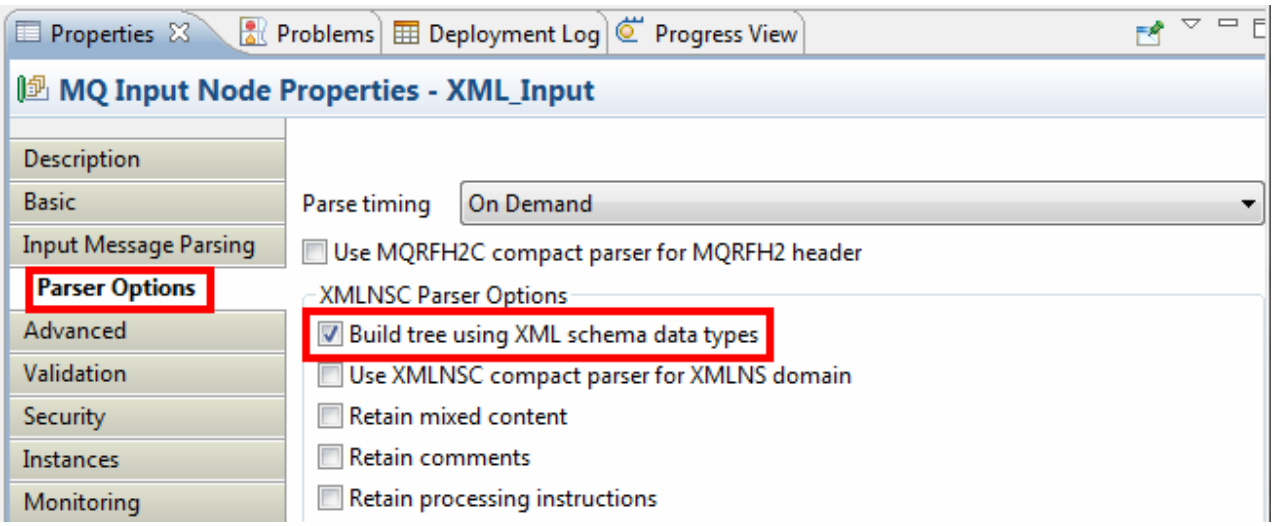
__29. In the **Properties** view, click on the **Validation** tab.

__30. In the **Validation** dropdown, select **Content**.



__31. Select the **Parser Options** tab.

__32. Select the **Build tree using XML schema data types** check box.



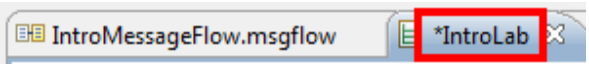
__33. 

Save the message flow (**Ctrl+S**).

2.4 Re-running the Test Client

The flow will now be run again. The trace output will then be examined.

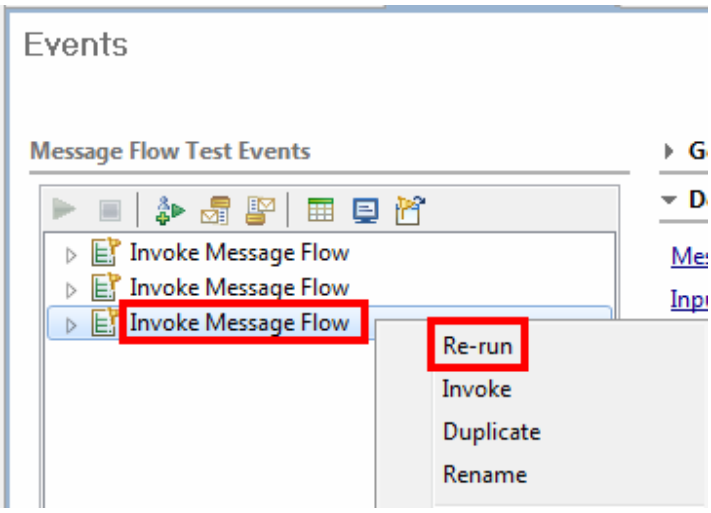
__1. In the editor, select the **IntroLab.mbtest** tab (or re-open from the App in the navigator).



__2. Right click **Invoke Message Flow**.

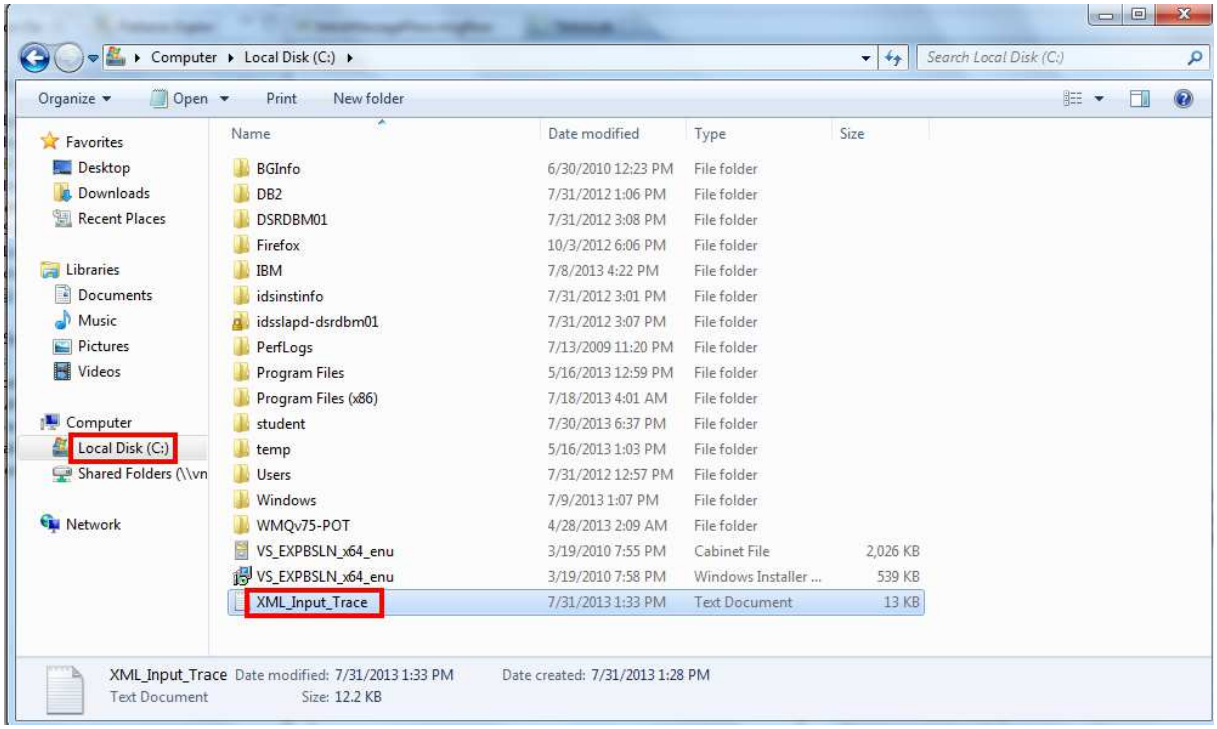
__3. Click **Re-run**.

The tooling will automatically re-build and redeploy the App with the dependent Library included.

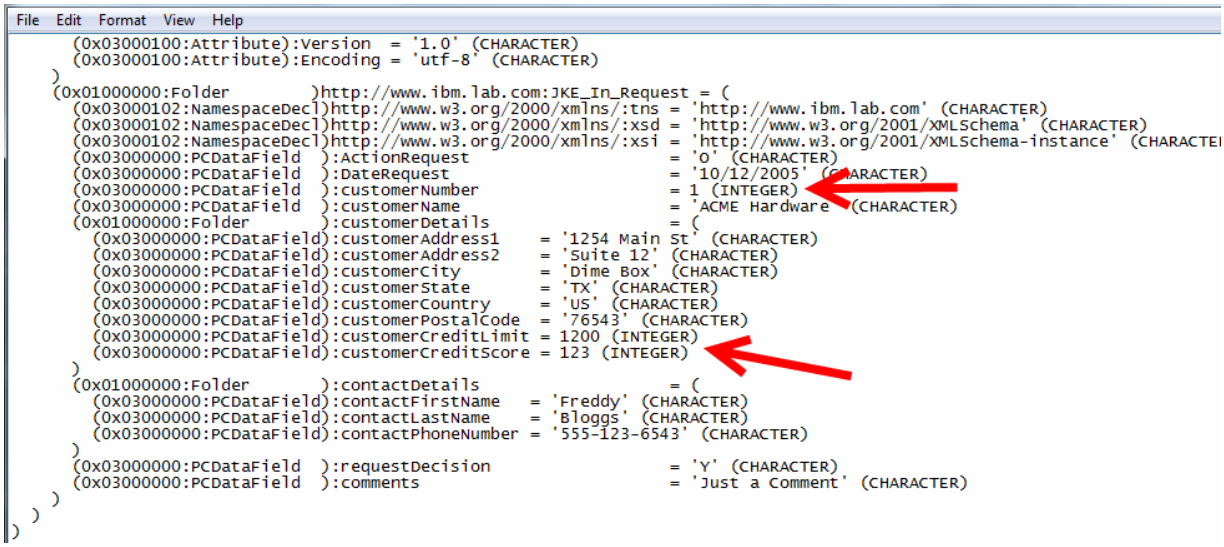


__4. Return to the Windows Explorer window.

__5. Double click the **C:\XML_Input_Trace.txt** file.



__6. Scroll down to the end of the file (or use **Ctl+End**), and view the new trace output.



```
File  Edit  Format  View  Help
      (0x03000100:Attribute):Version  = '1.0' (CHARACTER)
      (0x03000100:Attribute):Encoding = 'utf-8' (CHARACTER)
    )
    (0x01000000:Folder        )http://www.ibm.lab.com:JKE_In_Request = (
      (0x03000102:NamespaceDecl)http://www.w3.org/2000/xmlns/:tns = 'http://www.ibm.lab.com' (CHARACTER)
      (0x03000102:NamespaceDecl)http://www.w3.org/2000/xmlns/:xsd = 'http://www.w3.org/2001/XMLSchema' (CHARACTER)
      (0x03000102:NamespaceDecl)http://www.w3.org/2000/xmlns/:xsi = 'http://www.w3.org/2001/XMLSchema-instance' (CHARACTE
      (0x03000000:PCDataField  ):ActionRequest                    = 'O' (CHARACTER)
      (0x03000000:PCDataField  ):DateRequest                      = '10/12/2005' (CHARACTER)
      (0x03000000:PCDataField  ):customerNumber                   = 1 (INTEGER)
      (0x03000000:PCDataField  ):customerName                     = 'ACME Hardware' (CHARACTER)
      (0x01000000:Folder       ):customerDetails                  = (
        (0x03000000:PCDataField):customerAddress1    = '1254 Main St' (CHARACTER)
        (0x03000000:PCDataField):customerAddress2    = 'Suite 12' (CHARACTER)
        (0x03000000:PCDataField):customerCity        = 'Dime Box' (CHARACTER)
        (0x03000000:PCDataField):customerState       = 'TX' (CHARACTER)
        (0x03000000:PCDataField):customerCountry     = 'US' (CHARACTER)
        (0x03000000:PCDataField):customerPostalCode  = '76543' (CHARACTER)
        (0x03000000:PCDataField):customerCreditLimit = 1200 (INTEGER)
        (0x03000000:PCDataField):customerCreditScore = 123 (INTEGER)
      )
      (0x01000000:Folder       ):contactDetails                   = (
        (0x03000000:PCDataField):contactFirstName   = 'Freddy' (CHARACTER)
        (0x03000000:PCDataField):contactLastName    = 'Bloggs' (CHARACTER)
        (0x03000000:PCDataField):contactPhoneNumber = '555-123-6543' (CHARACTER)
      )
      (0x03000000:PCDataField  ):requestDecision                  = 'Y' (CHARACTER)
      (0x03000000:PCDataField  ):comments                         = 'Just a Comment' (CHARACTER)
    )
  )
)
```

__7. Close the Notepad window.

__8. Minimize Windows Explorer.

This is the end of lab 2.