IBM Cloud

An IBM Proof of Technology

Discovering the value of IBM App Connect Enterprise v11

Lab exercises

An IBM Proof of Technology

Contents

Icons

The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

Lab 1 Getting started

1.1 Building and executing a simple Integration Application

In this lab, you will build and execute an Integration *Application*, consisting of a single *message flow*. The Application will then be deployed to an *Integration Server* in an *Integration Node,* where it will execute.

To explain these terms, in order:

- **An *Application* is a container that holds all the artifacts that make up an Integration Solution.**
- **A *message flow* is like a program but is developed using a visual paradigm.**
- **An *Integration Server* is an operating system process where user flows execute.**
- **An *Integration Node* is a collection of operating system processes, including one or more Integration Servers, that together act as an Integration Control Point. This is sometimes referred to as a Broker.**

The Integration Application you build will be deployed to an Integration Server, where it will execute. The unit of deployment is a *Broker Archive* (BAR) file. Broker Archive files have a file extension of ".bar", and are essentially a .zip file with a deployment descriptor. The deployment descriptor allows certain properties of the message flow to be overridden. Once deployed, the flow will then be available to process requests, messages, events, and more.

There is no need to restart the Integration Node or the Integration Server for the deployment of a new or changed message flow.

Broker Archive (BAR) files and their use are explained in more detail at the end of Lab 2.

As a convention for these labs, a **red box** will be used to identify a particular area, and when information is to be entered or an action is to be taken, it will be in **bold** characters. **Red arrows or lines** may be used to indicate where nodes are to be placed when building your message flow.

Tools that you will use as part of these labs include:

- ACE Toolkit
- Flow Exerciser
- XPath Expression Builder

A number of sections will be labeled "Key Idea" or "Key Concept". Be sure to review these sections when you come across them, as they will explain concepts of App Connect Enterprise (ACE) that you will explore in some detail in the early labs, but in more detail as you continue to work through the more advanced labs.

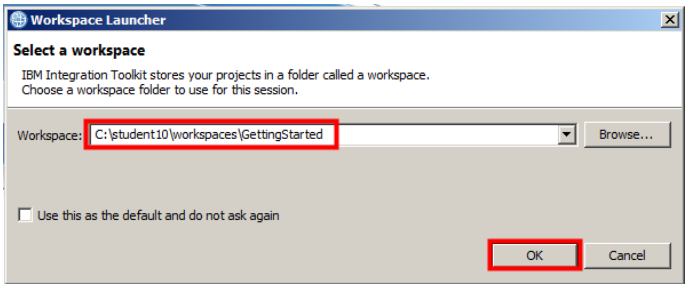
1.2 Getting to know the IBM ACE Toolkit

__1. To start the IBM ACE Toolkit, click the appropriate icon on the desktop:

You are prompted to select an Eclipse Workspace. Select the default here, but be aware that this facility allows you to switch between workspaces when starting or using the toolkit.

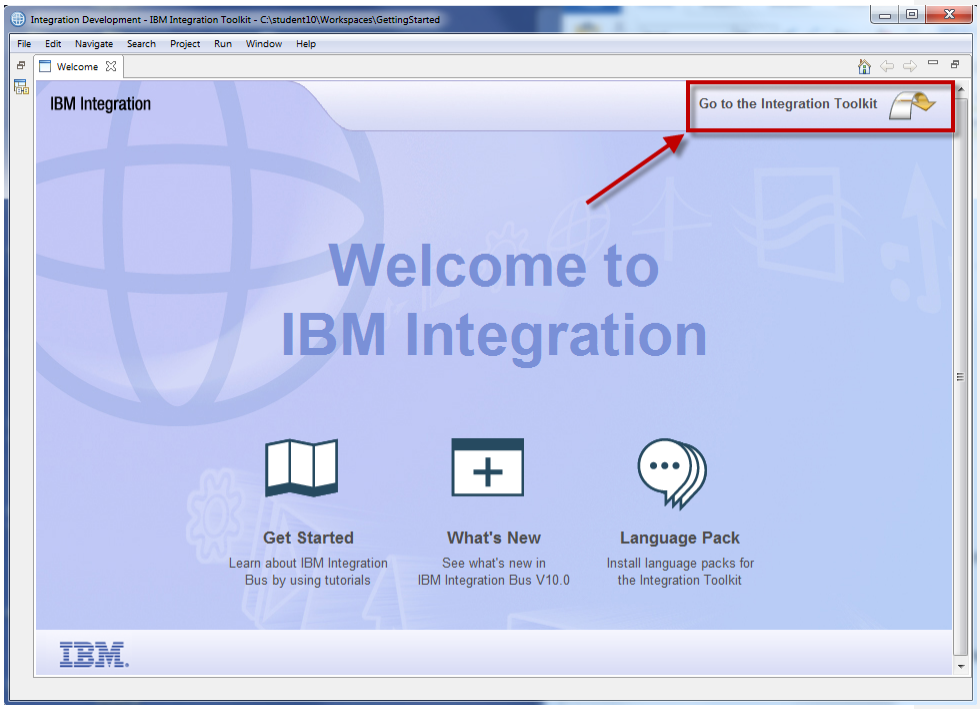
__2. Enter **C:\student10\workspaces\GettingStarted** , or some other directory which is more appropriate for you.

Click **OK**.

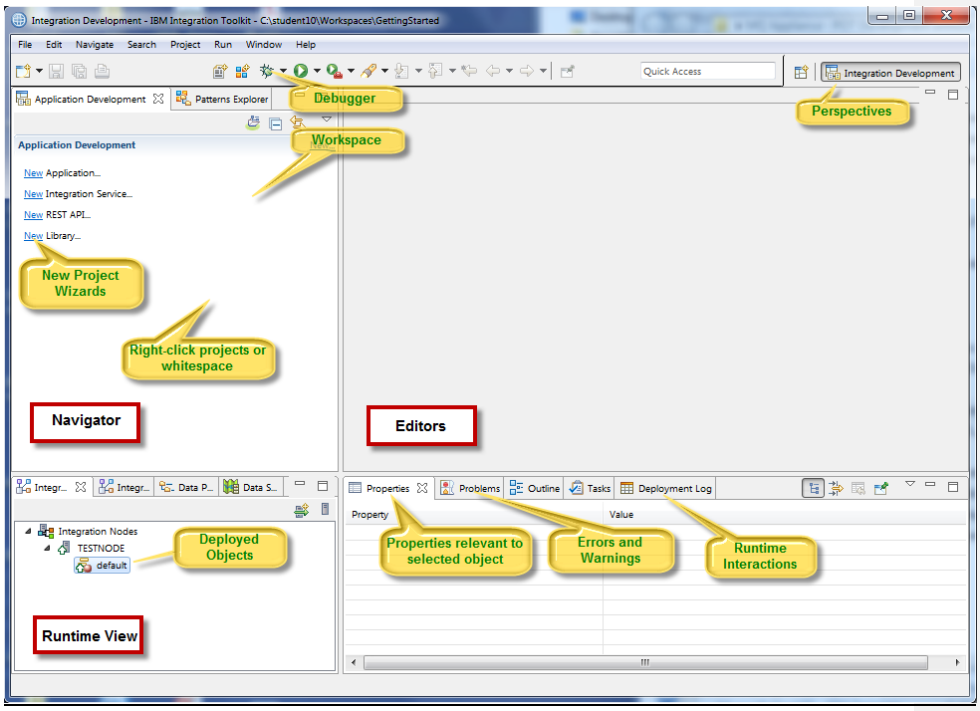


__3. If this workspace is being opened for the first time, you will see the Welcome screen.

Click **Go to the ACE Toolkit** to dismiss this screen.



__4. Now familiarize yourself with aspects of the Toolkit.



1.2.1 Key Idea: The Toolkit

The ACE Toolkit.
Read the section below for additional information on the IBM ACE Toolkit.

The ACE Toolkit is based on Eclipse and includes components from IBM Rational® Application Developer. It provides several perspectives specifically for App Connect Enterprise as well as additional perspectives from Rational Application Developer and Eclipse. This system is using the default installation. During the labs and lectures, you will be learning more about the components in a typical development and runtime environment.

The screenshot above is of the Integration Development perspective. It is divided into multiple views (or panes). Each view is identified by a tab at the top of the view. On the lower left are several additional views, such as the Integration Nodes view.

On the upper left is the Navigator view, which has tabs for projects (Application Development) and patterns (Patterns Explorer). The Navigator view contains the projects that are available within the Eclipse workspace. In this screenshot there are no projects at present; so what you see are a set of New Project wizards, for kick-starting the development of *Applications*, *Integration Services*, *REST APIs* and *Libraries*. You can also access these by right-clicking in the white space of the Application Development view.

The area below the Navigator view is the summary area. Several views are present in this region—the Integration Nodes view will show all defined local nodes as well as connections to remote nodes that have been created. Integration Servers and the resources deployed, as well as their status, will be shown here as well (the myaceworkdir integration server is shown in the screenshot above).

The large area on the right is used by the resource editors. When an editor has opened a resource, it will also be represented by a tab. Below the editor view is a pair of views for Properties and Problems.

At upper right are tabs for the one or more Perspectives that have been opened. To switch between Perspectives, you can simply click the desired tab.

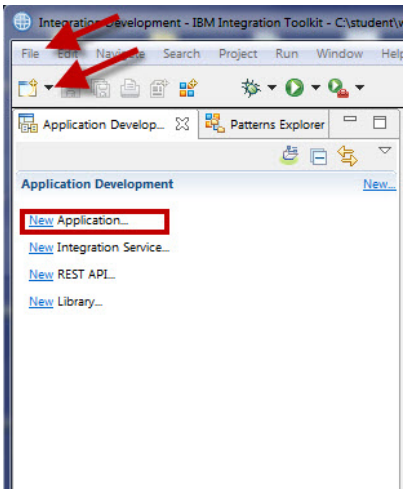
The ACE Toolkit also provides links to a number of predefined patterns. In later labs you will explore how these are used.

Eclipse is project oriented; artifacts are organized into projects. A project is typed. Project types that are specific to App Connect Enterprise include Applications, Integration Services, REST APIs and Libraries. Also, legacy projects of type Message Flow Project and Message Set Project can be created or imported into the toolkit. Since they predate the concept of Applications and Libraries, they will be visible in the hierarchy in a folder called “Independent Resources”.

1.3 Building a simple application

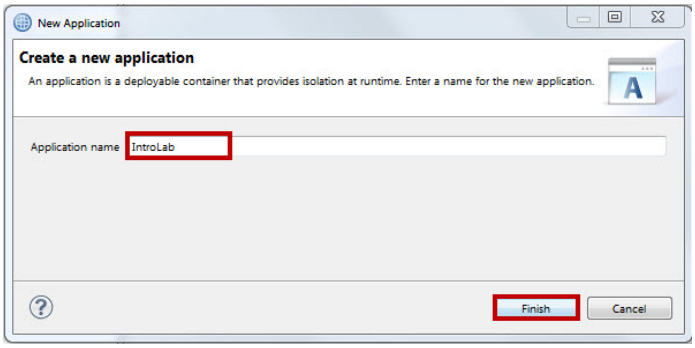
__5. In the Application Development pane, click the **New** **Application** link.

Alternatively, you can click **File** on the menu bar, and select **New > Application**, or use the drop-down list under File. Use these when you want to create project types other than those represented by the four wizards.



You are prompted to enter a name for your Application.

__6. Enter **IntroLab** for the Application name. Click **Finish**.

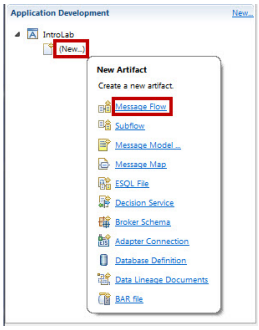


Now you will create a new message flow.

__7. Under **IntroLab**, click **(New)**.

__8. Select **Message Flow**.

The options for the New action will be different depending on how the request is made. For example, when using **File > New** from the Menu bar, all of the options will be listed. However, if you start from an Application, as you did in this case, the only options shown are those that are related to the selected project type.

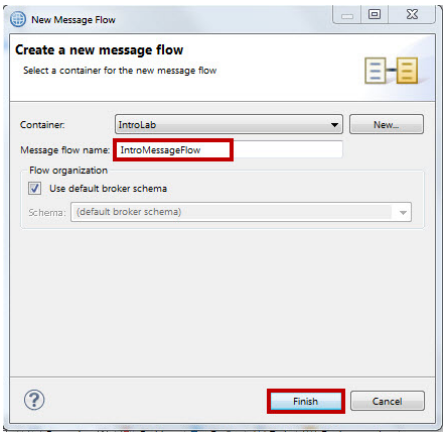


__9. Here you are asked to name the message flow.

Note: An **Application** may contain multiple message flows.

__10. Enter **IntroMessageFlow** in the Message Flow name box.

Click **Finish**.



You should see the first of several editors you will use. This one is the Message Flow Editor, which is used for message flow composition.

__11. Click the white space in the Message Flow Editor. Information about the message flow will appear in the Properties pane.

__12. Select the **Properties** tab.

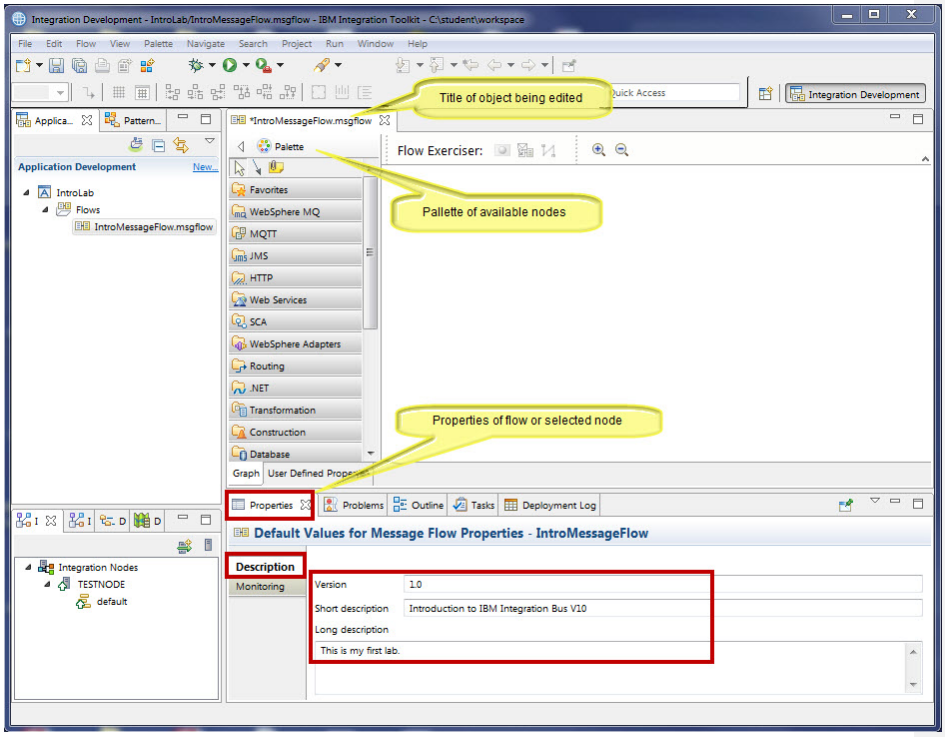
__13. Select the **Description** tab.

__14. Enter the following:

Enter **1.0** for the **Version** field.

Enter **Introduction to App Connect Enterprise V11** for the **Short description** field.

Enter your choice of information in the **Long description** field.



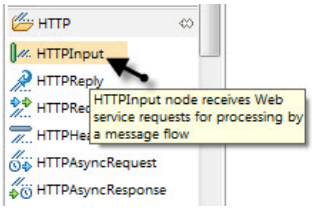
A message flow must begin with an input node. An input node establishes the runtime context for the flow. There is an input node for each of the many protocols that App Connect Enterprise supports. You will process an HTTP request with this flow so you need an **HTTPInput** node.

The **HTTPInput** node is in the **HTTP** drawer.

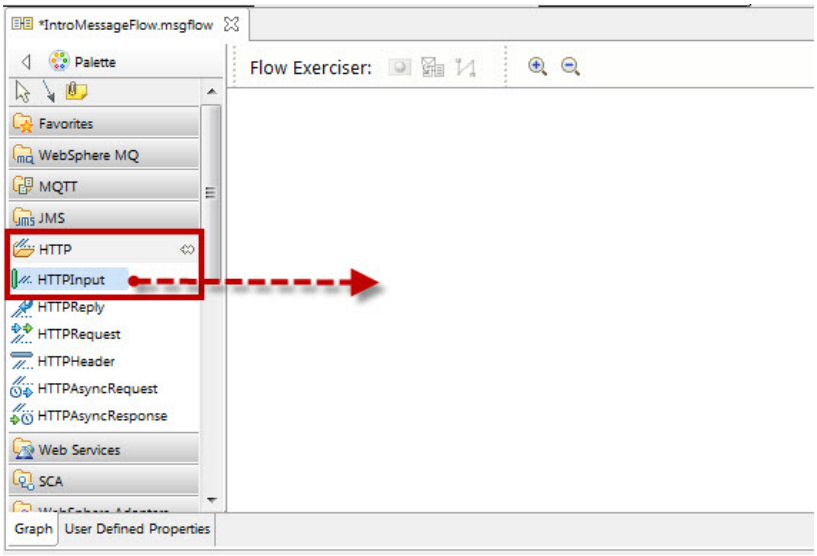
__15. Click on the 

drawer to open it.

__16. Hover over the HTTPInput node. You will see a brief description of the node's function.



__17. Click the HTTPInput node, and either drag it to the canvas or move the mouse to the canvas and click again.



When a node is initially added, its name can be changed immediately by typing over the default name or by entering a new value in the node name field in the Description tab of the Properties.

A best practice is to provide a new name for each node that is descriptive of the function that it provides. For most of the labs, you will be renaming the nodes.

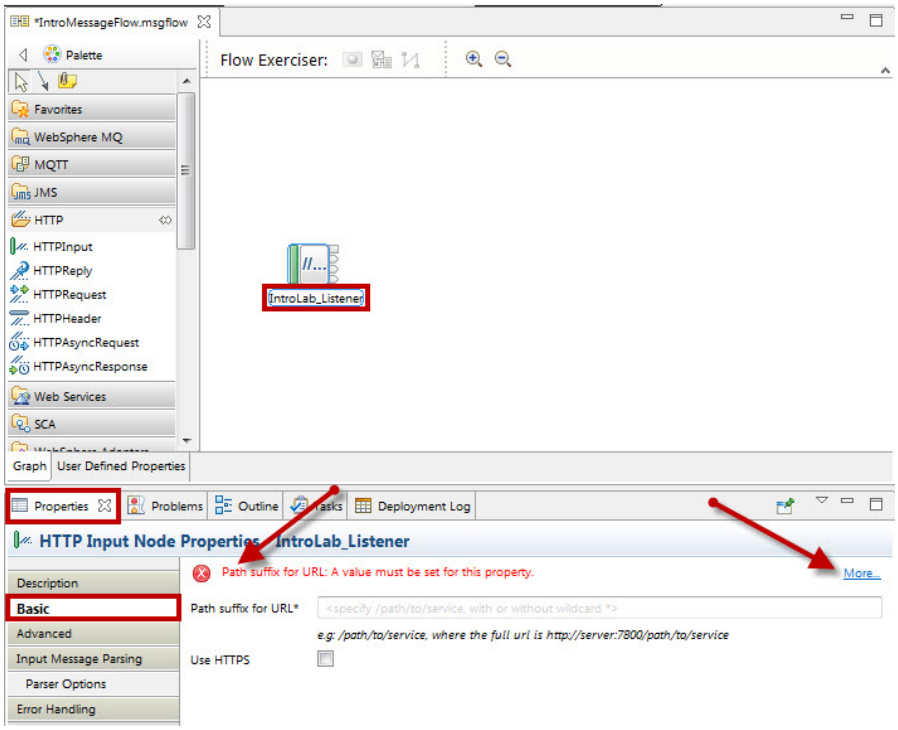
Note:
If you use the names as suggested it will make it easier to follow this lab guide.

__18. Change the name of the node to “IntroLab_Listener.”

Press **Enter** to complete the rename.

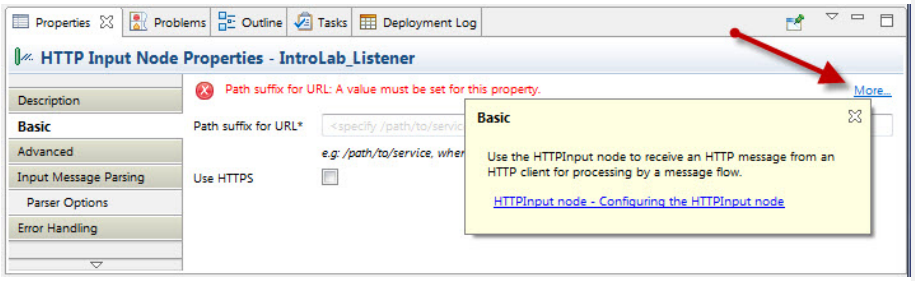
Select the **Basic** tab in the **Properties** pane.

The **Path suffix for URL** in the node properties must be set. Mandatory fields that are not set are indicated by a message in **red**.



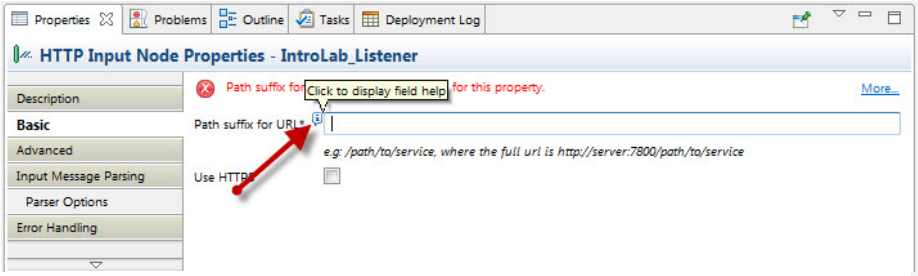
__19. Note the More... hotspot on the right.

Click this for a brief description of the node's function. In addition, there is a hyperlink provided to the Help section that describes the node in detail.



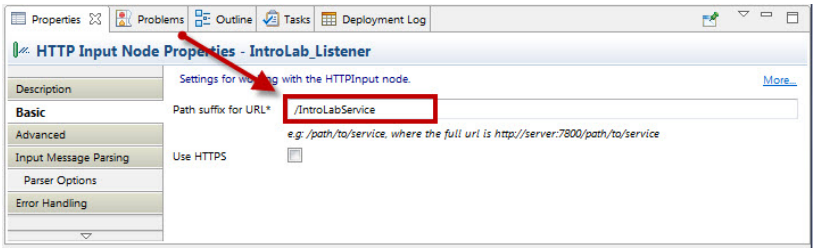
__20. Click in the **Path suffix for URL** field.

__21. Hover over the icon that appears. Field-level help is available by clicking on these when they appear.



__22. Enter **/IntroLabService** as the **Path suffix for URL**.

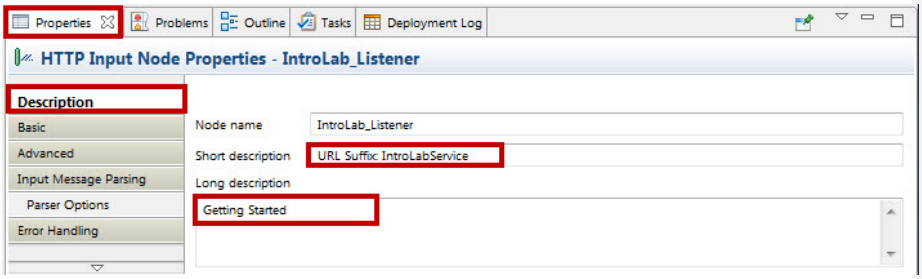
Note:
This name is case sensitive.
The path suffix must begin with "/".



__23. Select the **Description** tab. This section is used for documentation. The name of the node may also be changed. The node name is shown in the message flow editor. It does not affect the operation of the message flow.

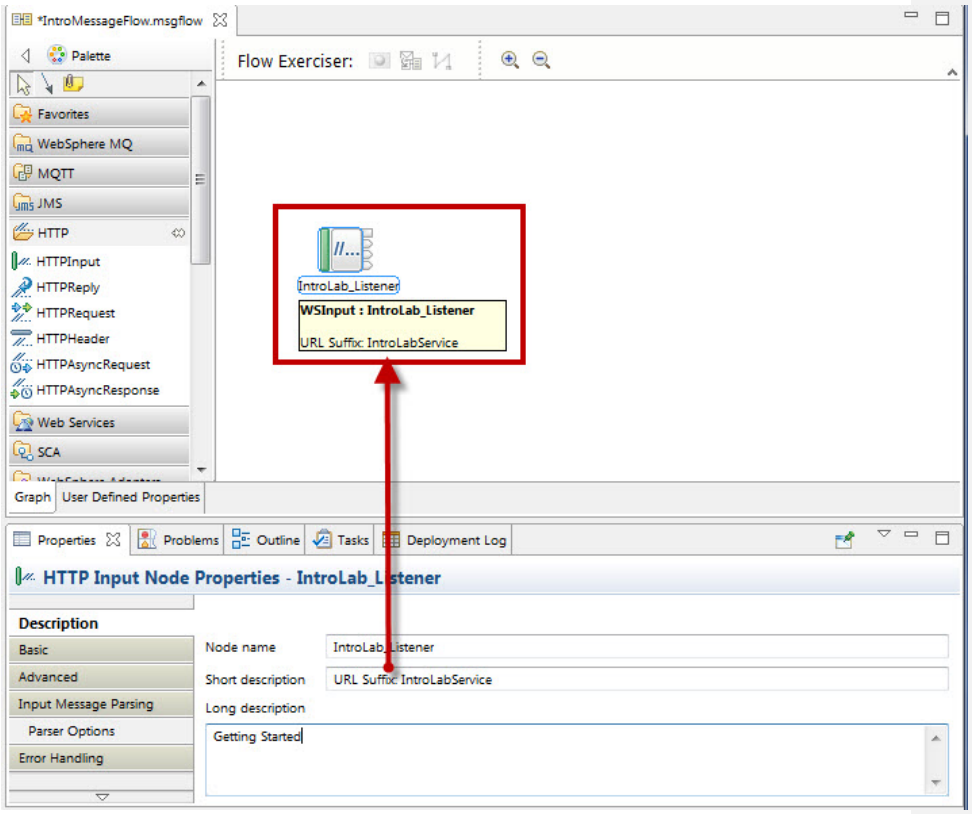
__24. Enter **URL Suffix: IntroLabService** in the **Short description** field.

__25. Enter your choice of text for the **Long description** (**Getting Started** is shown in the screen shot).



__26. In the flow editor, hover the mouse pointer over the node name.

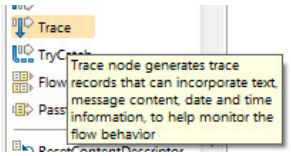
The information in the Short description field is displayed. When there are multiple nodes on the canvas, if you move from node to node with the mouse, the same tab in the Properties will be displayed.



__27. The **Trace** node is in the **Construction** drawer.

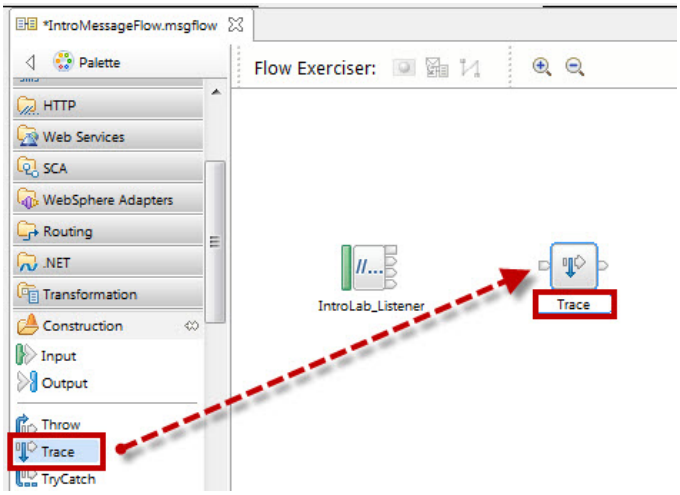
Click the Construction drawer to open it.

Hover over the Trace node to see a brief description of the node's function.



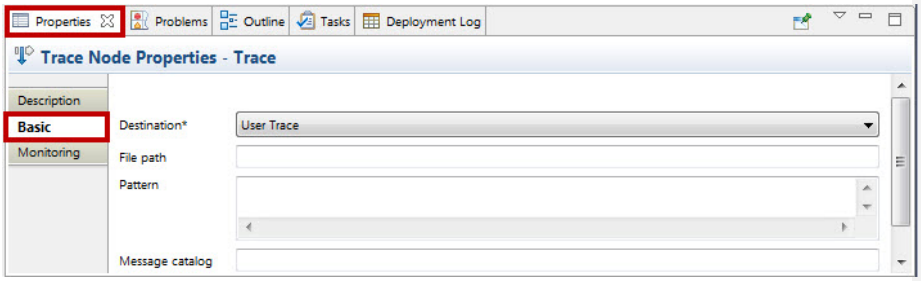
__28. Select the Trace node and place it on the canvas to the right of the **IntroLab_Listener** node. You do not need to rename the Trace node.

__29. Press **Enter** to accept the default node name (**Trace**).

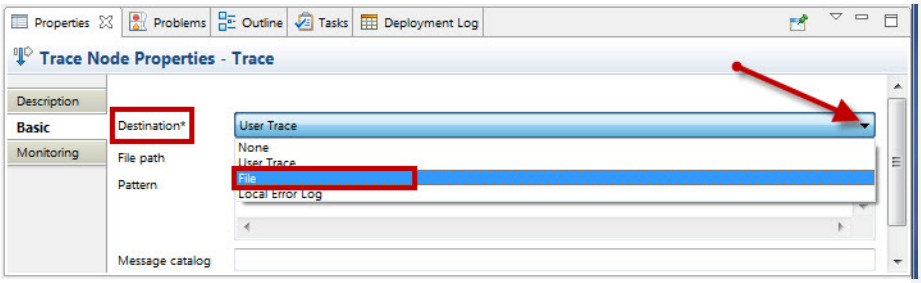


__30. Click the **Trace** node.

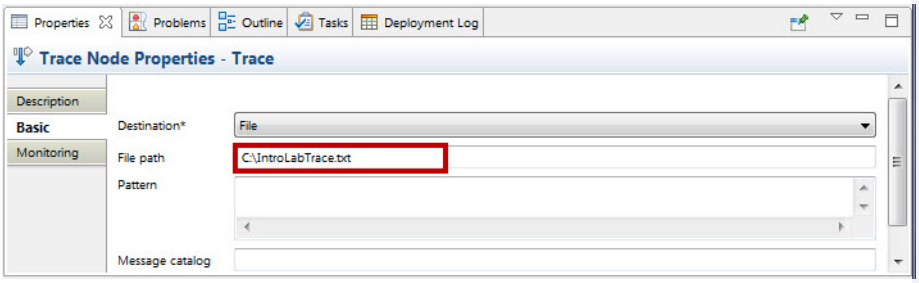
__31. In Properties, click the **Basic** tab.



__32. Use the pull down list on the **Destination** field to select **File**.



__33. In the File Path field, enter **C:\IntroLabTrace.txt(_or location of your choice, just remember it)**,.



The information in the **Pattern** box tells the node what information to produce in the trace output. If you type a line of raw text, it is echoed to the output.

__34. Enter a line of ========== in the Pattern box.

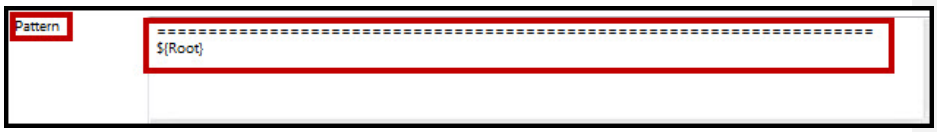
Note: You can actually enter any free-form text you like. You are doing this to make it easier to separate trace entries in the trace file, but this is not a requirement.

In the **Pattern** box, enter the string **${Root}** exactly as indicated – this tells the trace node to dump out the entire contents of the message syntax tree that enters the node.

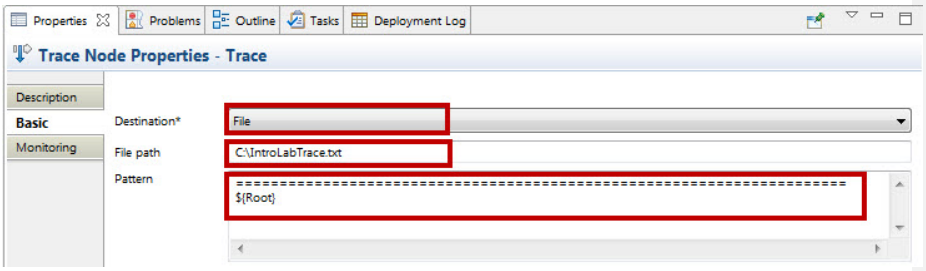
Note:
The pattern uses **curly braces**, not parentheses!
The pattern is case sensitive!

The expression between the curly braces will be evaluated at run time.

Review the values you entered into the **Pattern** box. Does it match what you see below?



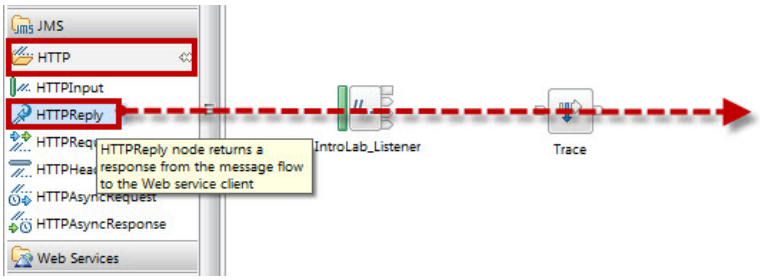
__35. Double-check that **Destination** is set to **File**, the **File path** and name are valid (for example, there are no embedded spaces), and that the **Pattern** is specified exactly as **${Root}**



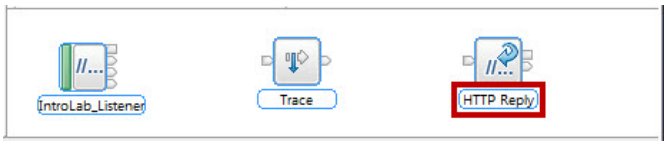
__36. On the Palette, return to the **HTTP** drawer and open it.

__37. Select an **HTTPReply** node from the drawer.

__38. Place it to the right of the **Trace** node.

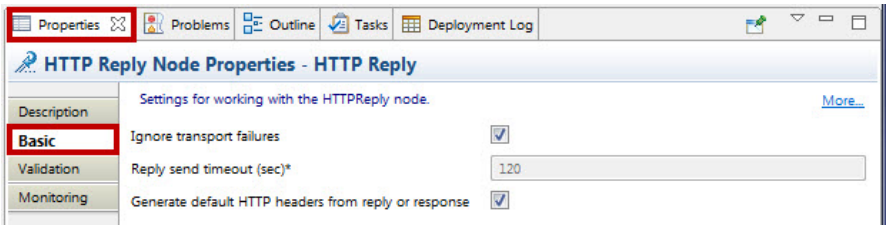


__39. Press **Enter** to accept the default node name of HTTP Reply.



__40. Click the HTTPReply node.

__41. Review the properties for this node. You will be using the default values.



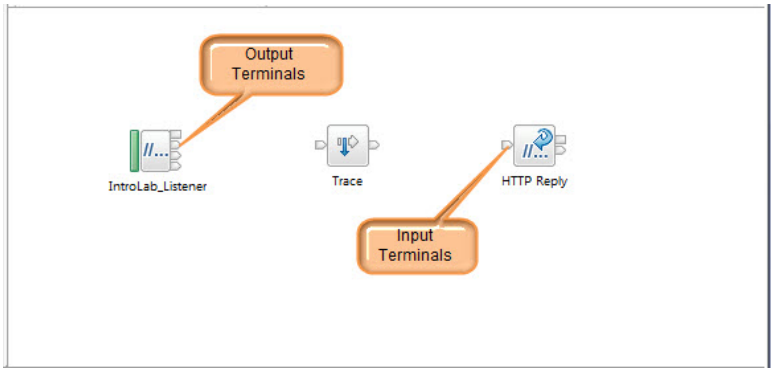
The nodes for this flow are configured. Next you will wire the flow together using the Node Terminals.

1.3.1 Key Concept: Node Terminals

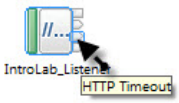
.

Key Concept: Node Terminals.
Read the section below for additional information on Node Terminals and their use.

As you work with the various nodes, you will also be working with their Input and Output terminals. Input terminals are typically named **In**. Most nodes have an Output terminal named **Out**. They may have several others.



You can hover the mouse pointer over the node terminals to see their names.



Some of these will have common names such as **Failure** or **Catch** and others will be unique to that particular node. Some nodes allow you to define the terminals. The terminals are given a name when they are defined.

The lab instructions will identify the output terminal to be used when connecting nodes together. If you hover the mouse pointer over a terminal, a small popup will appear that identifies the name of the terminal.

You will now wire the nodes together to create a path of control for the message to follow through the message flow.

Wire the Out terminal of the **IntroLab_Listener** node to the In terminal of the **Trace** node. There are two techniques that can accomplish this:

One – position the mouse over the **Out** terminal (second from the top), click and drag to the target and click again.

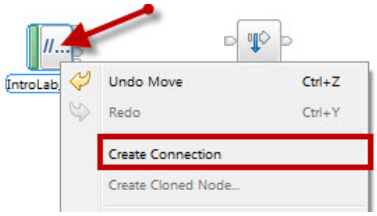
Two – right-click a node and select **Create Connection**. What follows is an example of terminal selection presented as a result of using Create Connection.

The rest of the Lab instructions show the second method for wiring.

Commented [EL1]: please confirm that this change is correct

__42. Right-click the **IntroLab_Listener** node.

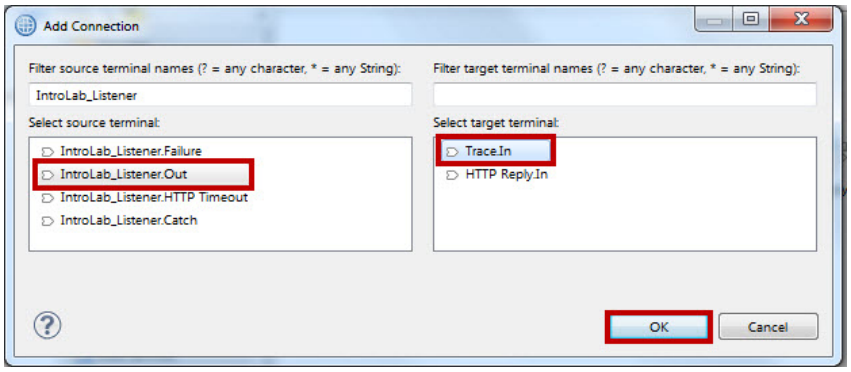
Select **Create Connection** from the menu.



__43. The **Add Connection** selection box is displayed.

The list of the available output terminals for the selected node is shown on the left.

A list of available input terminals for the nodes in the flow is shown on the right.

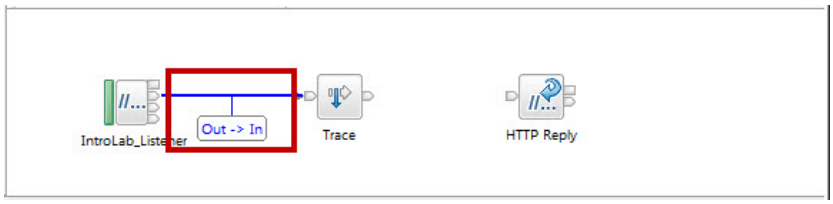


__44. Select **IntroLab_Listener.Out** on the left.

__45. Select **Trace.In** on the right.

__46. Click **OK**.

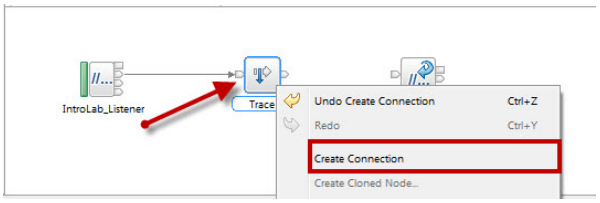
The connection will be created.



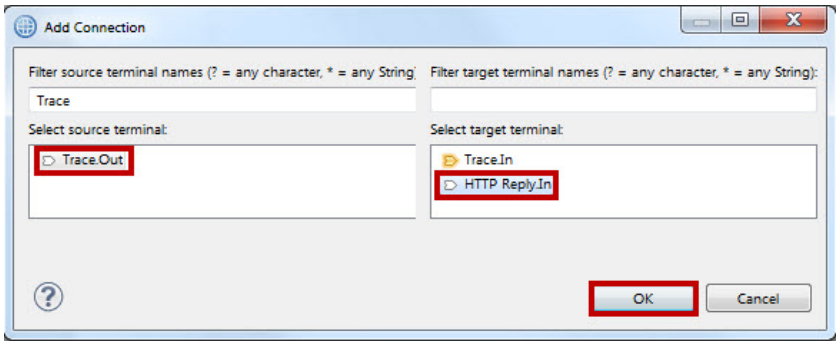
The same steps will be used to make a connection from the **Trace** node to the **HTTP Reply** node.

__47. Right-click the **Trace** node.

__48. Select **Create Connection**.



The **Add Connection** selection box is displayed.

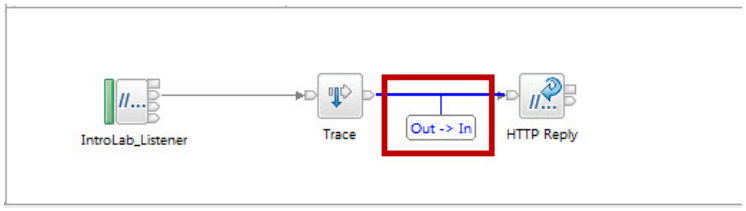


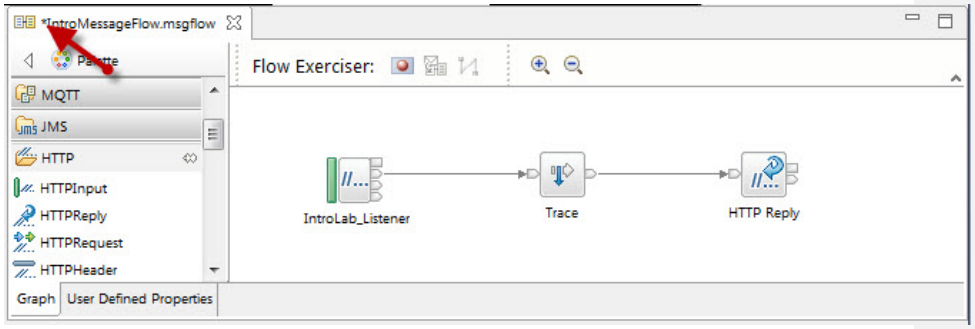
__49. Select **Trace.Out** on the left.

__50. Select **HTTP Reply.In** on the right.

__51. Click **OK**.

The connection will be created.





Your message flow should now look like the above diagram.

Notice the small asterisk next to the message flow name. This indicates that the message flow has not been saved to disk.

__52. It is time to save your work. Hold down the **Ctrl** key and press the **S** key to save the message flow. You can also click the “diskette” icon or use **File** > **Save** to perform a save.

The following graphic will be used as a reminder when it is time to save your work. 

1.4 Test the flow using the Flow Exerciser

The message flow is now complete. The next step is to test it. The integrated Flow Exerciser will be used to test the message flow.

1.4.1 Key Concept: The Flow Exerciser

.

Key Concept: The Flow Exerciser
Read the section below for additional information on the Flow Exerciser.

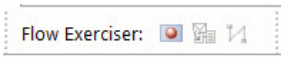
The App Connect Enterprise Flow Exerciser provides the ability to capture a "snapshot" of the message assembly as it is processed from node to node in a flow. These snapshots capture the state of the message, as well as other structures in the message assembly, highlighting the path of control through the message flow, providing an easy, visual way to test new or modified flows.

The recorded snapshot includes all syntax trees in the message assembly—message, local environment, environment and the exception list—as well as several other properties such as a sequence number that give more context to the recorded message.

When using the Flow Exerciser, recorded messages are displayed using an XML representation, which shows the structure and content of the logical tree at each point in the message flow. This provides a convenient visualization of the logical tree, and is a simplified version of the internal representation that ACE uses to represent the logical tree.

Fields in the message tree can be modified, providing an easy way to exercise different paths in the message flow.

A Flow Exerciser toolbar is located at the top of the flow editor. It has several icons, as described below.



 Start the Flow Exerciser. Once started, clicking this icon again will stop it.

 Create a message (or select an existing message or recorded message) to send through the flow.

 View the path a message takes through the flow.

Using this toolbar, message recording and display can be performed with only a handful of mouse clicks. This toolbar is activated when the user clicks the button to put the flow into record mode. This will deploy the message flow, along with any dependent resources, and enable recording. Once the message flow has been invoked, either by using an external client or injecting a message, the recorded message assembly at each stage in the flow can be displayed.

Message injection is a feature of the Flow Exerciser that provides the ability to "inject" recorded messages back into a flow, bypassing their normal transport method. For example, messages can be injected in the MQTT subscribe node even when an MQTT server is unavailable.

When recorded messages are displayed, those that are valid for injection include a "Save" icon in the top-right corner. Messages are saved to the same project as the message flow and can be shared with a project interchange file or backed up to a repository for use in a test environment.

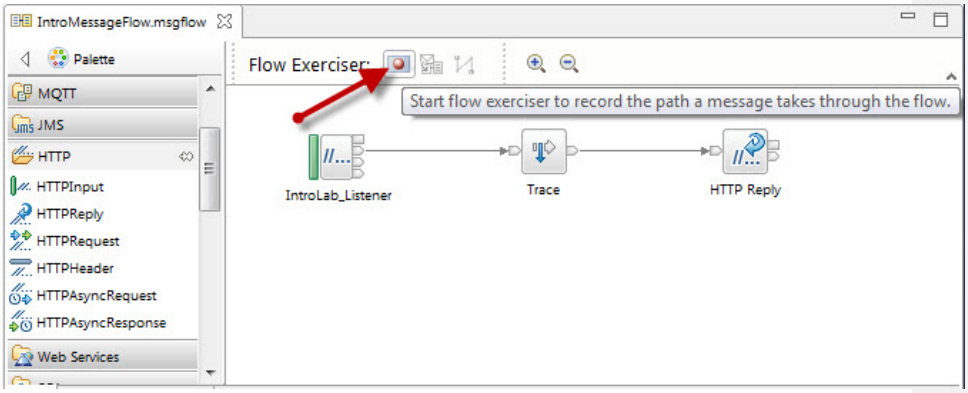
Message injection is supported on the following nodes:

- MQInput
- MQTTSubscribe
- SOAPInput
- HTTPInput
- FileInput

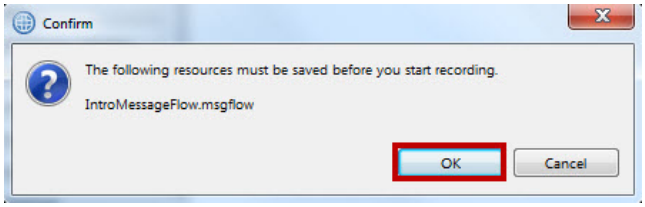
Input nodes will only process injected messages if they have no incoming data via their regular transport. Output nodes "know" that a message has been injected and will deal with it differently from a normal message.

In this section, you will use the Flow Exerciser to test your message flow.

__53. Click the **Start** control on the Flow Exerciser toolbar to start recording.

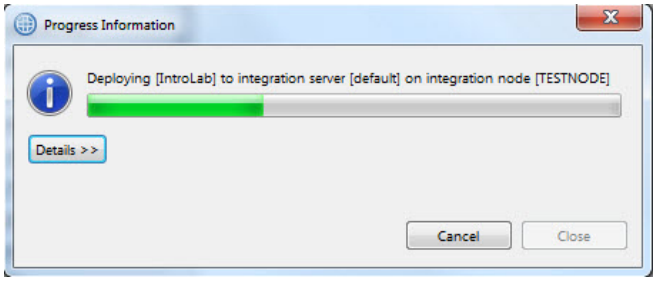


__54. If you did not save the flow before starting the Flow Exerciser, the following window will be displayed:



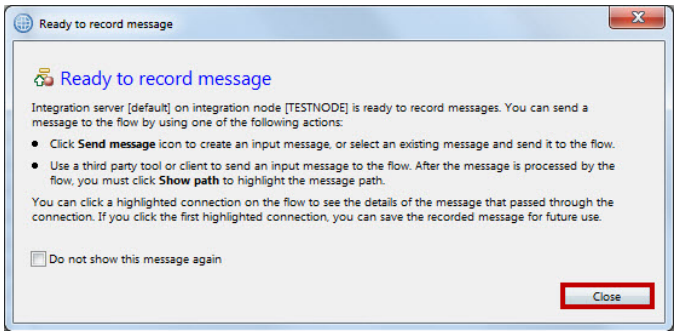
Click **OK**.

__55. The application must be deployed to the **myaceworkdir** integration server. This will be done automatically for you by the Flow Exerciser:



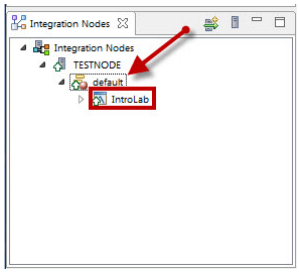
__56. Once the application has been deployed, the following window will be displayed.

Review the information, and when ready, click **Close**.

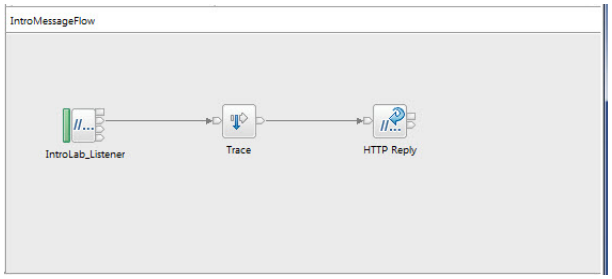


__57. You are now ready to start recording. You will notice two changes:

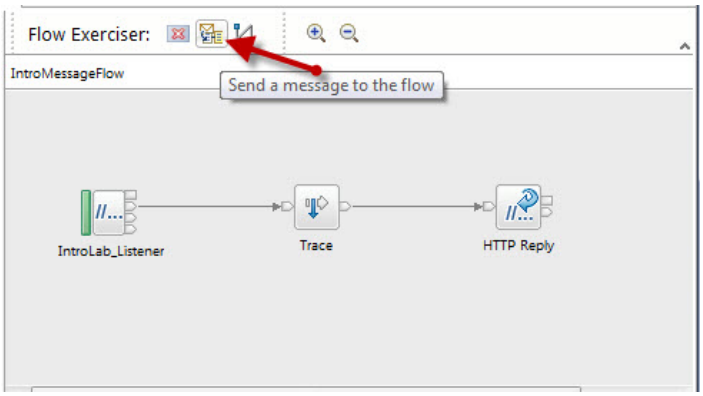
1) The application has been deployed to the **myaceworkdir** integration server. You can see this by looking at the Integration Nodes view at the bottom left of the Toolkit:



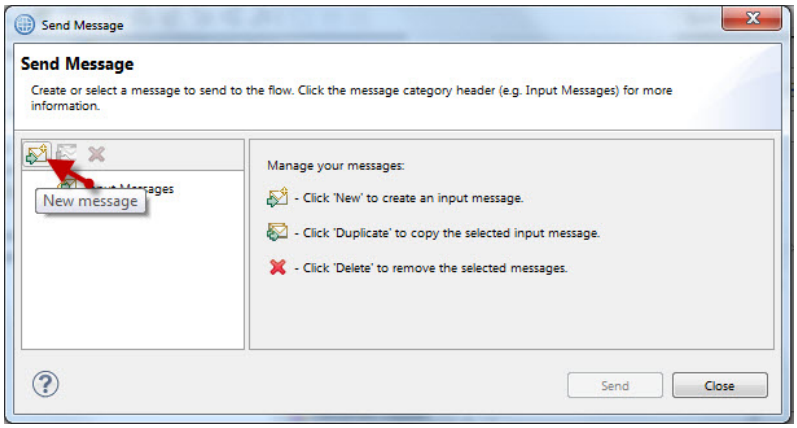
2) The message flow is now “locked” – the background color of the flow editor has changed, and you cannot modify the flow:



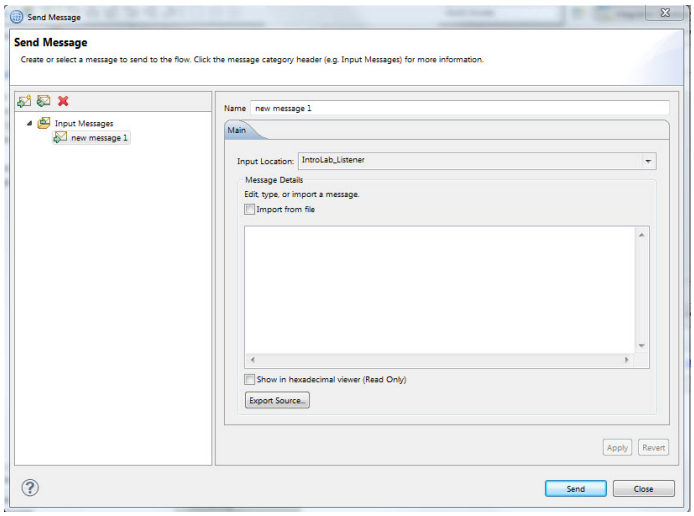
__58. To select a message to send, click the **Send Message** control on the Flow Exerciser toolbar.



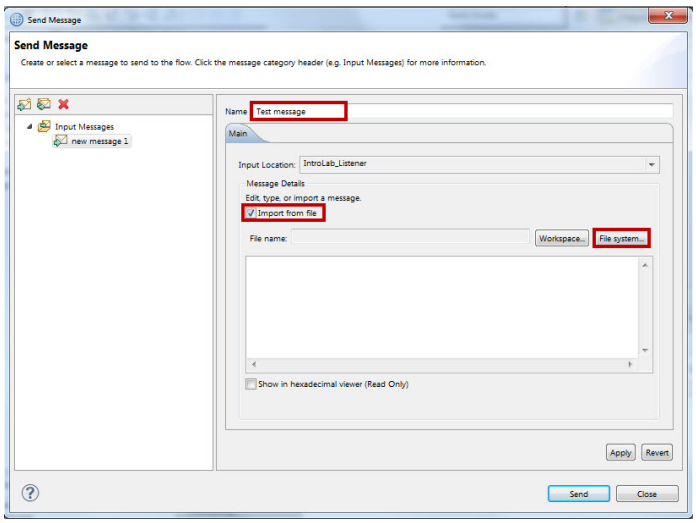
__59. In the **Send Message** dialog, click the **New message** control.



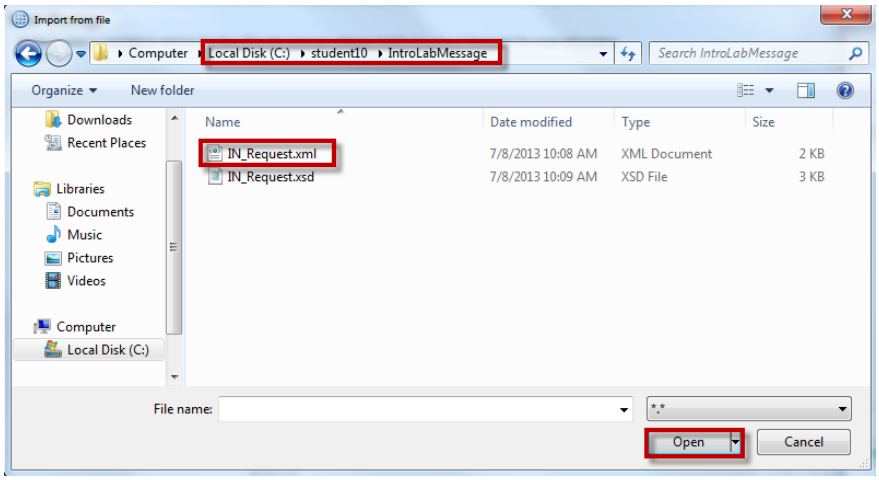
__60. A new message (by default called **new message 1**) will be created.



__61. Change the name to “Test message,” select **Import from file**, and click **File system**.

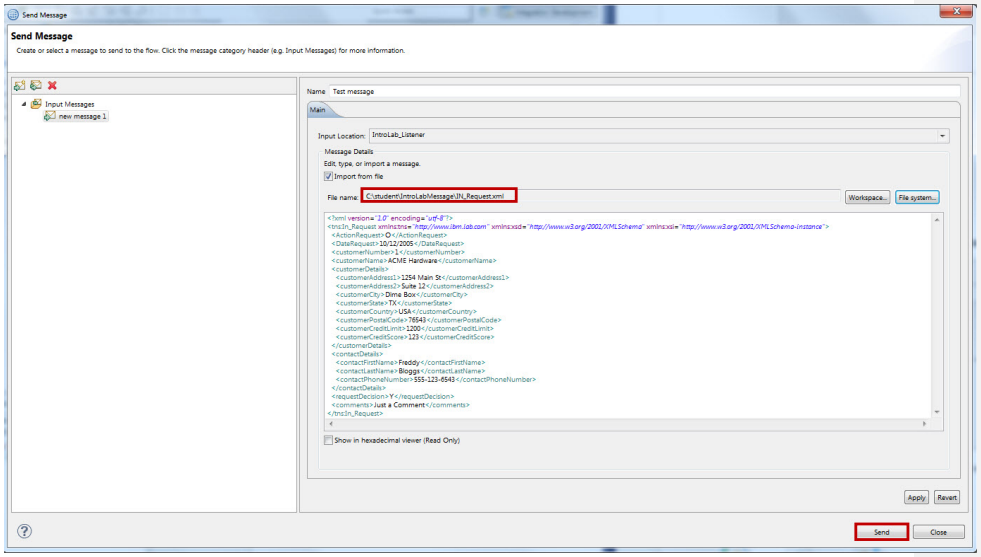


__62. In the Import from file dialog, navigate to **C:\student\IntroLabMessage**, select the **IN_Request.xml** file, and click **Open**.



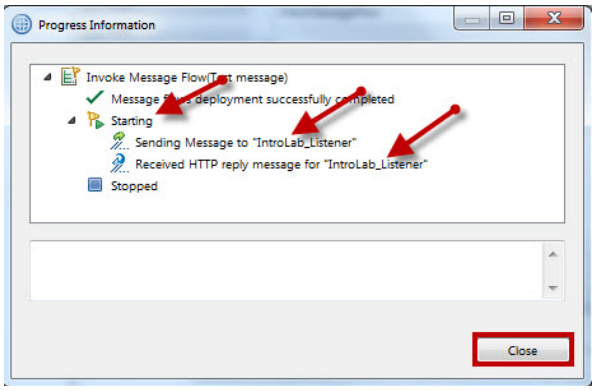
__63. The selected file will be loaded.

Click **Send** to inject the message into the message flow.



__64. The Flow Exerciser will start the flow, inject the message into the flow, and capture the HTTP reply message for viewing.

Click **Close** when you are ready to proceed.



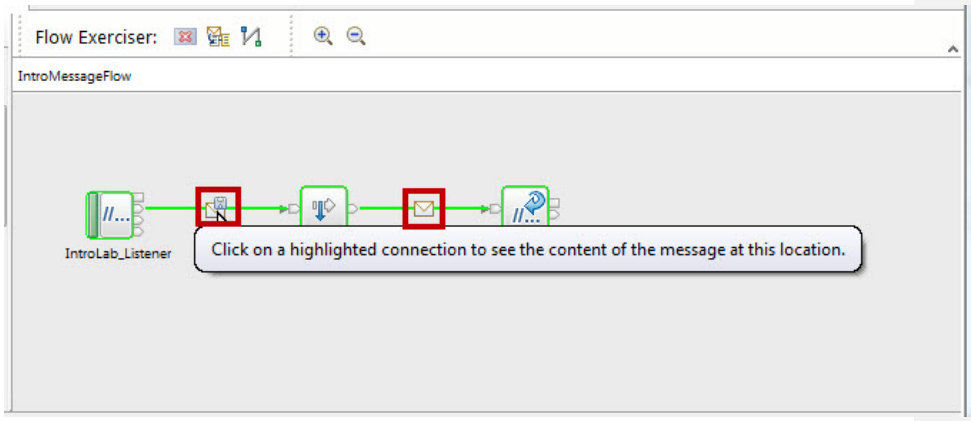
__65. When returned to the flow, you will be pointed to the injected message for analysis, capture (saving), or both.

Note the different icons on the connectors:

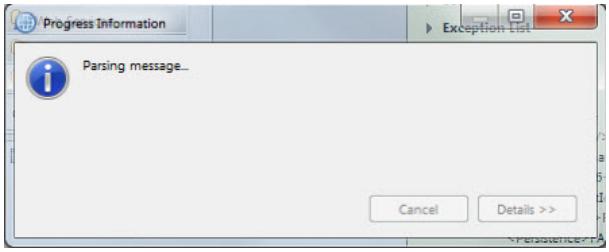
This icon means the recorded message can be viewed, changed and/or saved.

This icon means the recorded message can only be viewed.

Click the icon to see the injected message.

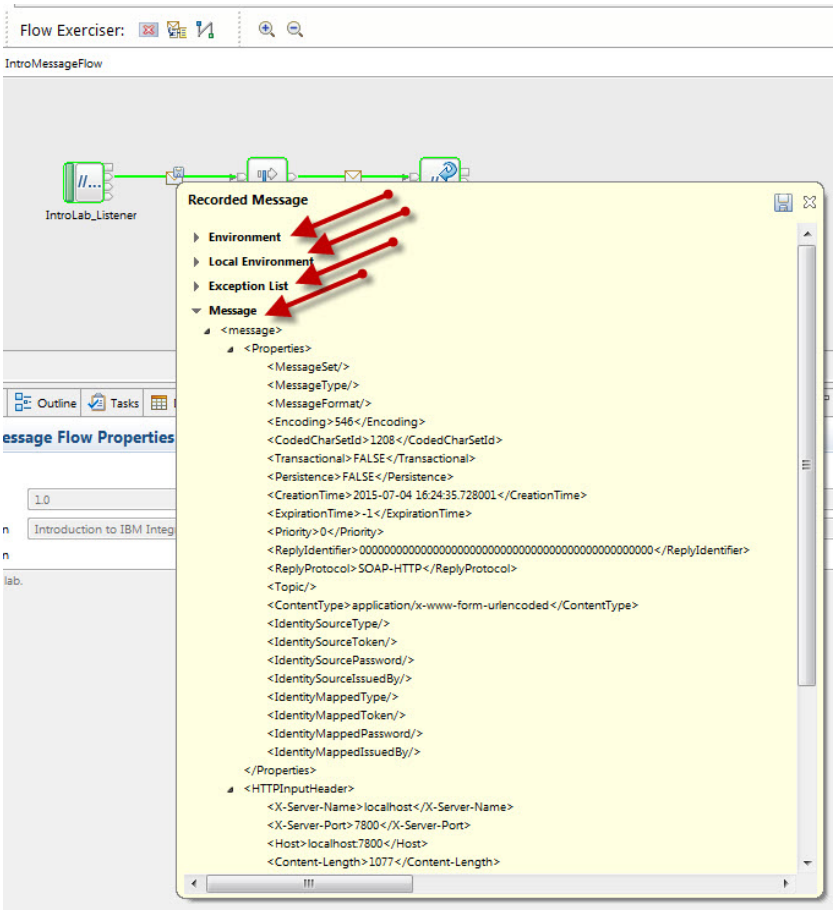


__66. The Flow Exerciser will retrieve the recorded message tree, parse the folders that make up the tree, and render them in XML format for easy viewing:



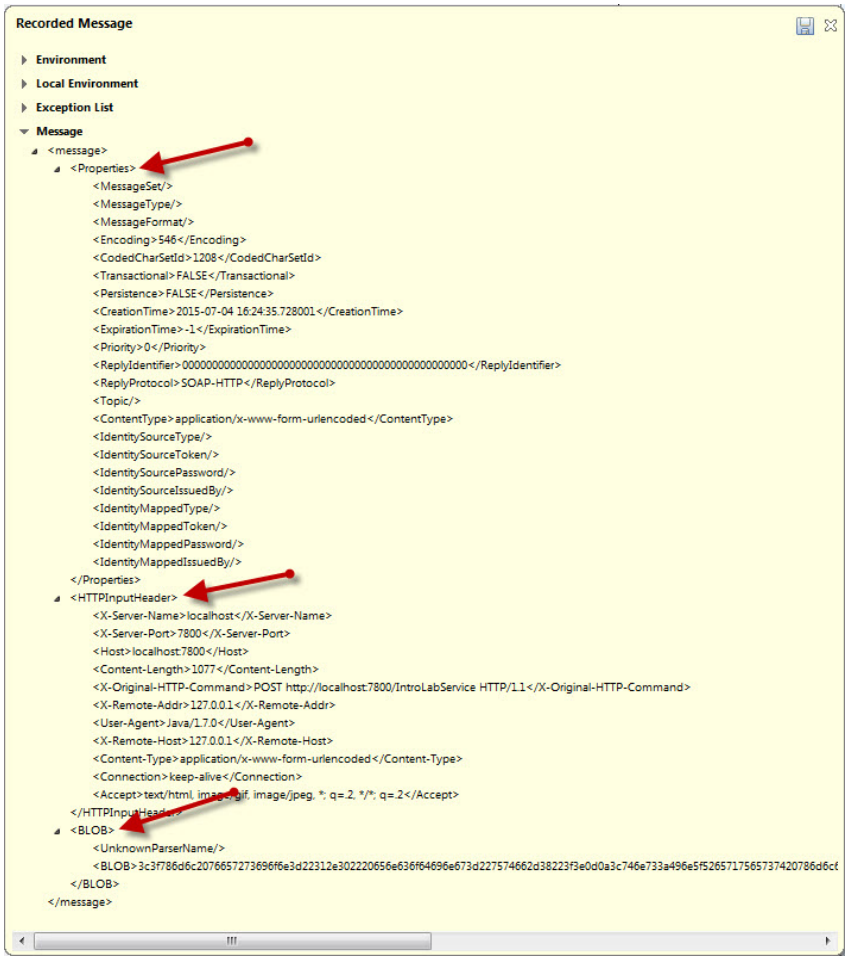
__67. When complete, the message tree will be displayed, with the Message portion of the tree expanded.

Expand the other folders in the tree and examine them if you like.



__68. Focus on the Message portion of the tree.

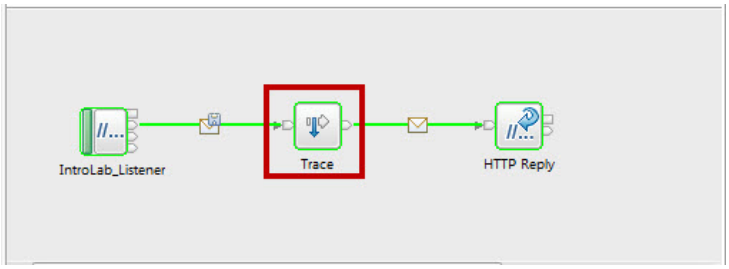
You can see the folders representing the message Properties, followed by the HTTP Input header, followed by the payload (in the <BLOB> folder).



The XML rendering of the message tree makes it easy to interpret.

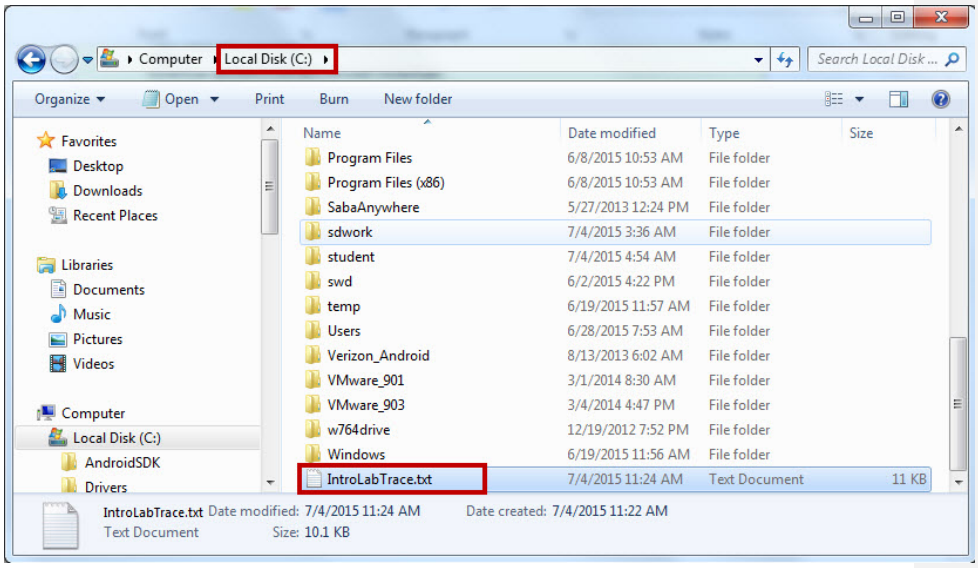
__69. Compare this to a more literal representation of the tree.

The next node in your flow was a Trace node.



__70. Use the Windows Explorer to locate the file the Trace node was configured to write to.

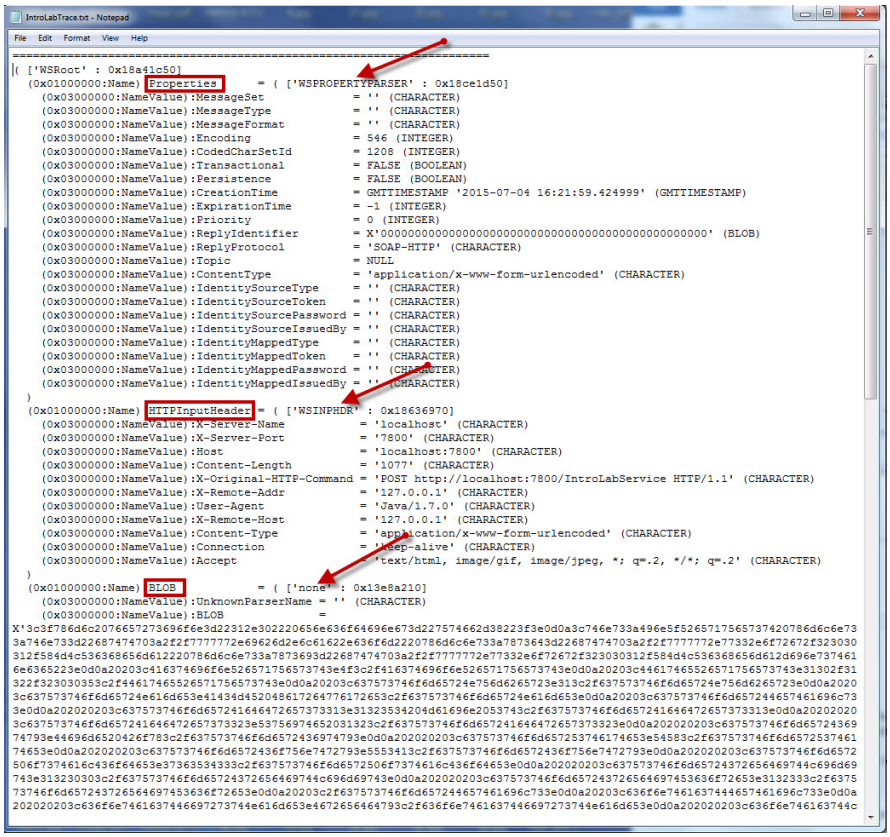
Navigate to **C:\IntroLabTrace.txt(_or location of your choice)**,, and double-click the file to open it.



__71. Recall that you configured the Trace node to capture the “ACE-internal” view of the message tree.

Examine the Trace output. You see ACE-supplied parsers for interpreting the Properties (“WSPROPERTYPARSER”) and HTTP Input Header (“WSINPHDR”). Trace output allows you to see more detail about the structure of header and message fields, such as their type (for example, CHARACTER, INTEGER, or BOOLEAN).

Although you see here ACE-supplied parsers for Properties (“WSPROPERTYPARSER”) and HTTP Input Header (“WSINPHDR”), what is missing is a parser for the payload. Because “none” is supplied, ACE treats the payload as a BLOB.



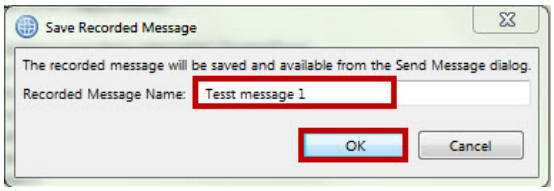
The next section will show how to tell ACE the payload is XML.

__72. Close the Notepad window.

__73. Return to the recorded message, and click the save icon in the upper right corner.

Note: If the recorded message is closed, you can double-click the icon to reopen it.

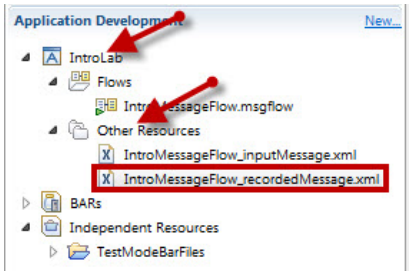
__74. Save the recorded message with the name “Test message 1”, and click **OK**.



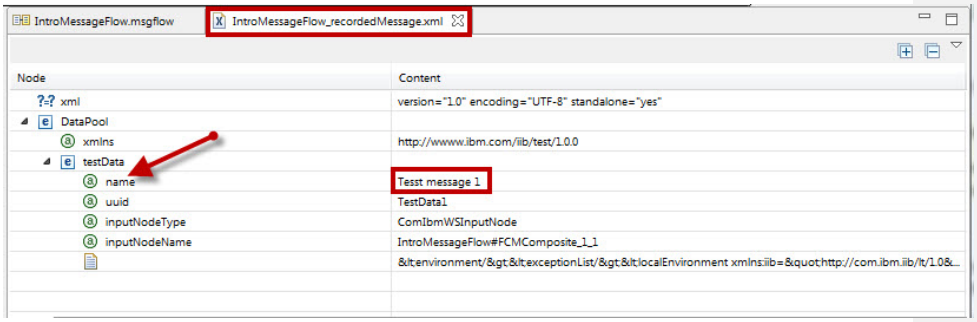
__75. If still open, close the recorded message window.

__76. In the Application Development pane, expand the **Other Resources** folder under the IntoLab project. You will see the recorded message you saved, with the name **IntroMessageFlow_recordedMessage.xml**.

__77. Double-click this file to open it.

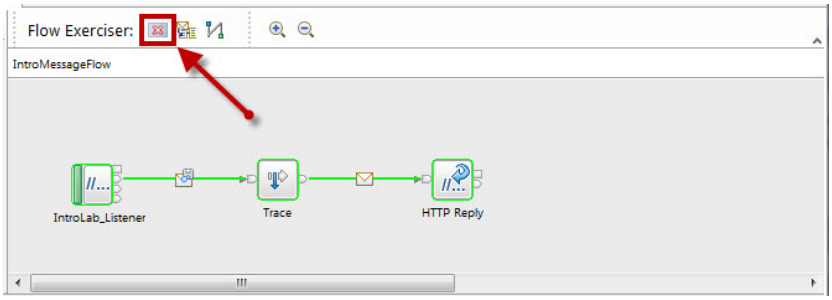


__78. The XML editor will open. You can see the name you assigned to the recorded message.

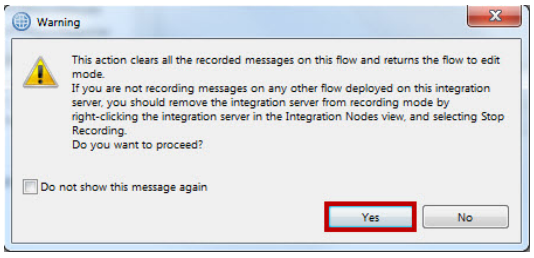


__79. Close the XML editor.

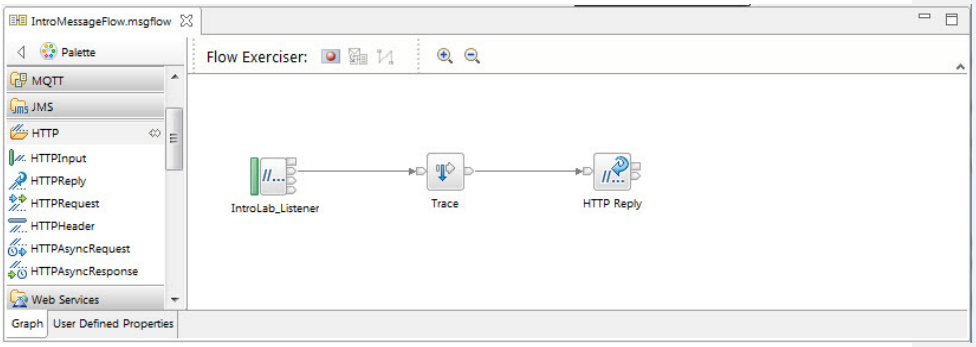
__80. Stop the Flow Exerciser.



__81. Click **Yes** on the pop-up.



__82. The Flow Exerciser is stopped, and the message flow is returned to editable mode.



Continue to the next section to see how to tell ACE that the payload is XML.

1.5 Message models and working with XML messages

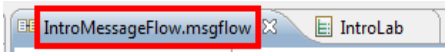
In this section, the IntroMessageFlow will be modified to identify the parser (XMLNSC) to be used to process the message.

The steps are very simple:

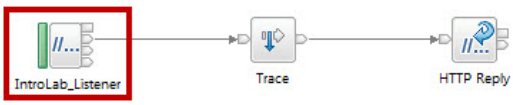
- The properties of the input node will be modified.
- The Flow Exerciser will be used to run another test.
- The recorded message, as well as the trace file contents, will be viewed to see the difference.

1.6 Using the XML Parser

__83. In the IBM ACE Toolkit, click the **IntroMessageFlow** tab to bring the message flow into view.



__84. Click the **IntroLab_Listener** node to bring its properties into view.

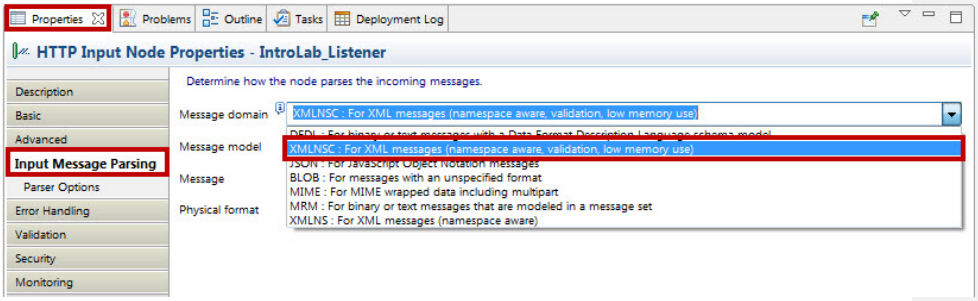


The message flow will be modified so that it uses the XMLNSC parser to process the input message.

__85. 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 Flow Exerciser as well as the Trace file.

__86. 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.

__87. Select the **XMLNSC** parser. The **XMLNSC** parser is the most modern and most efficient of the three XML parsers supplied with App Connect Enterprise, and should be the parser of choice for most if not all message flows that handle XML messages. The other parsers are there for backwards compatibility.



__88. 

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

1.6.1 Key Idea: Parsers and message domains

.

Key Idea: Parsers and message domains.
Read the section below for additional information on parsers and message domains.

As discussed previously, App Connect Enterprise supplies a range of parsers to parse message content and serialize message trees in different formats.

A parser is called when the bit stream that represents an input request or event 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 message syntax tree representation of the bit stream.

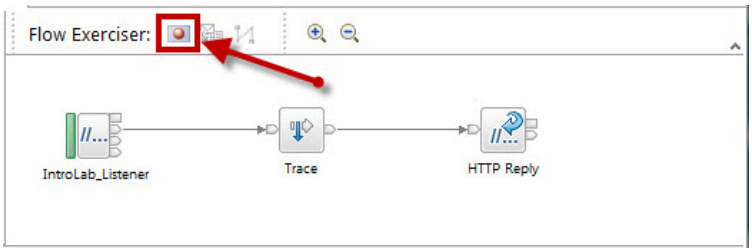
A parser is also used to serialize a logical syntax tree structure into a bit stream (for example on an output node). This process is known as serializing.

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 App Connect Enterprise:

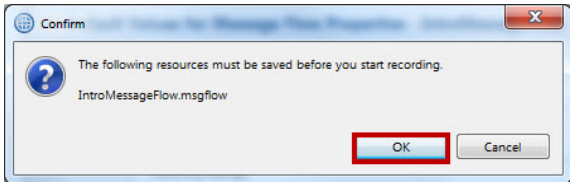
- XMLNSC – for XML documents
- DFDL – for general text or binary data streams including industry standards
- JSON – for JSON documents
- DataObject – for data without a stream representation (used by adapters and connectors)

Now, with the parser for the payload identified, re-run the Flow Exerciser.

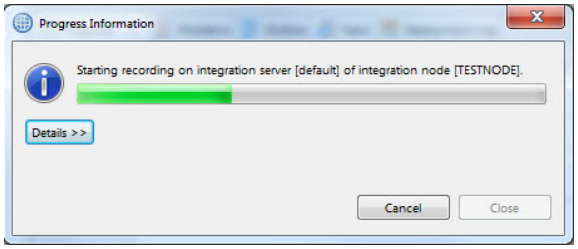
__89. Start the Flow Exerciser.



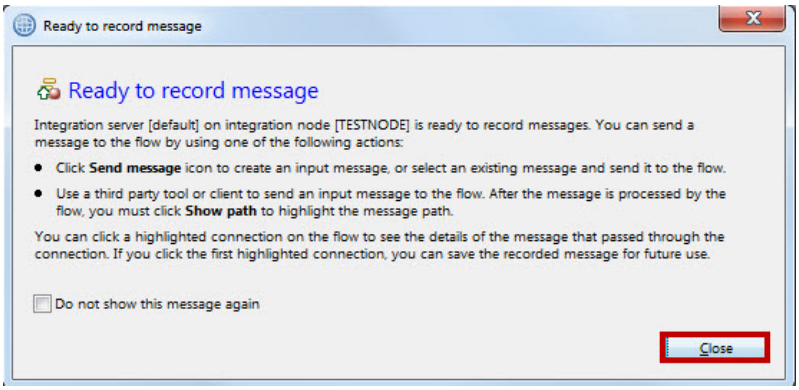
__90. If the flow needs to be saved you will see this pop-up. Click **OK**.



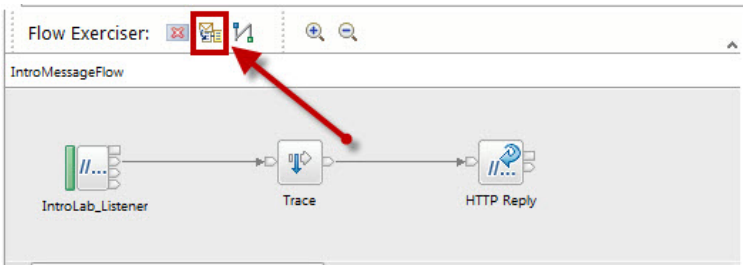
__91. The Flow Exerciser will start.



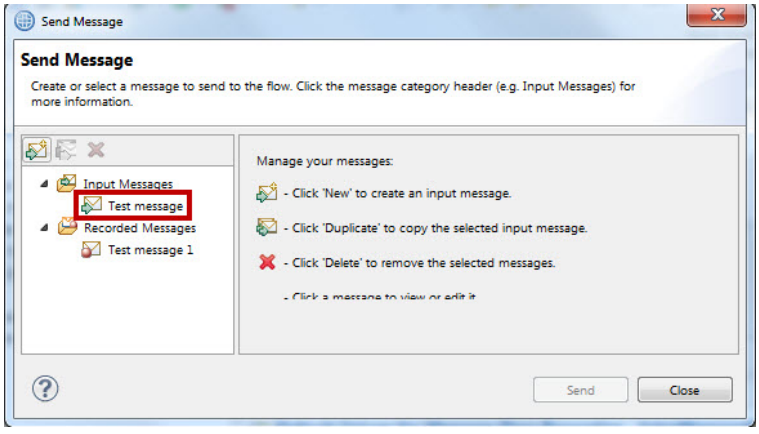
__92. Click **Close** to start recording.



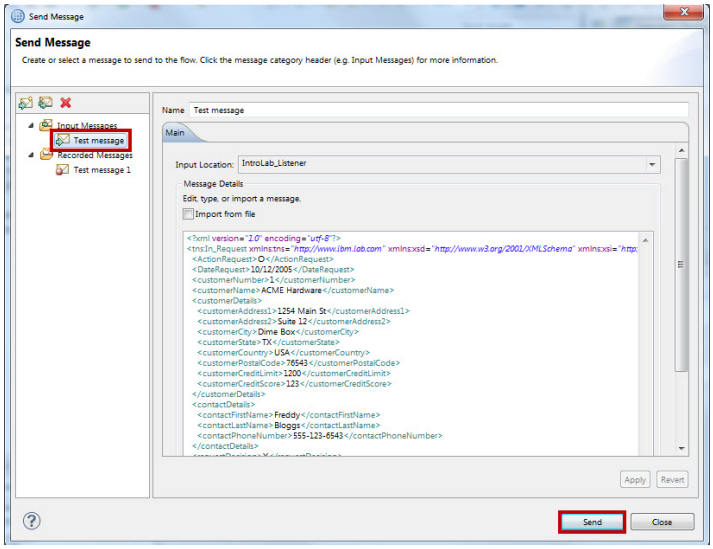
__93. Click the **Send message** icon.



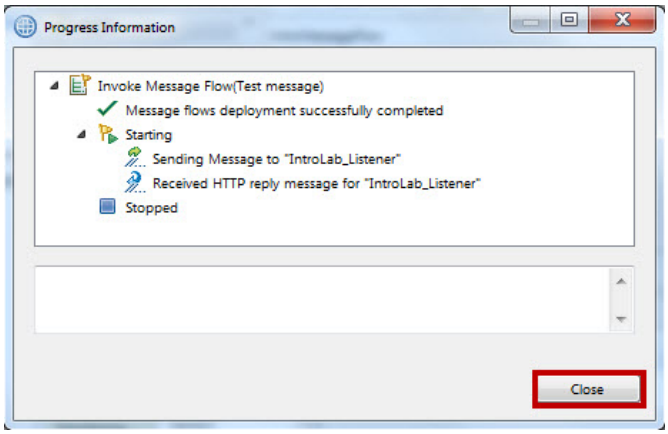
__94. The Input message from the previous run is still available. Click **Test message**.



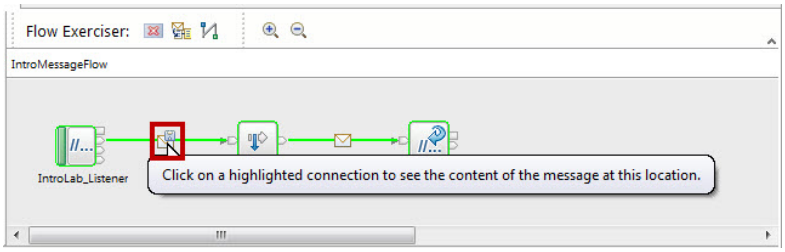
__95. The test message will be loaded. Click **Send**.



__96. The Flow Exerciser will run. Once it stops, click **Close**.

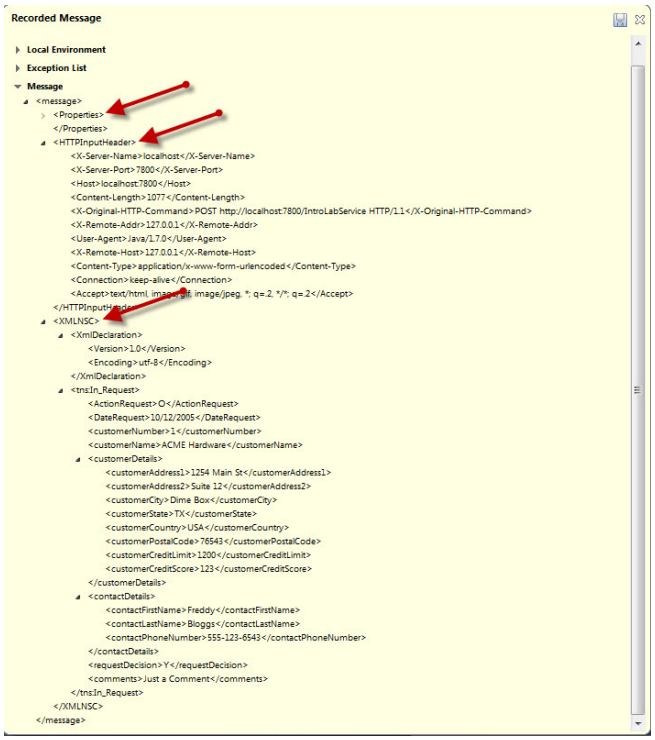


__97. Click the highlighted message.



__98. Note the difference between this run and the previous one.

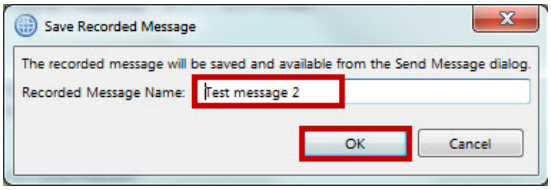
The Properties and HTTPInputHeader folders are there as before. But this time, instead of BLOB and UnknownParserName, you see the XMLNSC parser was used to successfully parse the message payload.



__99. Save this recorded message by clicking on the save icon in the upper right corner.

Note: If the recorded message is closed, you can double-click the icon to reopen it.

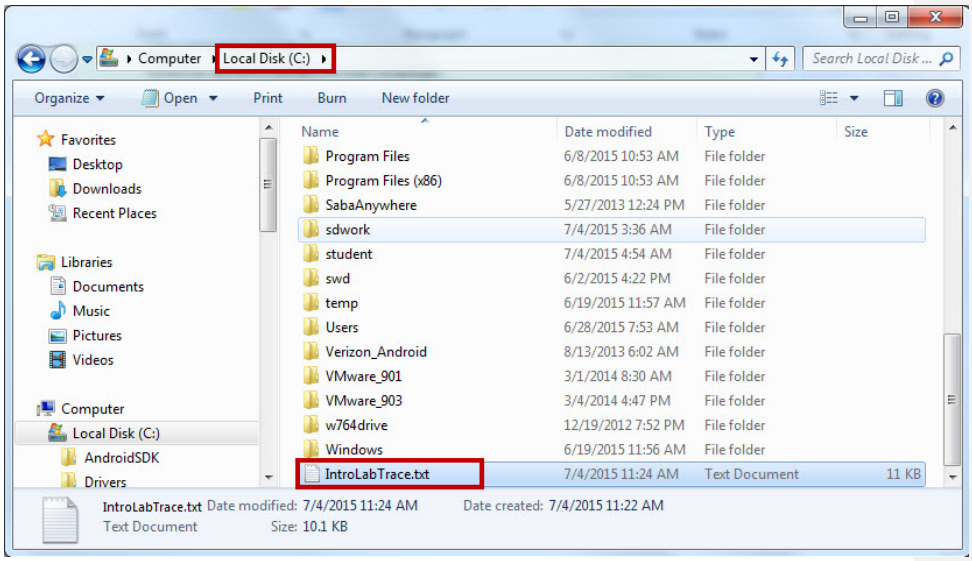
__100. Call this recorded message “Test message 2” and click **OK**.



__101. Close the Recorded Message window.

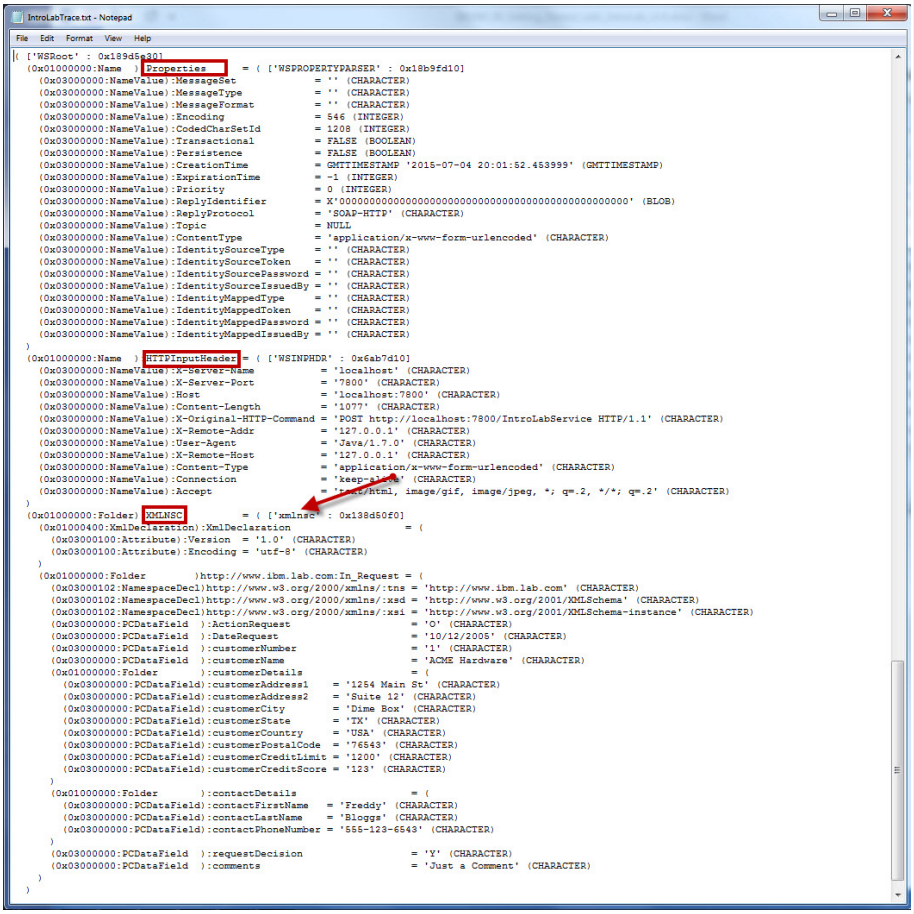
__102. Use the Windows Explorer to locate the file that the Trace node was configured to write to.

Navigate to **C:\IntroLabTrace.txt (_or a different location if you changed it)**, and double-click the file to open it.



__103. Scroll to the bottom of the Trace file.

Again you see the ACE-supplied parsers for interpreting the Properties and HTTP Input Header. But now you also see XMLNSC rather than BLOB for the payload parser, and you see the payload has been parsed and interpreted.



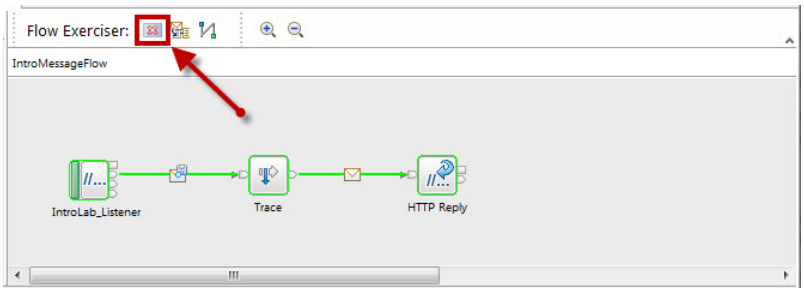
One thing you will notice here is that all the XML fields are CHARACTER.

Without a schema, ACE has no way to tell if any of the fields are another type.

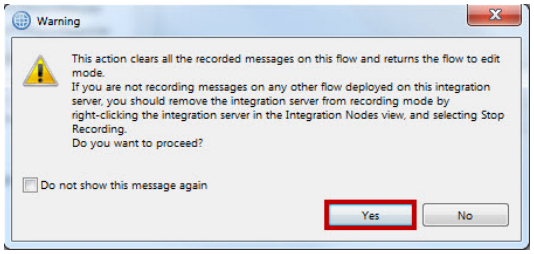
Next you will supply a schema and tell ACE to validate the payload against that schema.

__104. Close the Notepad window.

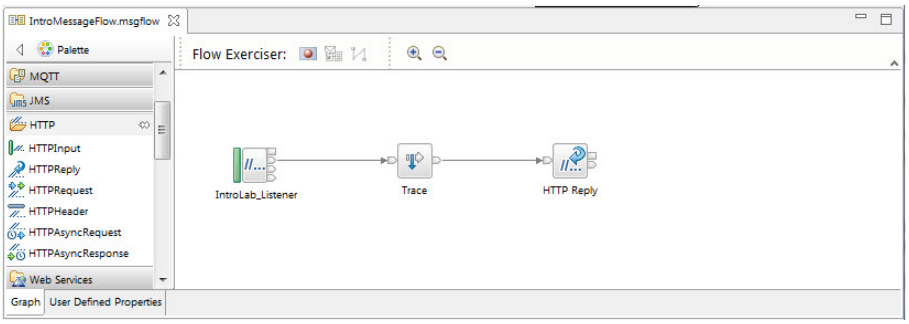
__105. Stop the Flow Exerciser.



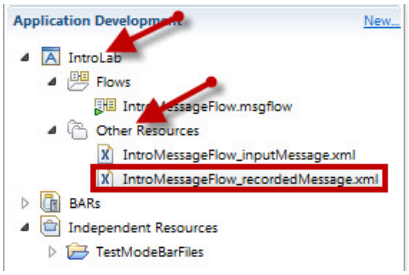
__106. Click **Yes** on the pop-up.



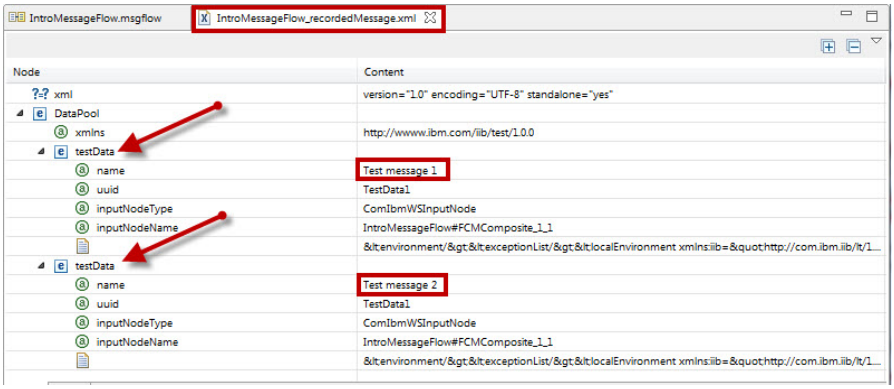
__107. The Flow Exerciser is stopped, and the message flow is returned to editable mode.



__108. In the Application Development pane, expand the **Other Resources** folder. Double-click the **IntroMessageFlow_recordedMessage.xml** file.



__109. The XML editor will open. You should now see two testData messages, with the names you assigned to each.



__110. Close the XML editor.

The next section will show how to tell ACE to use a message model (XML schema) to validate the message contents.

1.7 Using message models

In this section, the IntroMessageFlow will again be modified, this time to identify the XML Schema file that the XMLNSC parser should use in order to correctly parse the input content and serialize the output, as well as perform schema-based validation.

This is needed in this case because the XML data contains three fields that are not CHARACTER data. A message model is needed for the XMLNSC parser to be able to correctly identify those fields.

1.7.1 Key Idea: Message models

Key Idea: Message models.
Read the section below for additional information on message models and their purpose.

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
- REST APIs using JSON

You can model a wide variety of message formats so that they can be understood by App Connect Enterprise 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.

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

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 or Java code as part of your message flows, the language editors can use message models to provide code completion assistance.
- **Re-use 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.

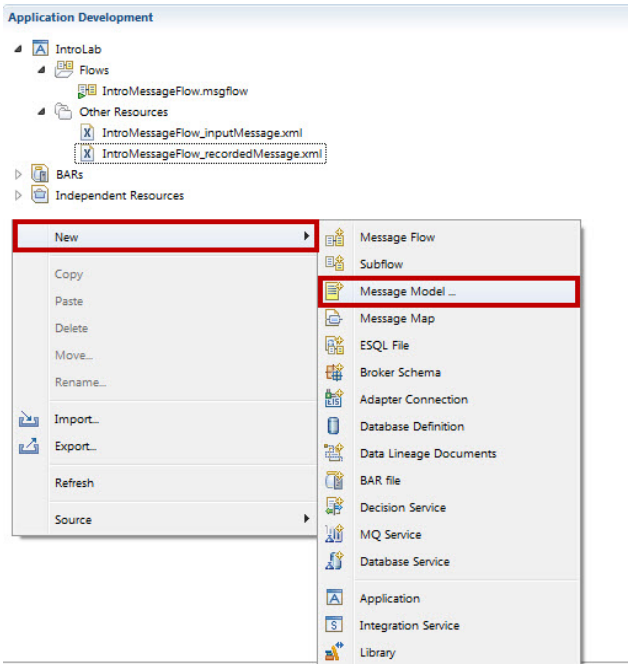
Message models allow the full use of the facilities that are offered by App Connect Enterprise.

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, you can get a message with typed elements and one that is subject to constraints (such as required fields or max field lengths). The toolkit provides wizards to import your existing models (such as WSDLs, XSDs or copybooks).

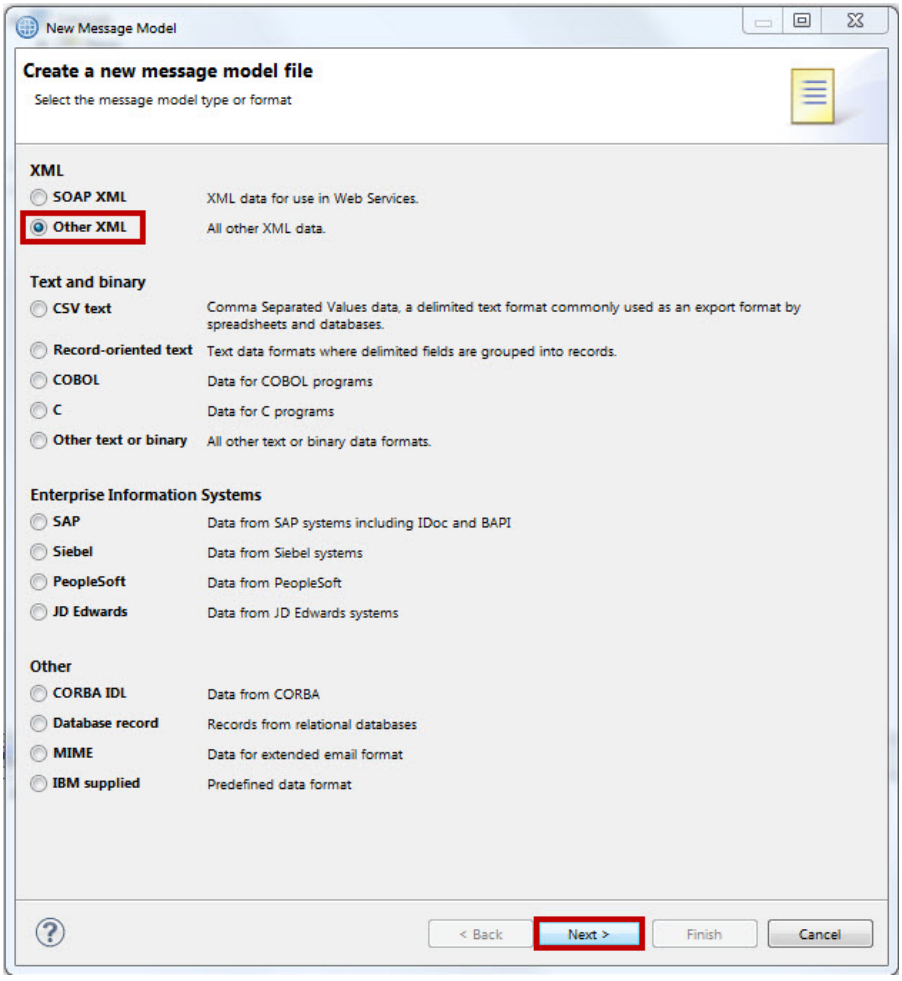
__111. In the Application Development view (project navigator) on the left, right-click the whitespace.

__112. Select **New > Message Model.**



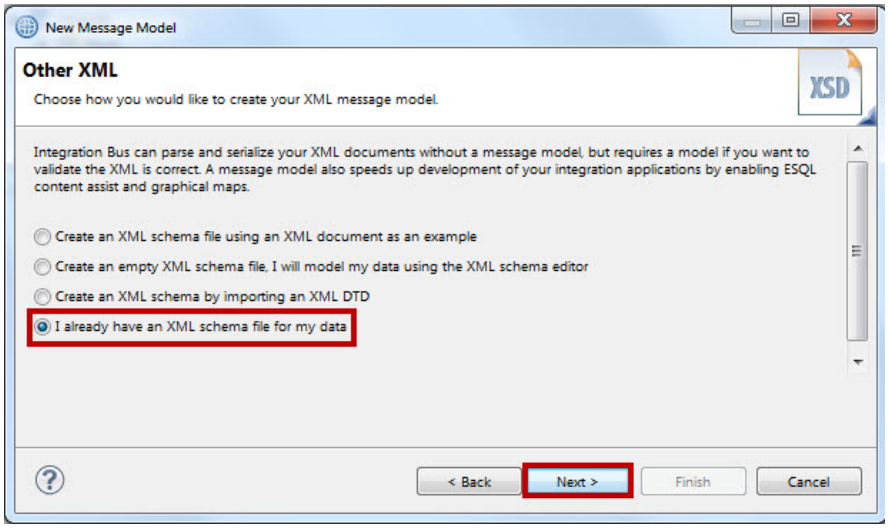
__113. Select the **Other XML** radio button (under **XML**).

Click **Next**.

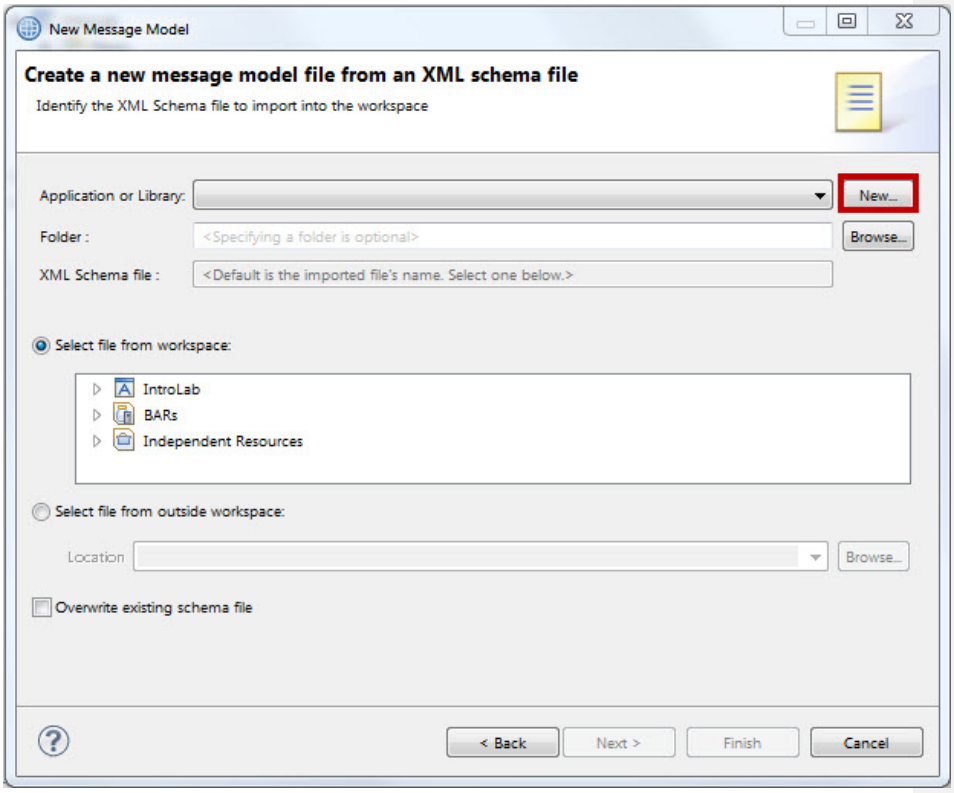


__114. Select the **I already have an XML schema for my data** radio button.

Click Next.

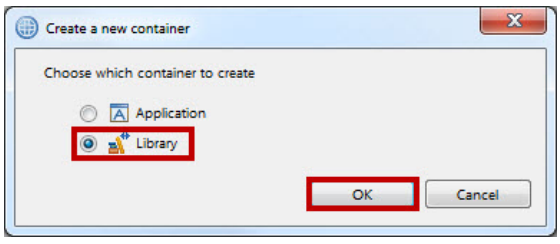

New Message Model
Other XML
Choose how you would like to create your XML message model.
XSD
Integration Bus can parse and serialize your XML documents without a message model, but requires a model if you want to validate the XML is correct. A message model also speeds up development of your integration applications by enabling ESQL content assist and graphical maps.
Create an XML schema file using an XML document as an example
Create an empty XML schema file, I will model my data using the XML schema editor
Create an XML schema by importing an XML DTD
I already have an XML schema file for my data
< Back
Next >
Finish
Cancel

__115. Click the **New** button next to the **Application or Library** field.



__116. In the pop-up window, select **Library**.

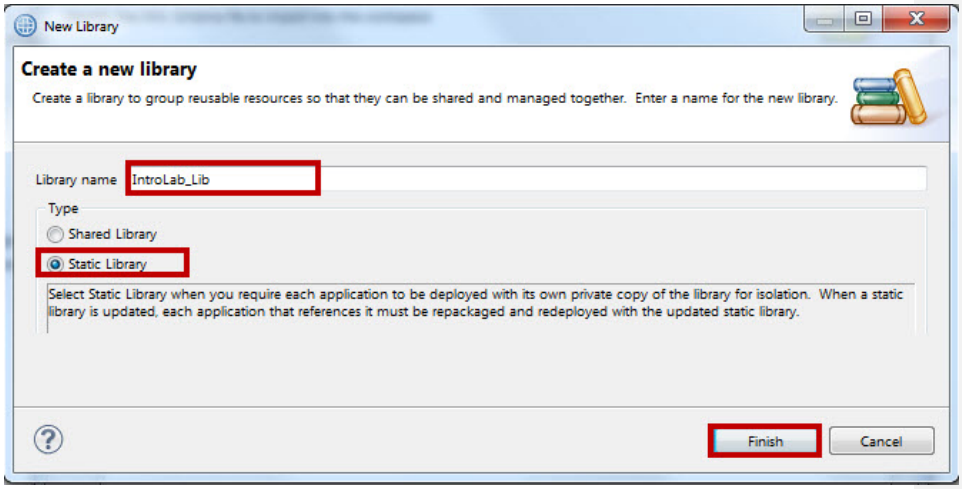
Click **OK**.



__117. In the New Library dialog, type “IntroLab_Lib” as the **Library name**.

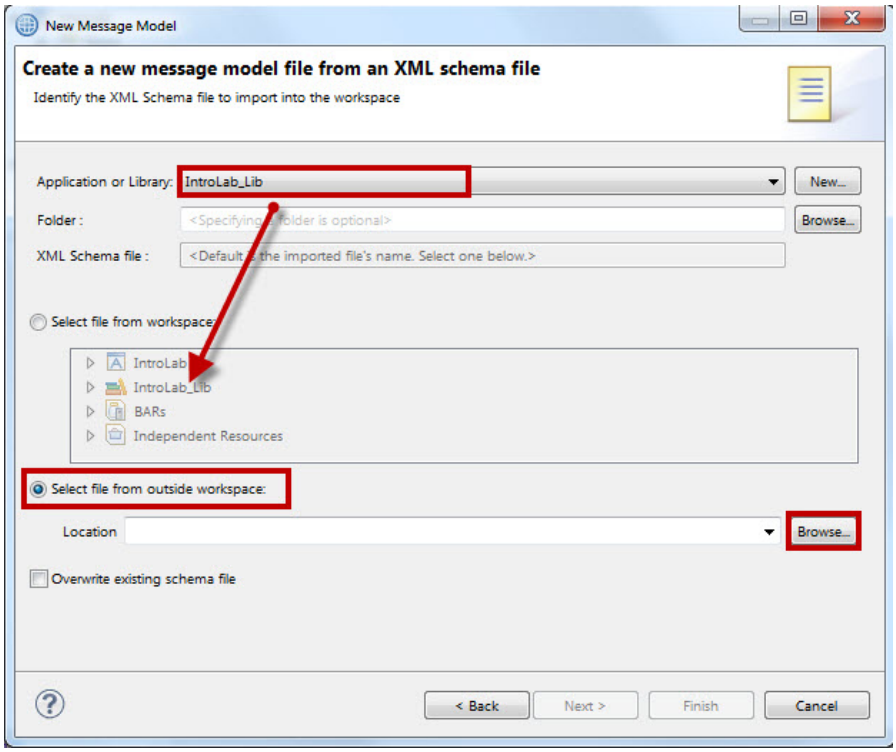
Select the **Static Library** radio button. This means the contents of this library will not be shared across applications and services.

Click **Finish**.



__118. Back in the message model wizard, select the radio button **Select file from outside workspace**.

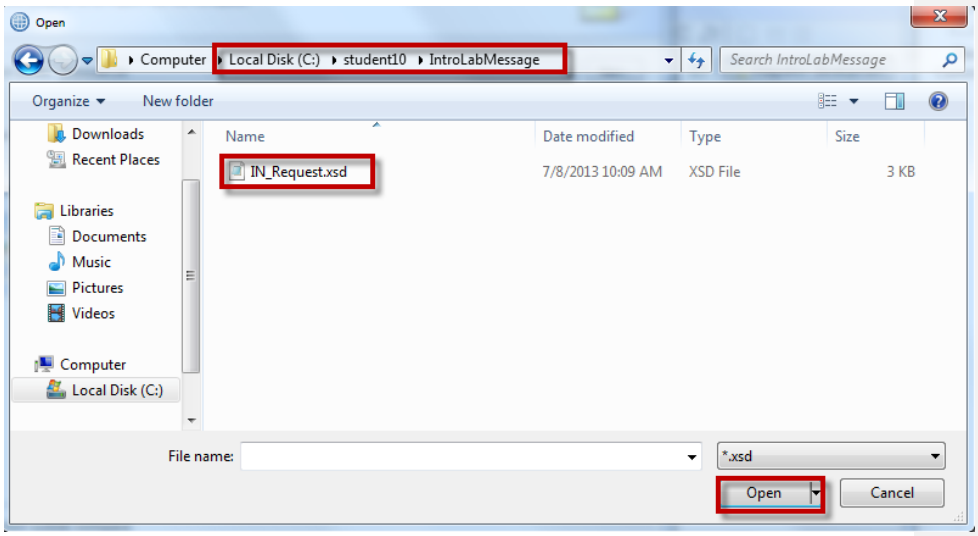
__119. Click **Browse.**



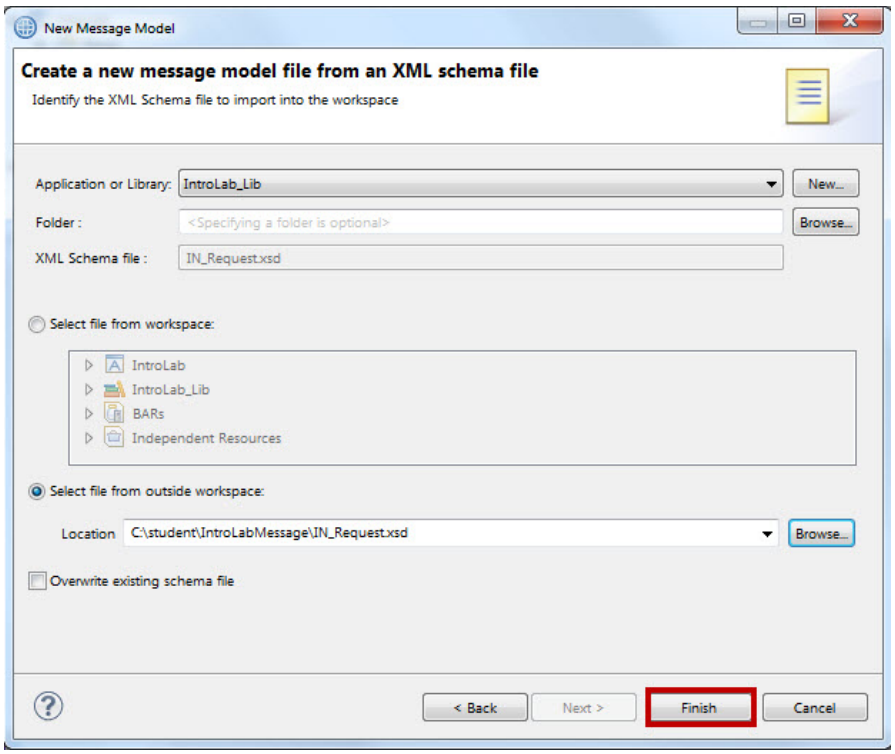
__120. Navigate to the **(your selected lab materials folder)\IntroLabMessage** folder.

Select **IN_Request.xsd**.

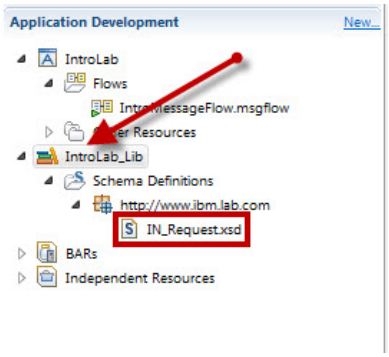
Click **Open**.


Open
Computer
Local Disk (C:)
student10
IntroLabMessage
Search IntroLabMessage
Organize
New folder
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Name
Date modified
Type
Size
IN_Request.xsd
7/8/2013 10:09 AM
XSD File
3 KB
File name:
*.xsd
Open
Cancel

__121. Back in the message model wizard, click **Finish.**



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



1.7.2 Key Idea: Library projects

Key Idea: Library projects.
Read the section below for additional information on the purpose and use of library projects.

Applications and libraries are deployable containers of resources: The contents of these can include 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. Use multiple libraries to group related resources (for example, by type or function).

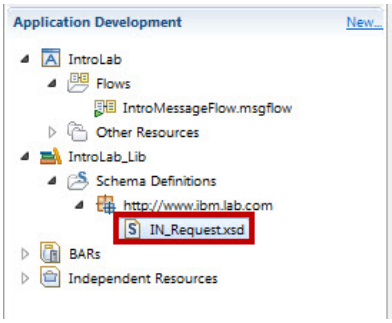
Libraries can be either static (embedded in the applications or services that use them) or shared (meaning they can be referenced across multiple applications or services).

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 App Connect Enterprise routines or definitions.

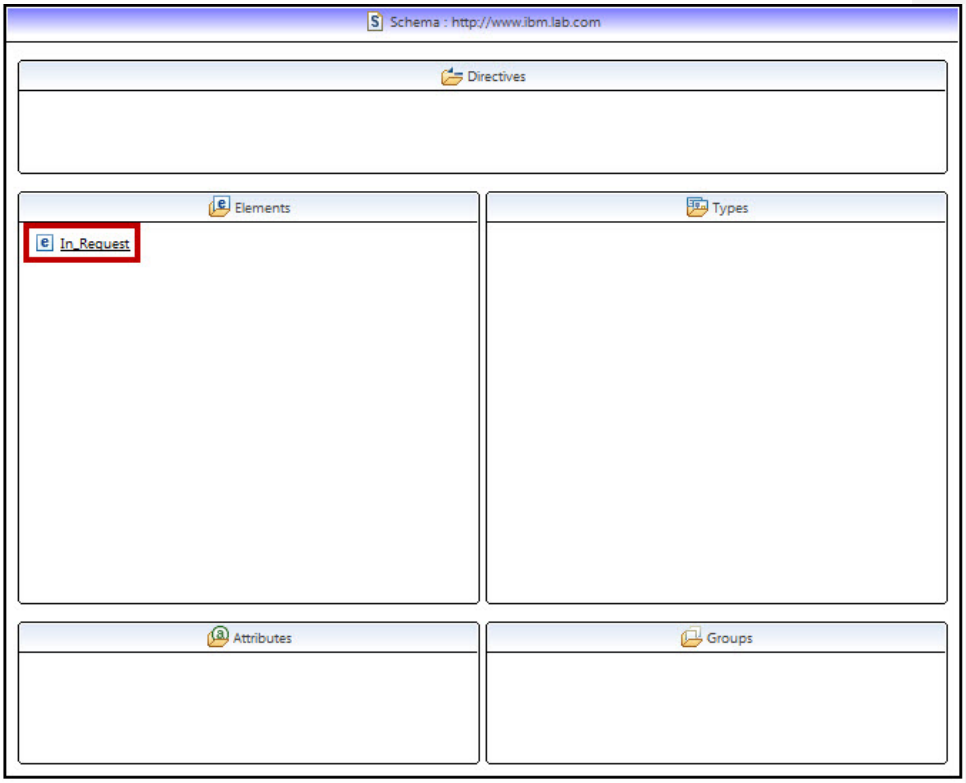
__122. The XML Schema Editor should be opened for you after import.

If it is not, double-click **In_Request.xsd** to open it.

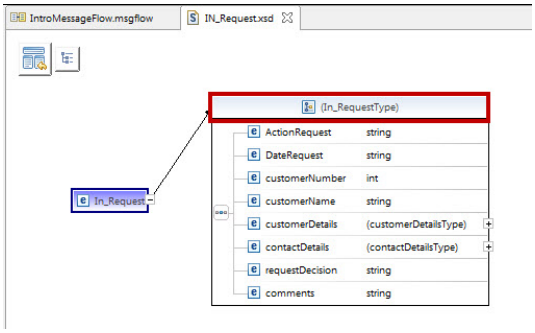


__123. The message model should now be visible in the XML Schema Editor.

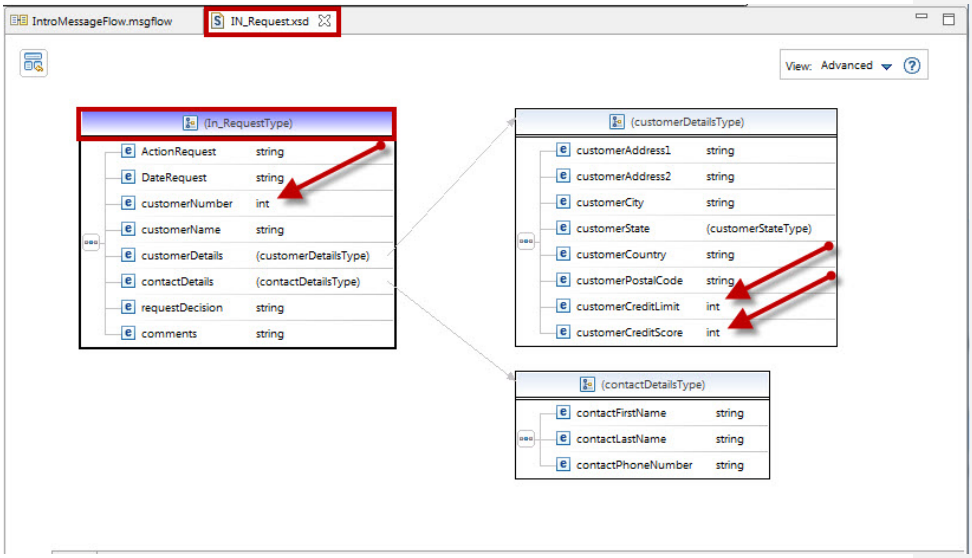
Double-click the **In_Request** element to view the message elements.



__124. Double-click the **(IN_RequestType)** title bar to expand the schema view.



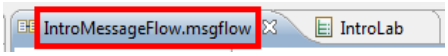
__125. Note that some elements, such as customerNumber and customerCreditScore, are integers (ints) and not strings.



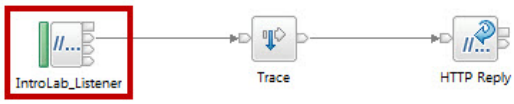
__126. Close the **IN_Request.xsd** tab.

Now, in the message flow, you need to tell the parser to run in *schema-driven* mode, rather than operate in *programmatic* mode.

__127. Click the **IntroMessageFlow** tab to bring the message flow into view.

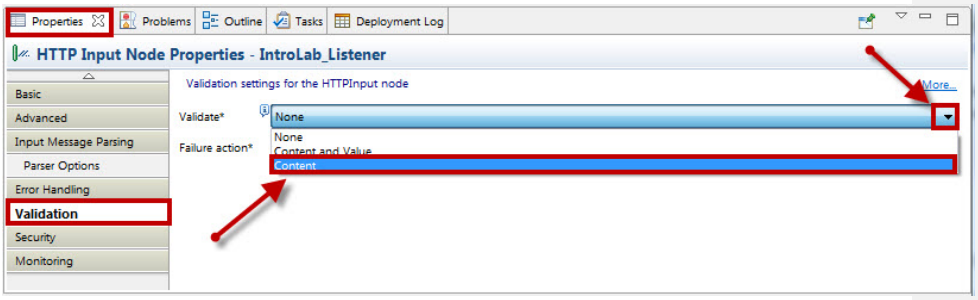


__128. Click the **IntroLab_Listener** node to bring its properties into view.



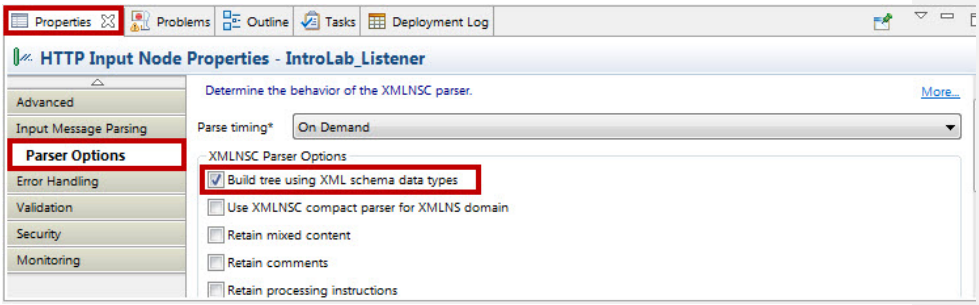
__129. In the **Properties** view, click the **Validation** tab.

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



__130. Select the **Parser Options** tab.

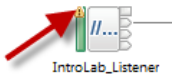
Select the **Build tree using XML schema data types** check box.



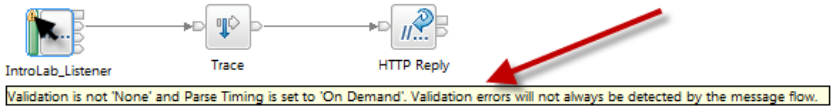
__131. 

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

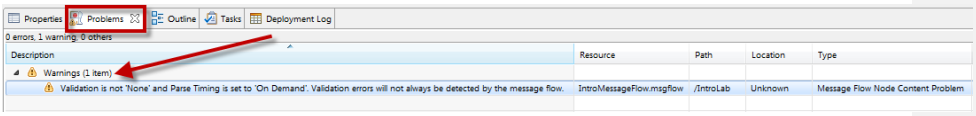
__132. Note that when the flow is saved, a warning appears on the Input node.



__133. Hover the mouse pointer over the node to see the warning.



__134. You can also see this warning if you click the **Problems** tab in the lower right window.



You could resolve this warning, but for now ignore it.

However, this is a good point to spend a minute reviewing how App Connect Enterprise handles parsing.

1.7.3 Key Concept: Parse timing

Key Idea: Parse timing.
Read the section below for information on on-demand vs immediate parsing.

Parsers such as the XMLNSC parser simplify your integration projects by allowing you to work with the logical message tree, as opposed to having to work directly with the wire-format message. And parsers that support messages models such as XMLNSC and DFDL can also be used to validate the structure or content of messages.

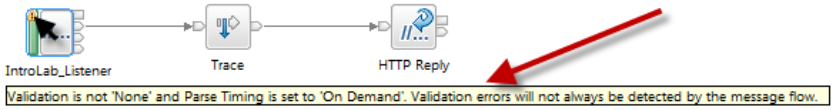
However, in many integration projects there is no need to parse the entire message. For example, you may be simply routing a message based on a small subset of the information in the message (you will see an example of this in the next lab). Or, you may be modifying the message, but only in a trivial way. In such cases, parsing the entire message may incur needless overhead.

For this reason, App Connect Enterprise gives you the ability to dictate when and to what extent parsing is to be performed. One example of this is parse timing.

By default, Integration Bus will use on-demand parsing. This means the message will only be parsed if elements within it are referenced, the parsing will occur at the time of reference (“on demand”), and the message will only be parsed up to the point of reference. In many cases, on-demand parsing will save memory and processor overhead, particularly if referenced fields are near the beginning of a message.

On-demand parsing involves a trade-off, though: If a message is malformed, that fact may not be discovered, as the malformation may be at a point in the message for which parsing was avoided. Similarly, if validation is requested, it will only be performed for those parts of the message that are parsed, and so an invalid message may not be detected.

The latter situation is the reason for the warning you see on the IntroLab_Listener node. As the warning indicated, you requested that validation be performed on the content of the message, but the parse timing was set to its default value of “On Demand”, creating the possibility that validation errors would not be



detected.

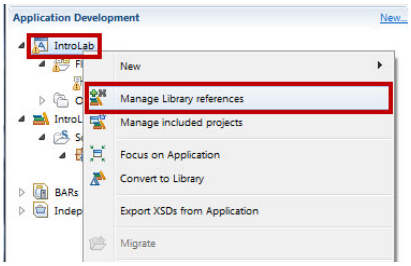
If Parse Timing were set to “Immediate,” the result would be that the entire message would be parsed, allowing validation to be performed against the full message.

As mentioned, you could resolve this warning, but for now ignore it. You will explore dealing with errors and problem determination in a later lab, including parsing errors.

1.8 Project references

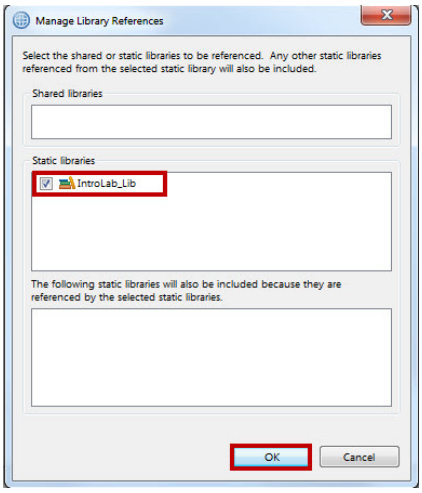
Because **IntroLab_Lib** is defined as a static library, the IntroLab application must have a reference to it so that it is included when the application is deployed.

__135. In the Application Development pane, right-click the **IntroLab** application, and select **Manage Library References** from the menu.

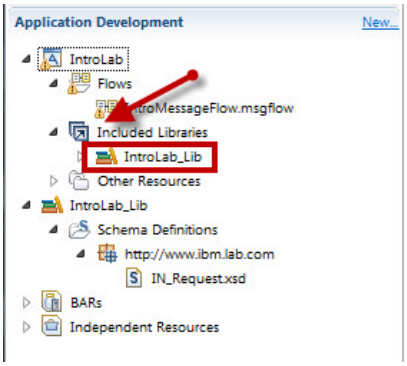


__136. Select the **IntroLab_Lib** checkbox.

Click **OK**.



__137. In the Application Development pane, you should now see that **IntroLab** includes a reference to the **IntroLab_Lib** library.

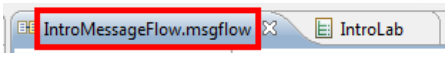


You are now ready to test the application again, using the Flow Exerciser.

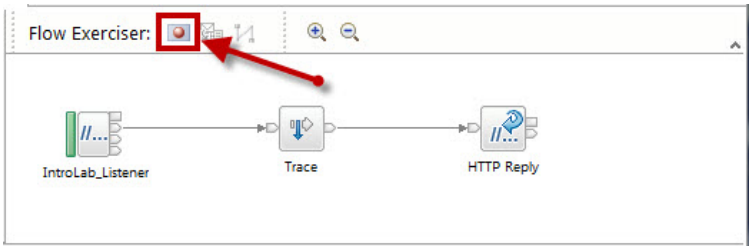
1.9 Test again using the Flow Exerciser

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

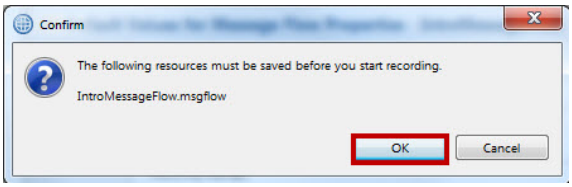
__138. In the IBM ACE Toolkit, click the **IntroMessageFlow** tab to bring the message flow into view.



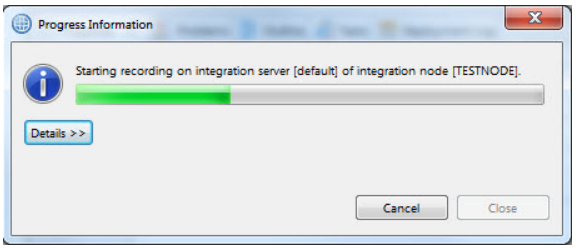
__139. Start the Flow Exerciser.



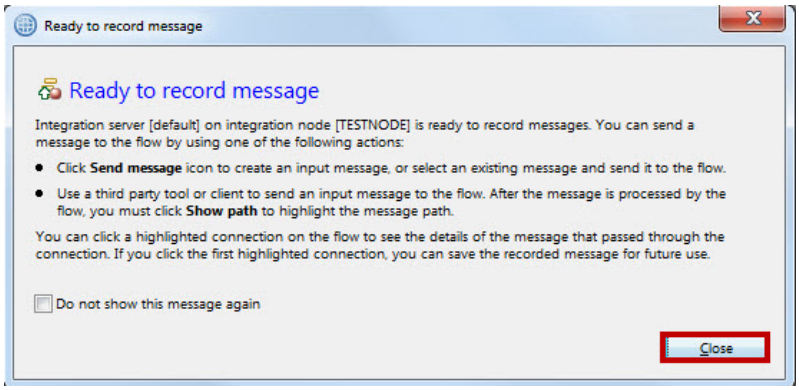
__140. If the flow needs to be saved you will see this pop-up. Click **OK**.



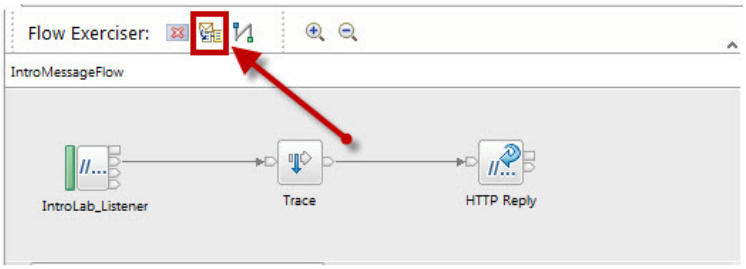
__141. The Flow Exerciser will start.



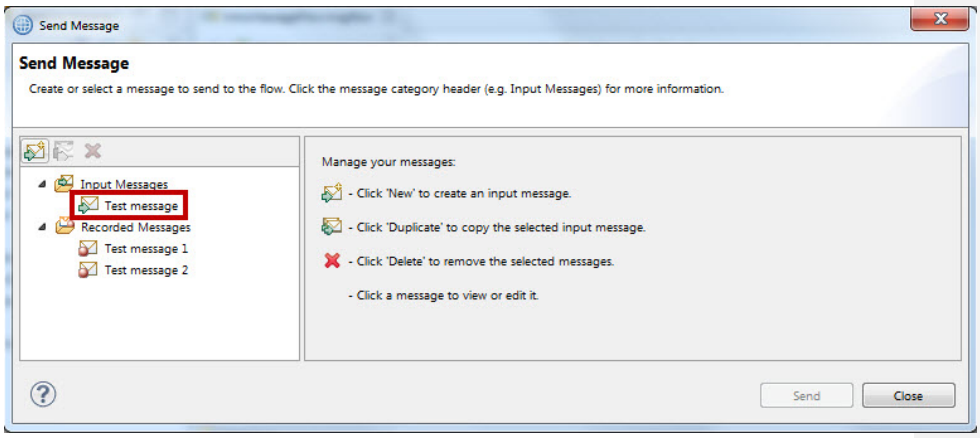
__142. Click **Close** to start recording.



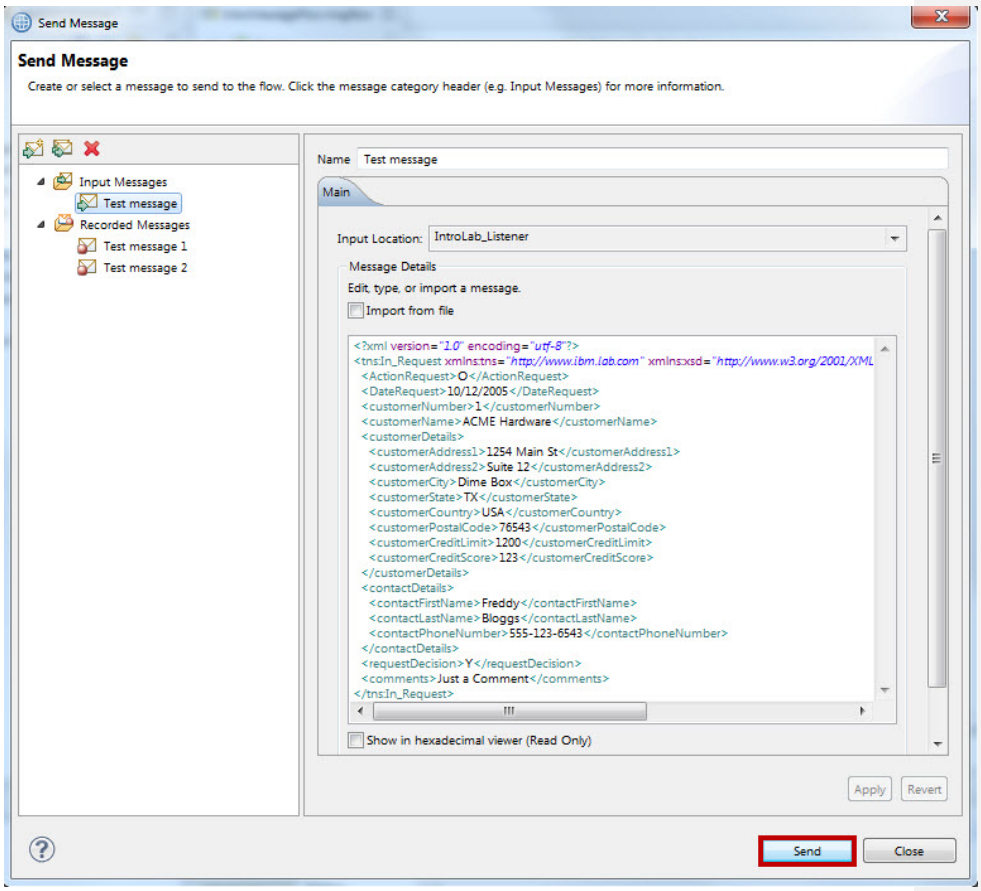
__143. Click the **Send message** icon.



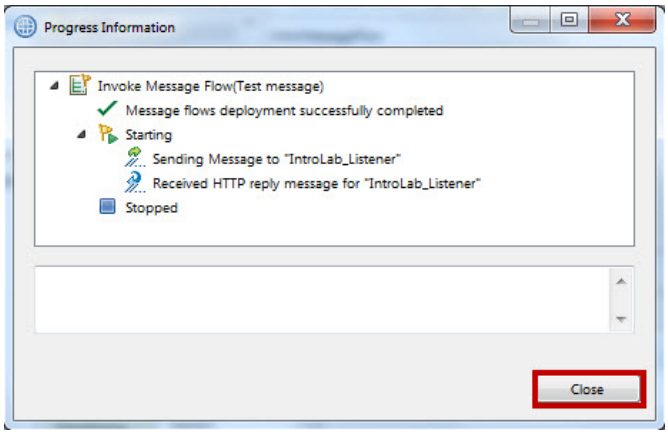
__144. The Input message from the previous tests is still available. Click **Test message**.



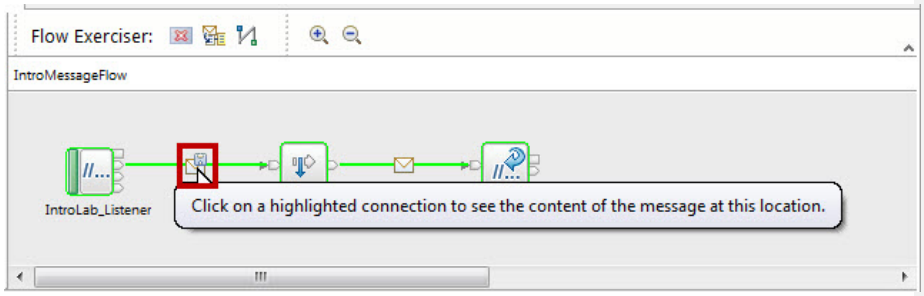
__145. The test message will be loaded. Click **Send**.



__146. The Flow Exerciser will run. Click **Close** after it stops.



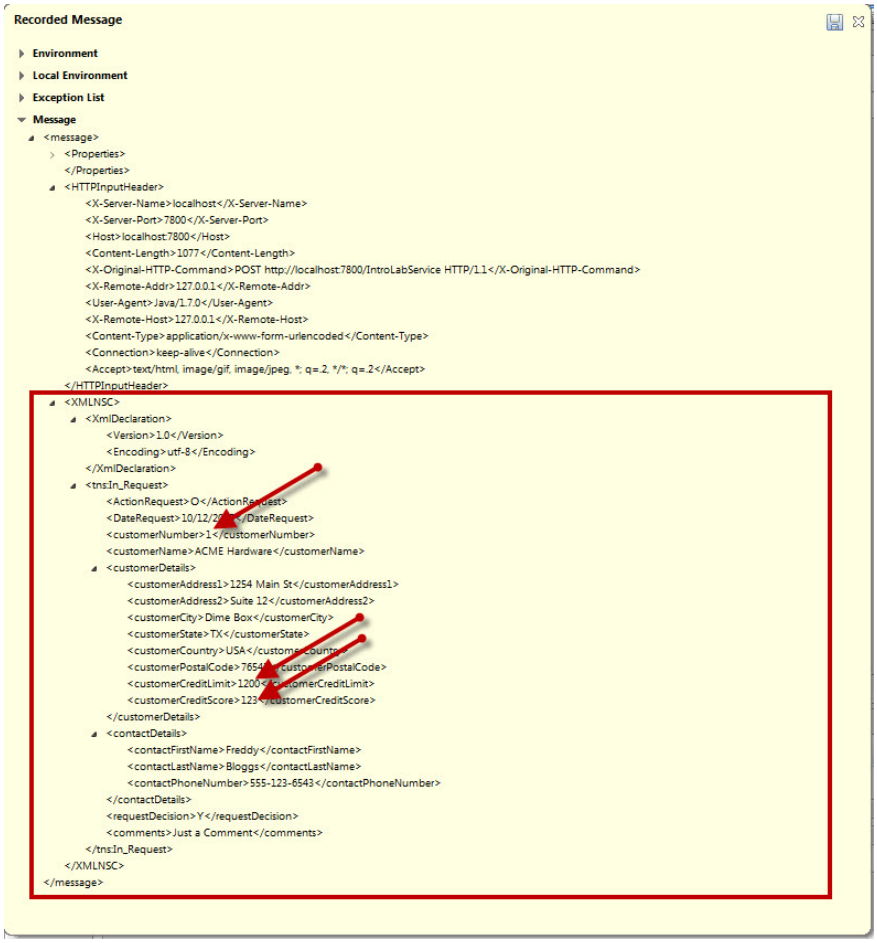
__147. Click the highlighted message.



__148. Note that the elements the XML schema identified as integers look no different here than in your previous test.

But that is correct, because the Flow Exerciser renders the message tree using an XML representation, regardless of their format, and elements in those messages as strings, again regardless of their actual type.

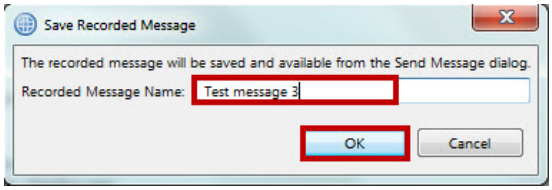
We can return to the Trace file to see the actual physical representation of tree elements, to see if the schema was in fact used when parsing the message.



__149. Save this recorded message by clicking on the save icon in the upper right corner.

Note: If the recorded message is closed, you can double-click the icon to reopen it.

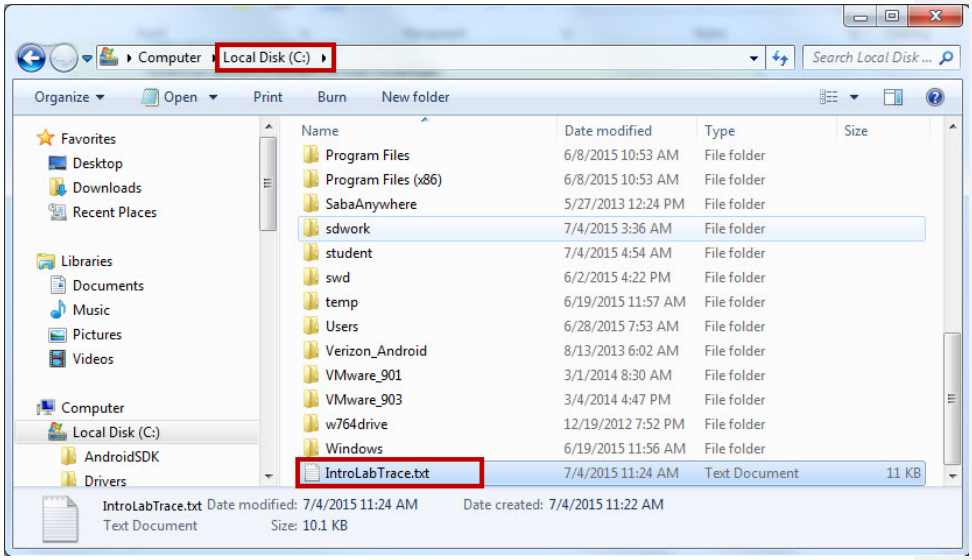
__150. Call this recorded message Test message 3 and click **OK**.



__151. Close the Recorded Message window.

__152. Use the Windows Explorer to locate the file the Trace node was configured to write to.

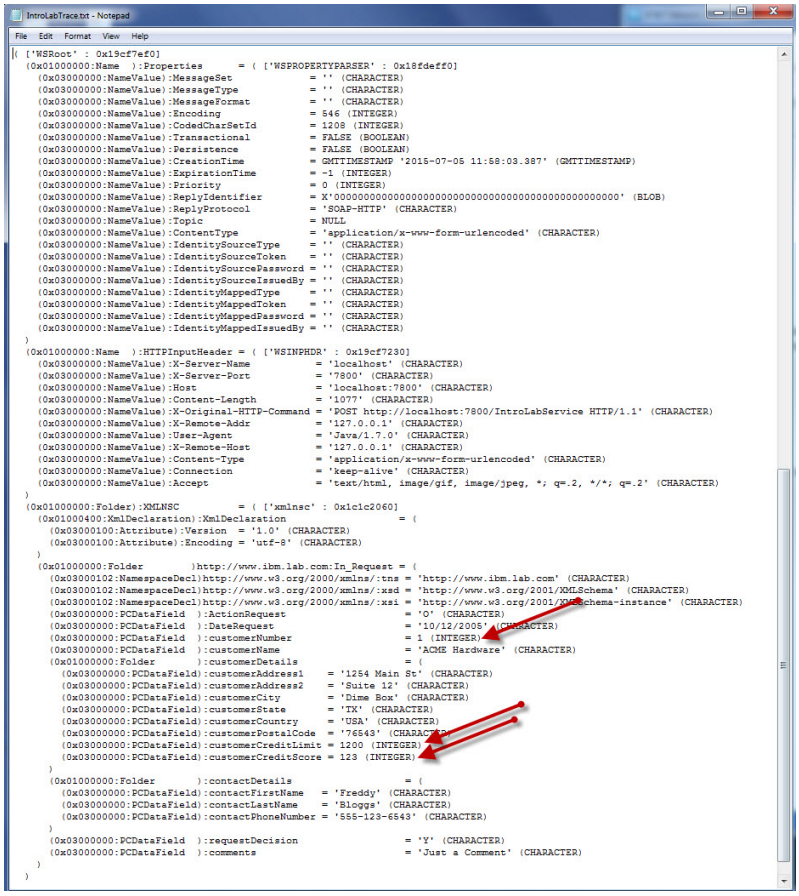
Navigate to **C:\IntroLabTrace.txt**, **(or location your selected in previous step)**, and double-click the file to open it.



__153. Scroll to the bottom of the Trace file (or use **Ctl+End**) to view the new trace output

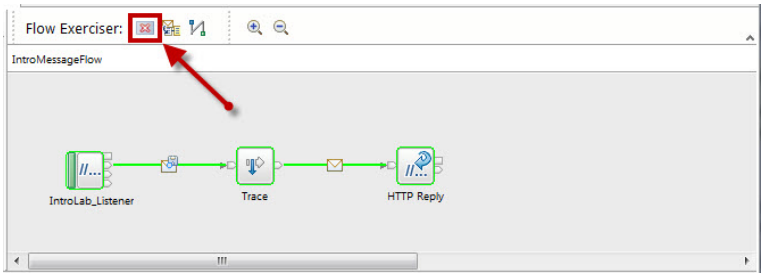
Again you see the ACE-supplied parsers for interpreting the Properties and HTTP Input Header. And you see XMLNSC for the payload parser, and you see the payload has been parsed and interpreted.

However, this time the parser used the schema when constructing the message tree, and correctly identified the type as “integer” for the three indicated elements.

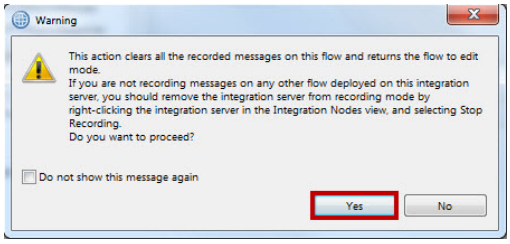


__154. Close the Notepad window.

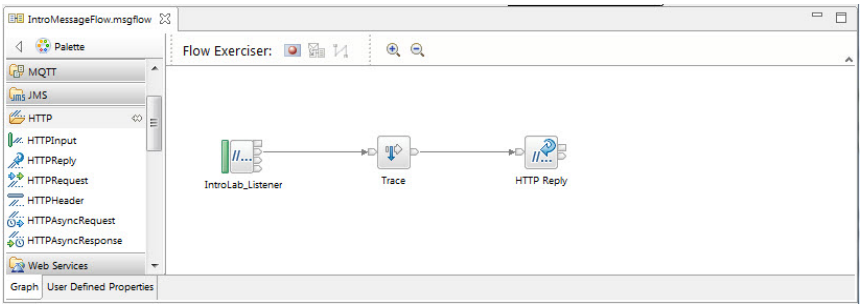
__155. Stop the Flow Exerciser.



__156. Click **Yes** on the pop-up.



__157. The Flow Exerciser is stopped, and the message flow is returned to editable mode.



END OF LAB 1

Lab 2 Content-based routing

Commented [EL2]: Section footer should be updated to match (hyphenated and sentence case). In fact, all footers should be sentence case.

2.1 Overview

In this lab, you will modify the flow you created in Lab 1 to perform simple routing. Input messages will be sent to one of two destinations depending on the country code. Addresses in the United States will be routed to a US reply path, while addresses in Canada will be routed to a Canadian reply path.

You will also include some basic exception handling, to deal with addresses that are not in the United States or Canada.

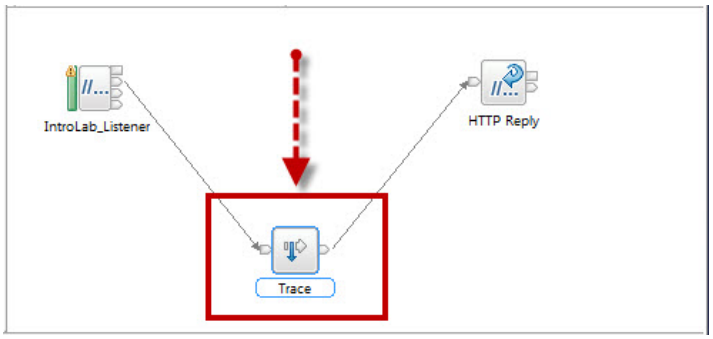
2.2 Add routing logic

__1. Return to the Integration Bus Toolkit.

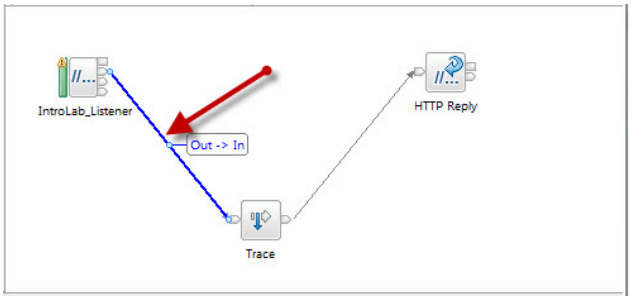
__2. Select the **IntroMessageFlow** message flow in the message flow editor.

__3. Select the **Trace** node.

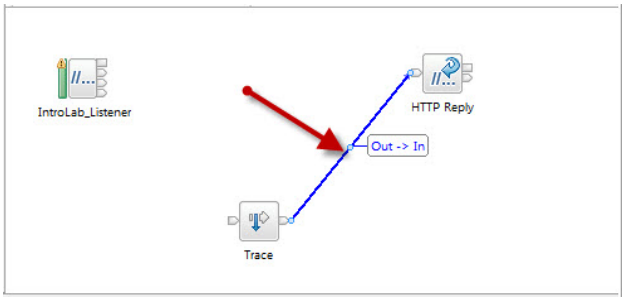
Drag the node down a short distance.



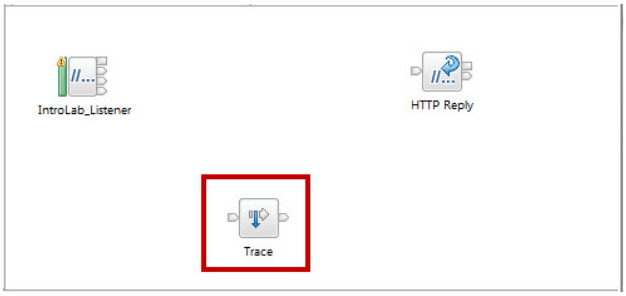
__4. Click the **Out -> In** connector shown – the connector will turn **blue**. Press the **Delete** key to remove the connector.



__5. Click the other **Out -> In** connector shown – the connector will turn **blue**. Press the **Delete** key to remove the connector.

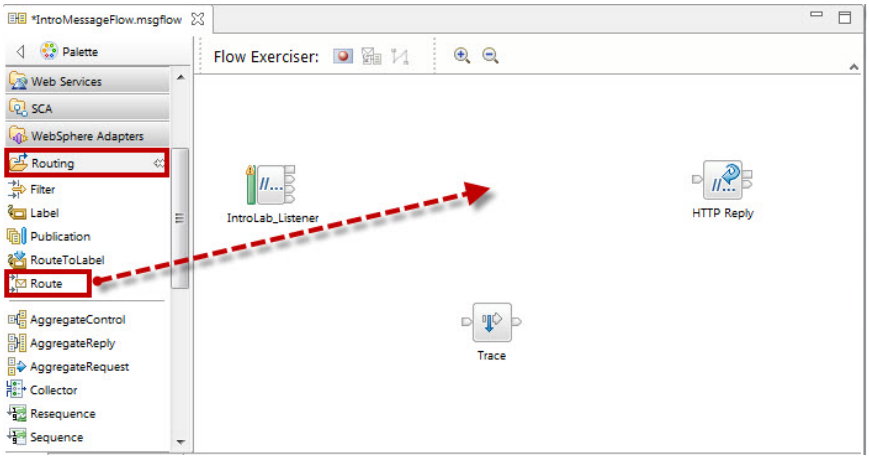


__6. Leave the Trace node unwired for now. Do not delete it. You will be repurposing it in a later lab.



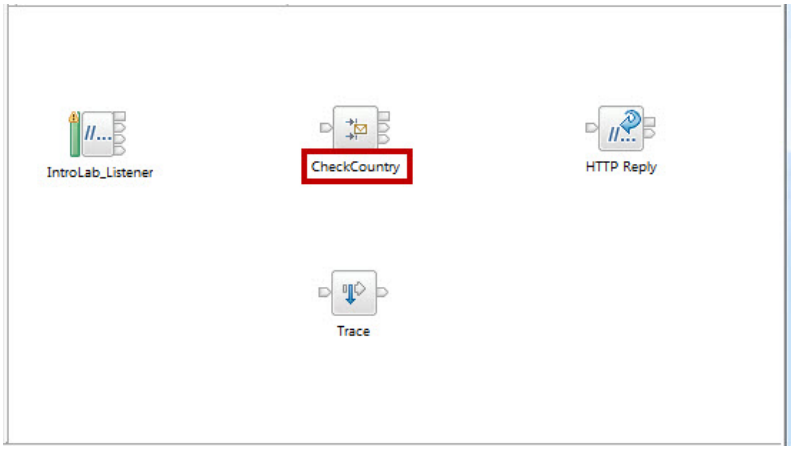
__7. Expand the **Routing** drawer in the Palette.

__8. Select a **Route** node and place it between the IntroLab_Listener and HTTP Reply nodes.



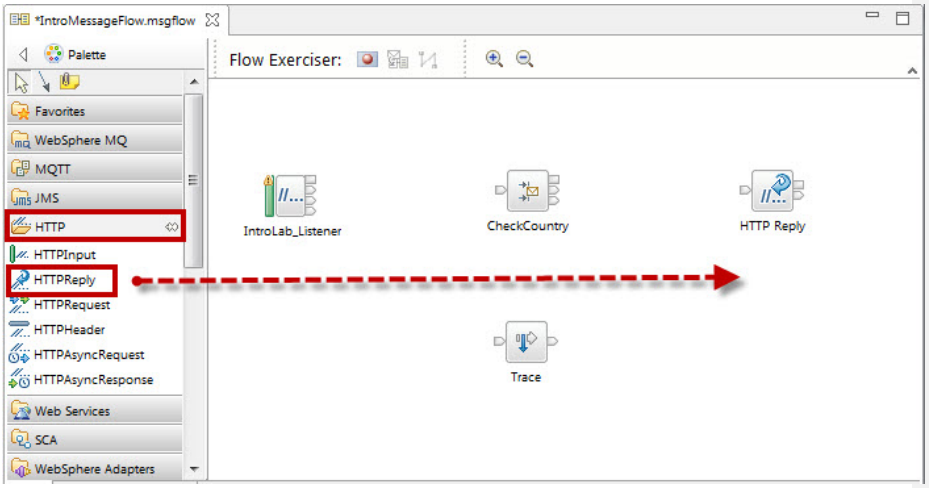
__9. Change the name of the new routing node to “CheckCountry.”

Press **Enter** to complete the rename operation.



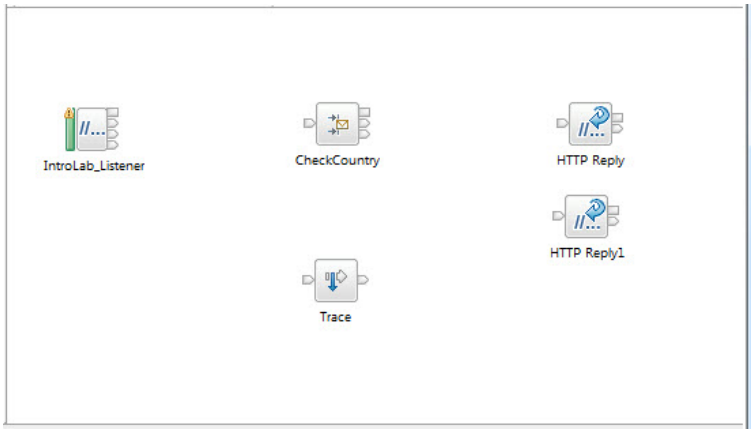
__10. Expand the **HTTP** drawer in the Palette.

__11. Select an **HTTPReply** node and place it beneath the existing **HTTP Reply** node.



__12. Your flow should look something like this.

Do not rename the new HTTP Reply node; you will be renaming them both shortly.

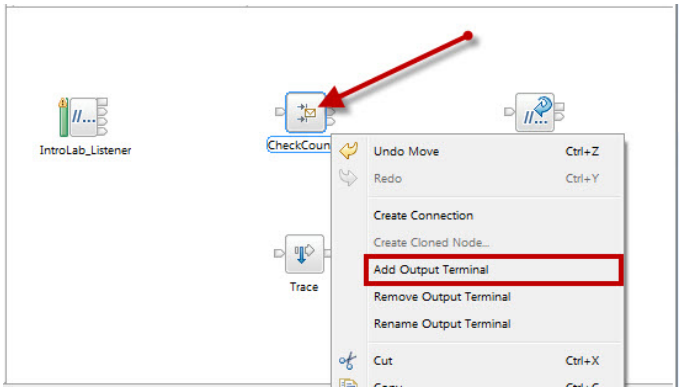


Terminals will now be added to the CheckCountry routing node for US and Canadian addresses.

__13. Select the **CheckCountry** route node.

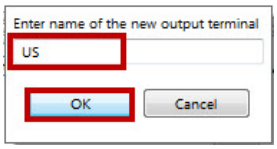
Click the right mouse button.

Select **Add Output Terminal** from the menu.



__14. Enter **US** as the name of the new output terminal.

Click **OK** to continue.

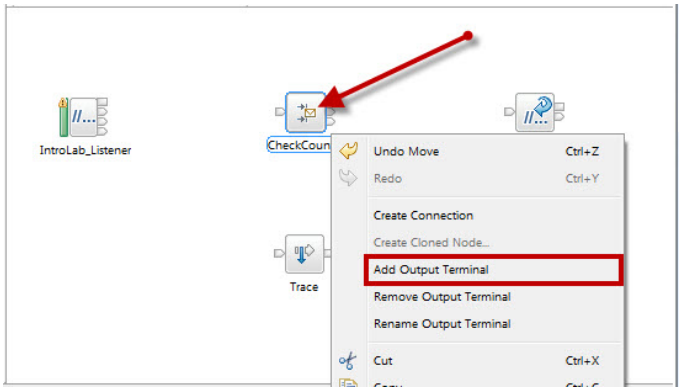


The steps will now be repeated to add a terminal called **Canada**.

__15. Select the **CheckCountry** node.

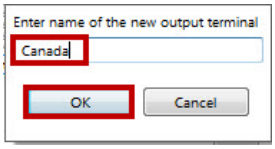
Click the right mouse button.

Select **Add Output Terminal** from the menu.



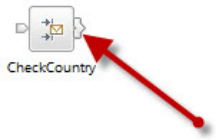
__16. Enter **Canada** as the name of the new output terminal.

Click **OK** to continue.



__17. Notice that the shape of the CheckCountry node has changed slightly.

Once there are too many terminals to display discretely, they will be grouped together.



__18. Change the name of the **HTTP Reply** node to “Reply if US.”

Press **Enter** to complete the rename operation.

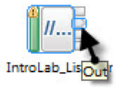
__19. Change the name of the **HTTP Reply1** node to “Reply if Canada.”

Note:
Two HTTPReply nodes are not actually necessary – a single one could have been used. But separate nodes were used here for clarity, as well as to make the flow more self-documenting.

__20. Press **Enter** to complete the rename operation.

__21. Another way to make a connection is just to click the terminal itself.

Try this by selecting the **Out** terminal (second from the top) of the IntroLab_Listener node.



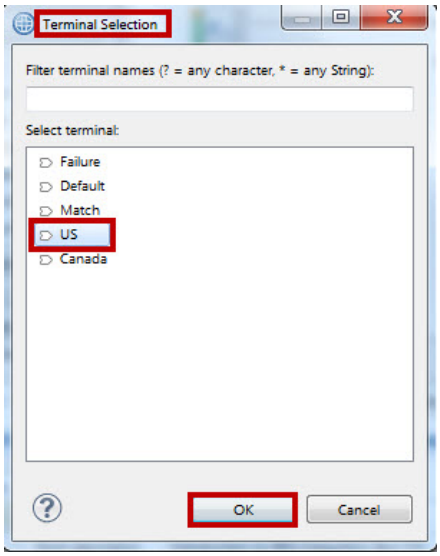
__22. Wire the Out terminal of the **IntroLab_Listener** node to the In terminal of the **CheckCountry** node.

__23. Hover the mouse pointer over the connector to verify that the correct terminals are wired.

__24. Click the group of output terminals on the CheckCountry route node.

__25. A Terminal Selection list will appear.

Select **US** and click **OK**.

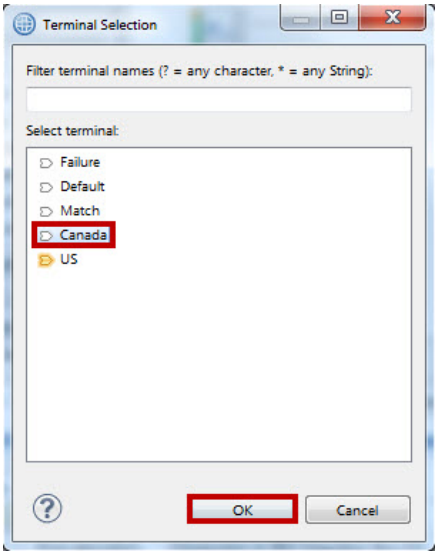


__26. Verify that the correct terminals are wired.

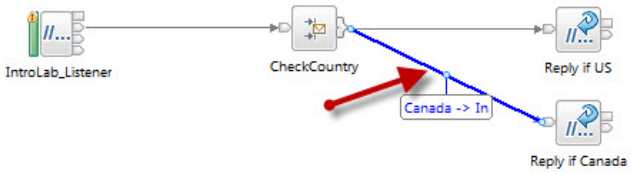
__27. Again click the group of output terminals on the CheckCountry route node.

__28. Select **Canada** from the list.

Click **OK**.



__29. Verify that the correct terminals are wired.

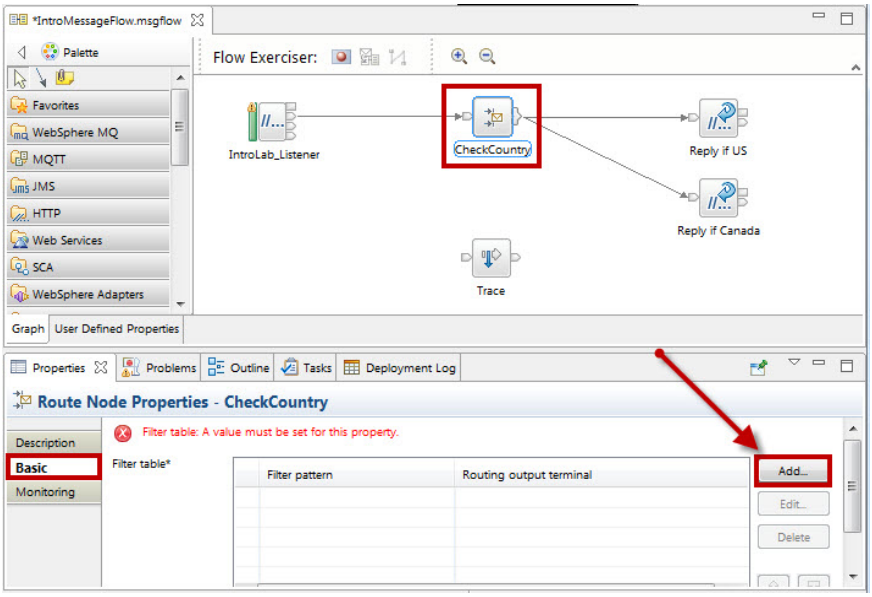


The criteria to be used by the CheckCountry routing node must now be specified.

__30. Select the **CheckCountry** node.

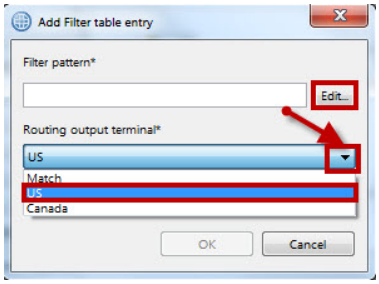
In the Properties pane select the **Basic** tab.

Click **Add**.



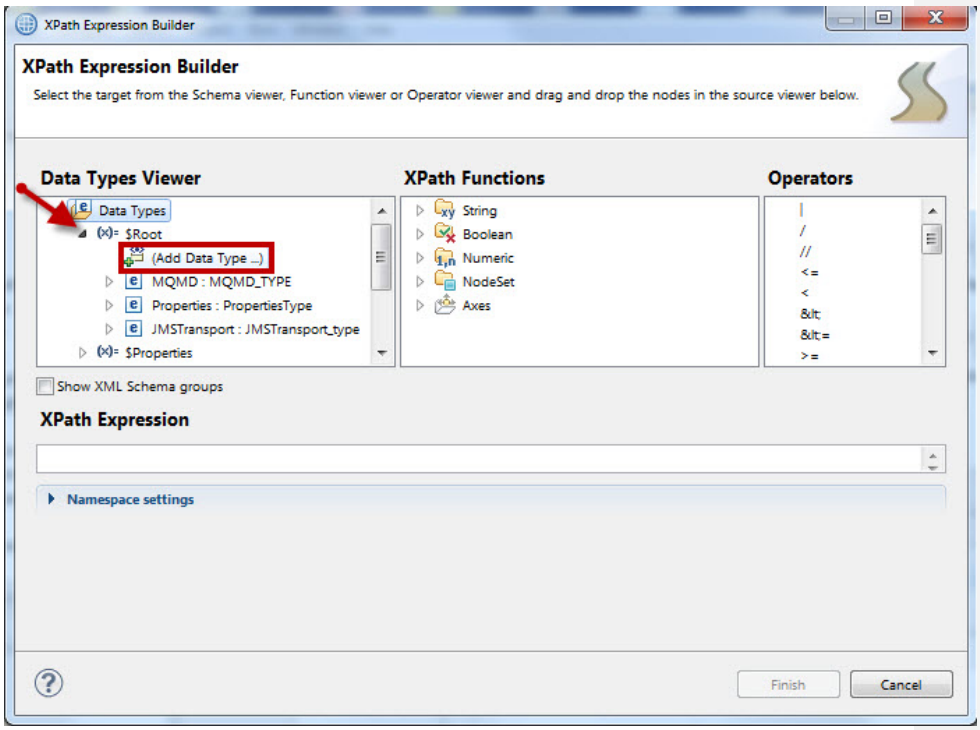
__31. Use the drop down menu to select the **US** terminal as the **Routing output terminal**.

Click **Edit**.



__32. Expand **$Root**.

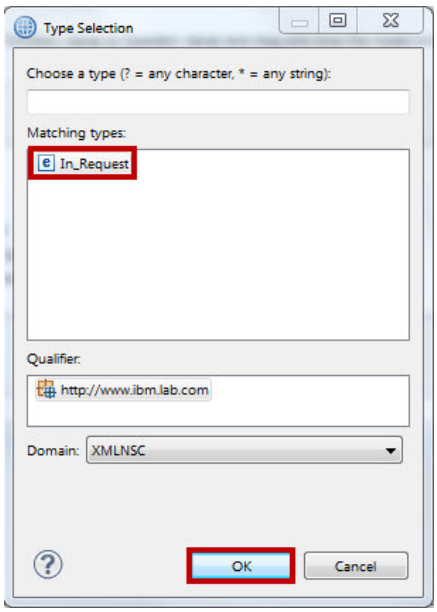
Select **(Add Data Type)**.



__33. Select **In_Request**.

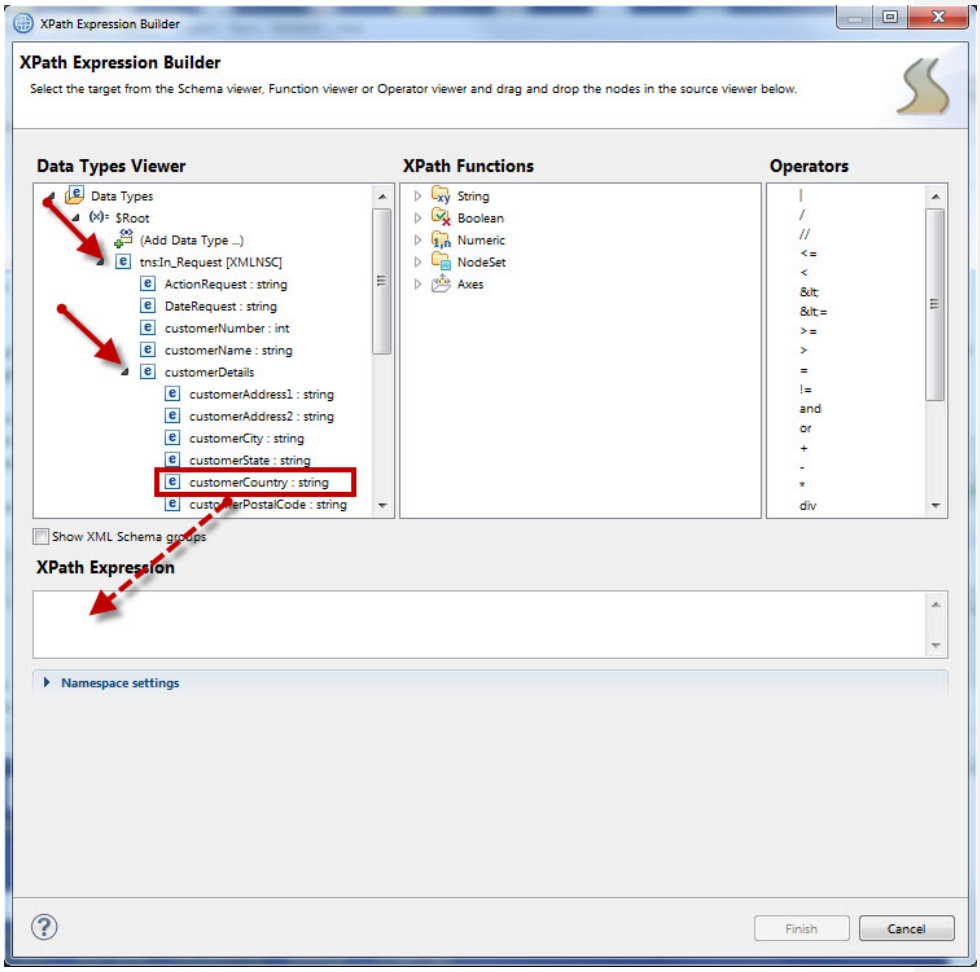
Note:
If you do not see **In_Request** in the list, then you must add a reference to the library by right-clicking the IntroLab application and selecting **Manage Library references**.

Click **OK** to continue.



__34. Expand the **tns:In_Request > customerDetails** elements.

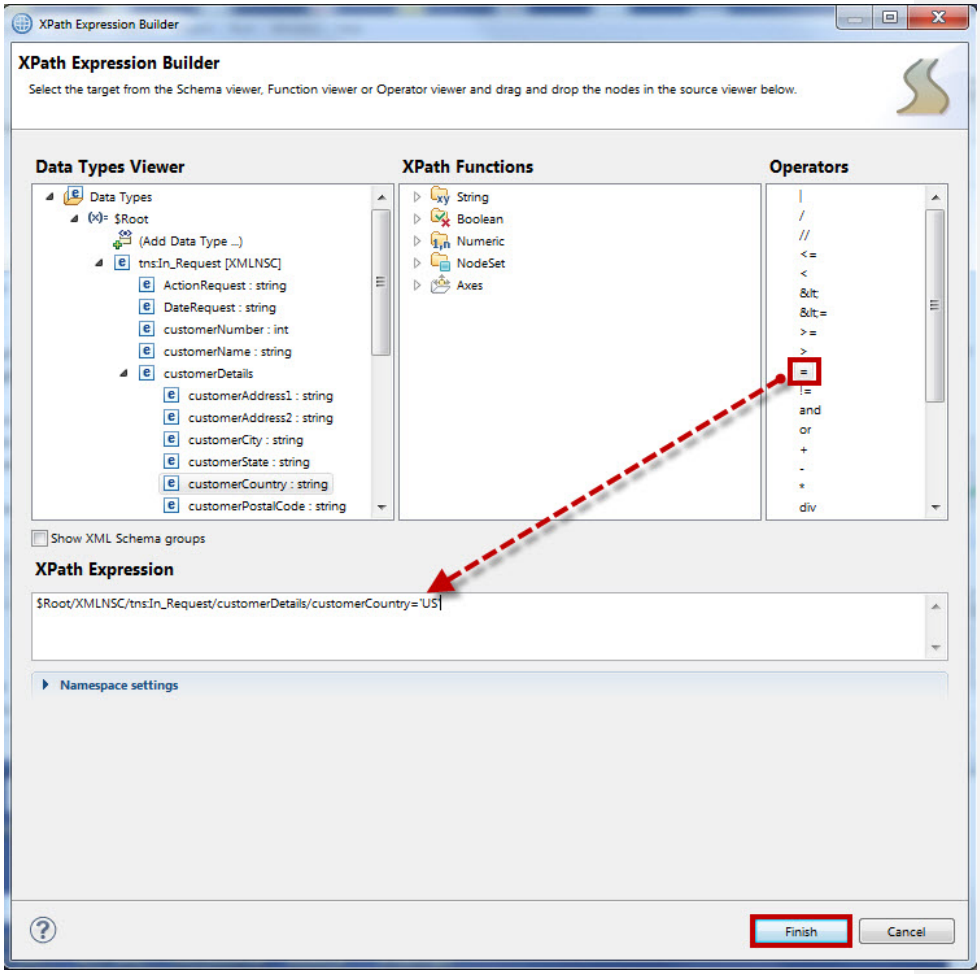
Select the **customerCountry** element and drag it into the **XPath Expression** dialog box.



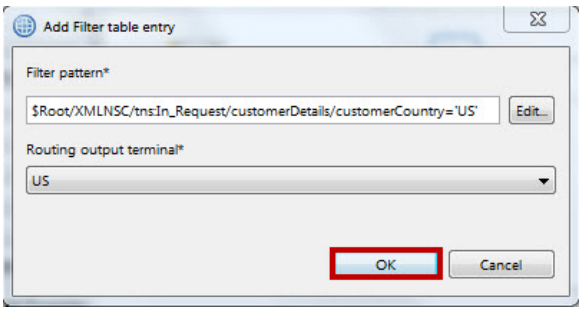
__35. Drag an equal sign from the Operators pane to the end of the expression.

Append the letters **'US'** (including the single quotes) after the equal sign.

Click **Finish** to complete the XPath expression.

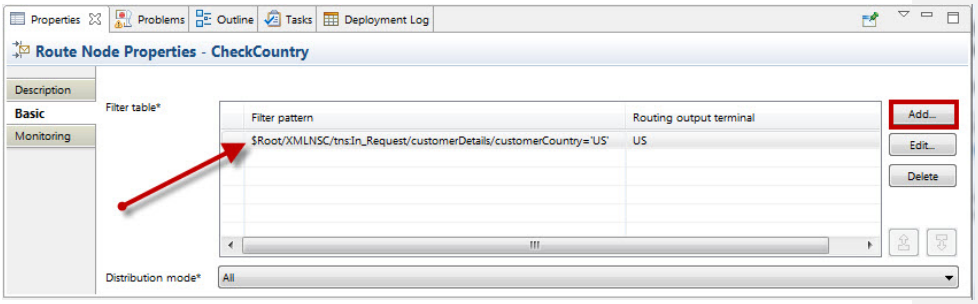


__36. Click **OK** to complete the Filter table entry.



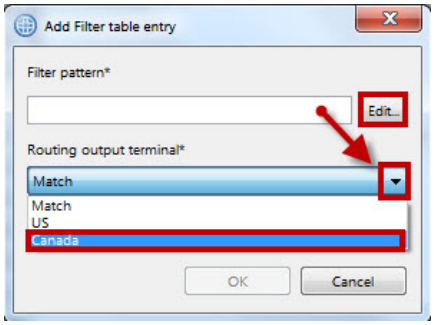
The filter pattern for the US terminal should now be visible. The process will now be repeated to create a Filter table entry for the Canada terminal.

__37. Click **Add** to enter a filter pattern for the Canada terminal.



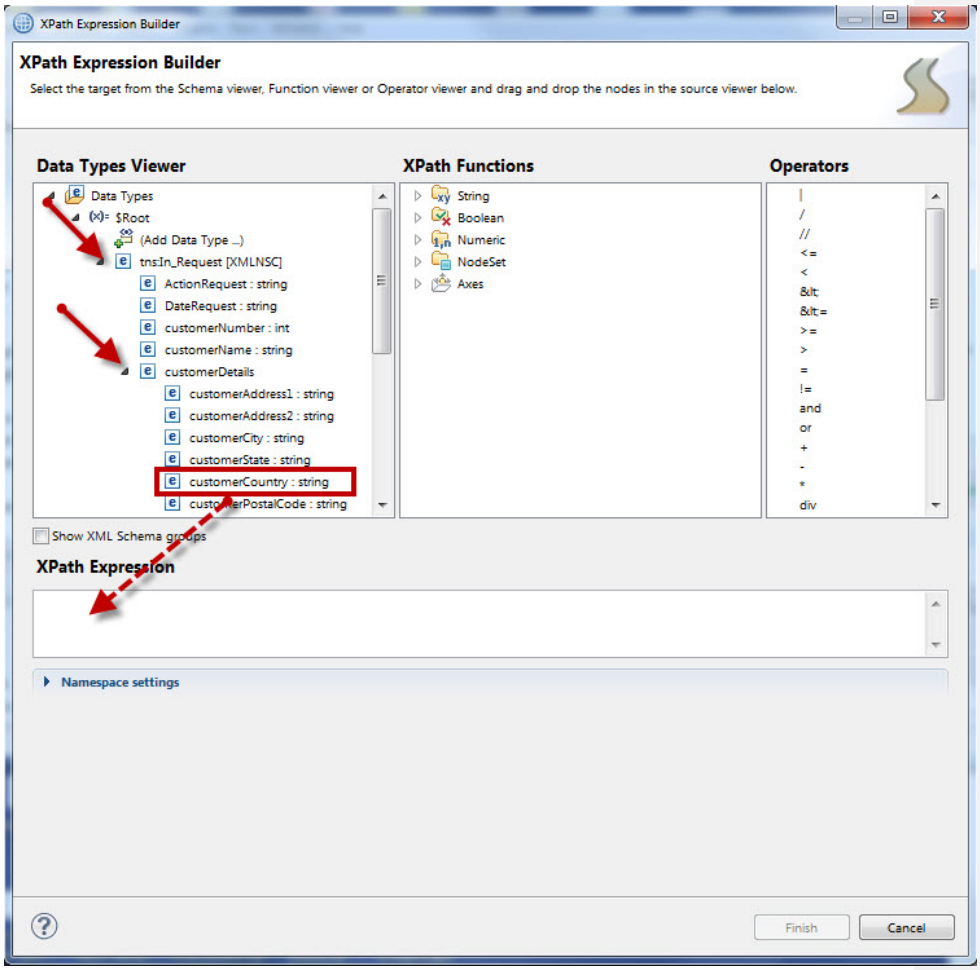
__38. Use the drop down menu to select the **Canada** terminal as the Routing output terminal.

Click **Edit**.



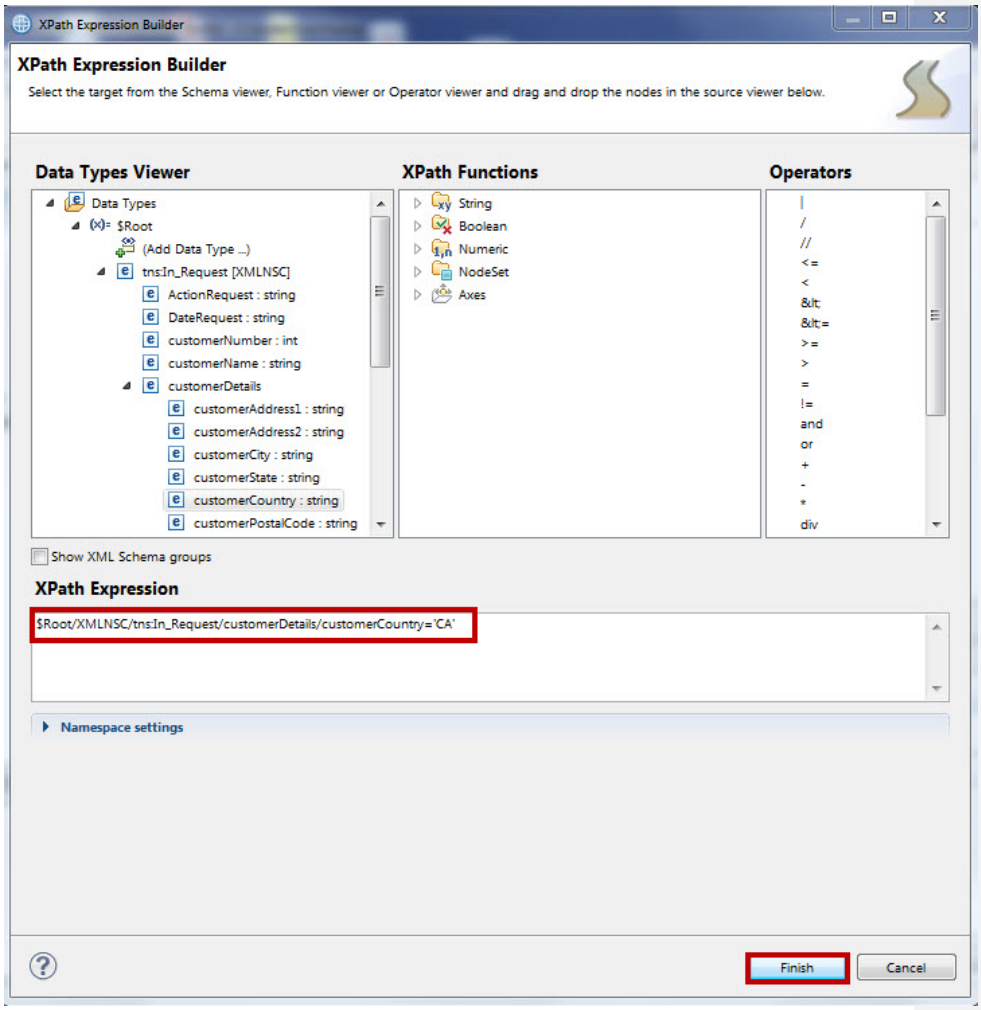
__39. Expand **Root > tns:In_Request→customerDetails**. Tip: If you do not see In_Request in the viewer, re-add it as explained in steps 31-32.

__40. Select the **customerCountry** field and drag it to the XPath Expression window.

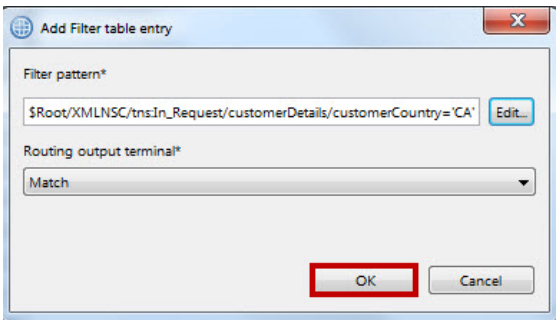


__41. Complete the XPath Expression by typing **='CA'**.

Click **Finish** to complete the XPath expression.

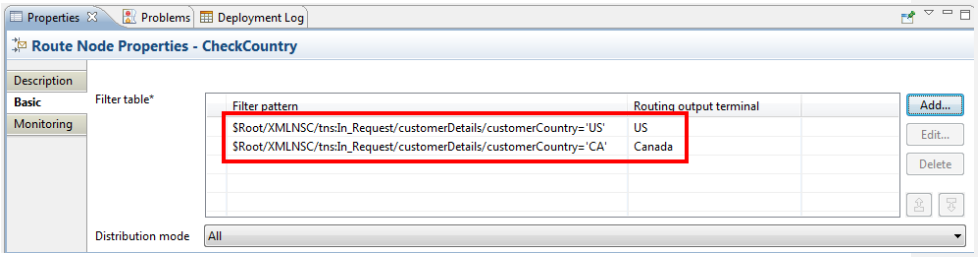

XPath Expression Builder
XPath Expression Builder
Select the target from the Schema viewer, Function viewer or Operator viewer and drag and drop the nodes in the source viewer below.
Data Types Viewer
XPath Functions
Operators
Data Types
$Root
(Add Data Type ...)
tns:In_Request [XMLNSC]
ActionRequest : string
DateRequest : string
customerNumber : int
customerName : string
customerDetails
customerAddress1 : string
customerAddress2 : string
customerCity : string
customerState : string
customerCountry : string
customerPostalCode : string
String
Boolean
Numeric
NodeSet
Axes
|
/
//
<=
<
<
<=
>=
>
=
!=
and
or
+
-
*
div
Show XML Schema groups
XPath Expression
$Root/XMLNSC/tns:In_Request/customerDetails/customerCountry='CA'
Namespace settings
Finish
Cancel

__42. Click **OK** to complete the Filter table entry.



The filter pattern for the Canada terminal should now be visible.

Verify that the two Filter table entries appear as shown below.



The updates to the message flow are now complete.

__43. 

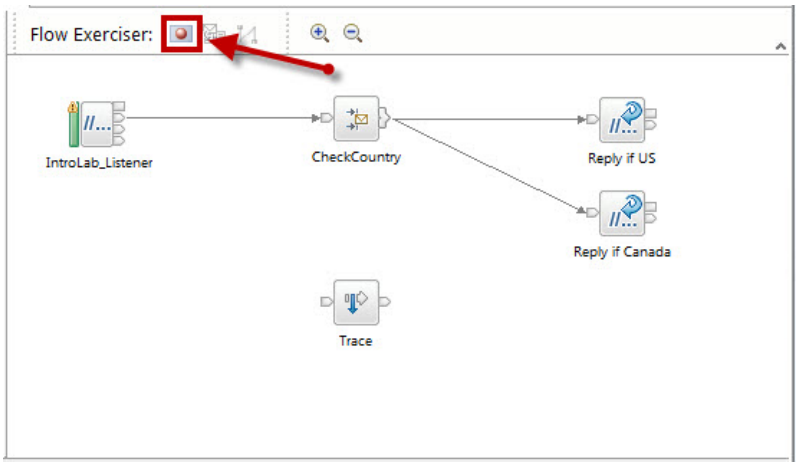
Save the message flow.

2.3 Test content-based routing using the Flow Exerciser

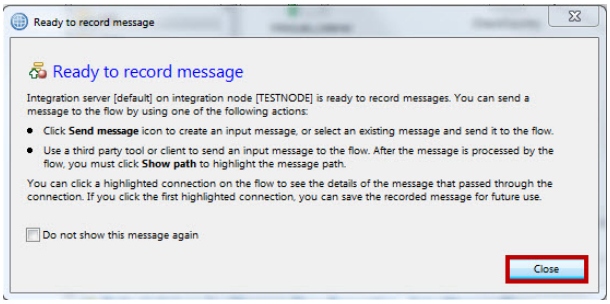
You will use the Flow Exerciser to test the flow. In this lab you will see some additional features of the Flow Exerciser that make it a very useful tool when testing and debugging more complex message flows.

__44. Start the Flow Exerciser.

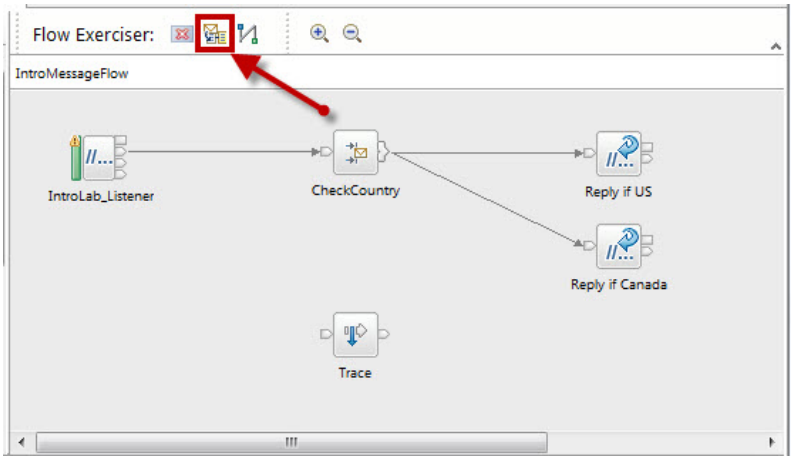
Note: The Trace node is still unwired. But this will not cause any problems.



__45. Click **Close** to start recording.

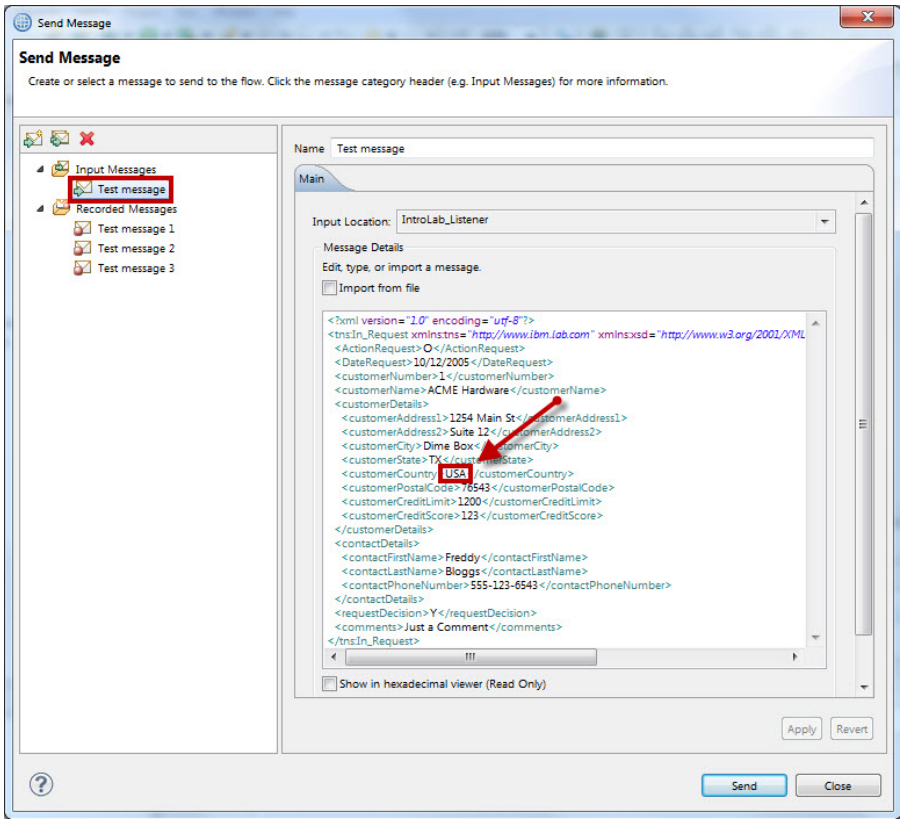


__46. Click the **Send** message icon.



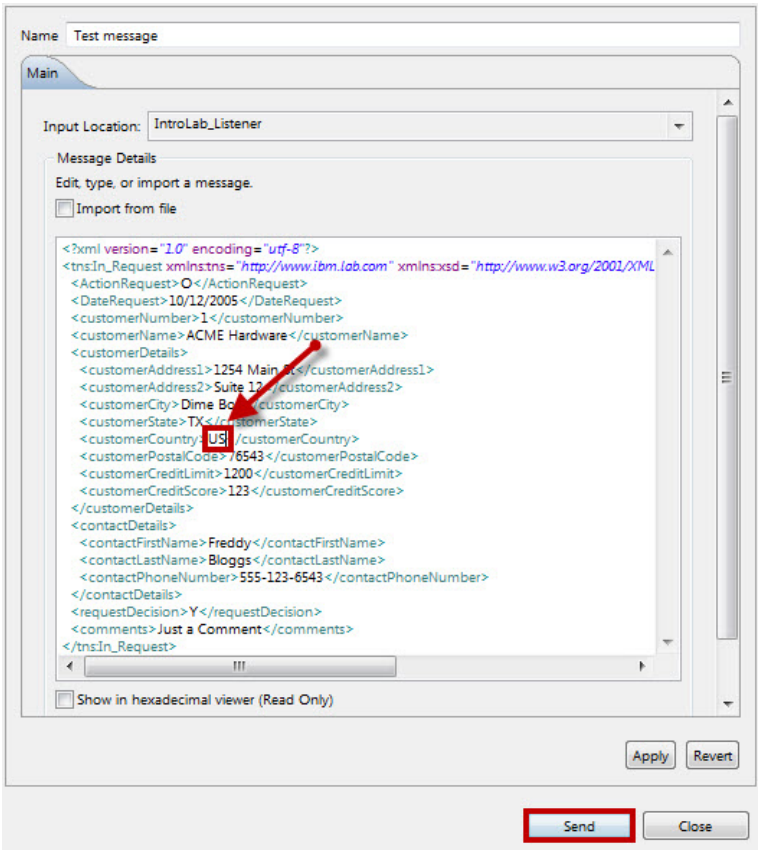
__47. The Input message from the previous tests is still available. Click **Test message**.

In the Test message, locate **customerCountry**. The value should be USA.

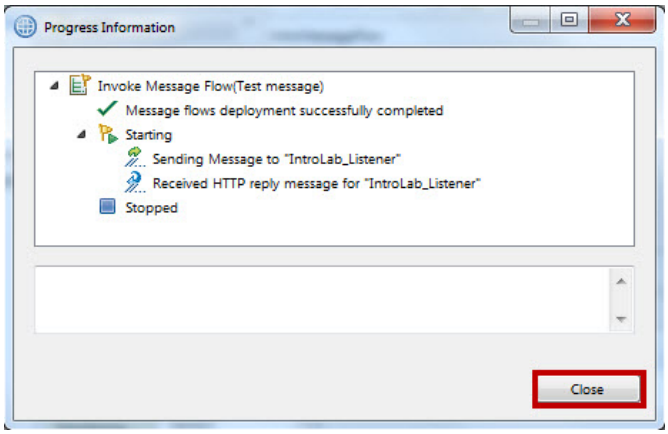


__48. Change the value of **customerCountry** to “US”.

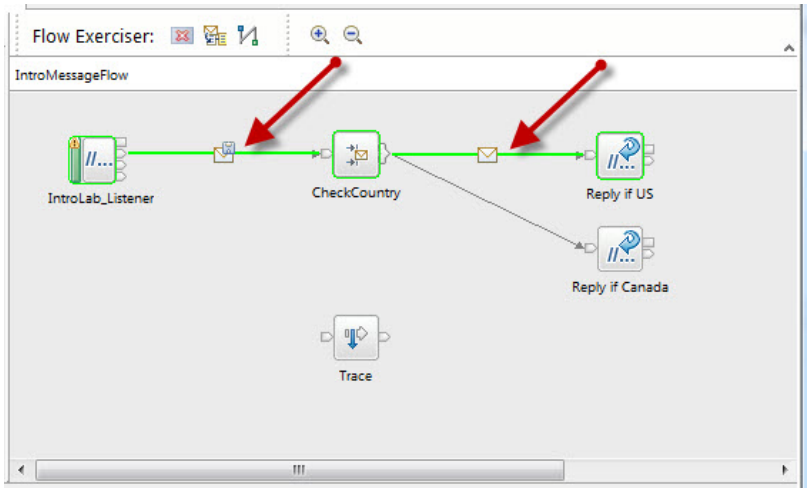
Click **Send**.



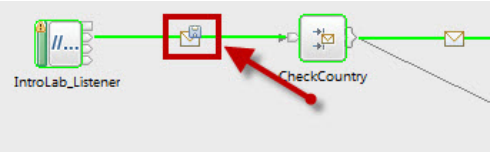
__49. The Flow Exerciser will run. Click **Close** after it stops.



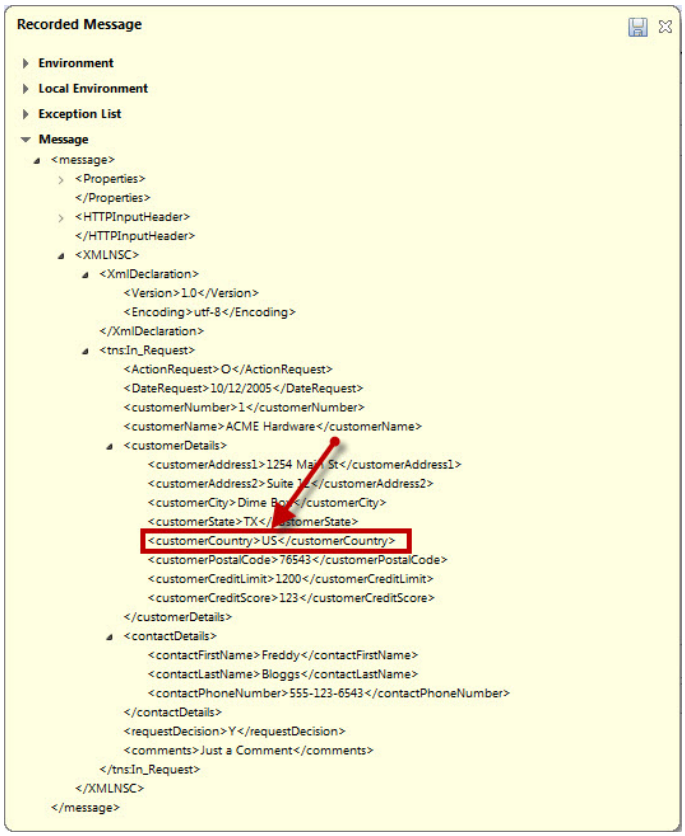
__50. The Flow Exerciser shows the path the **US** message took through the flow.



__51. Click the recorded message to view it.



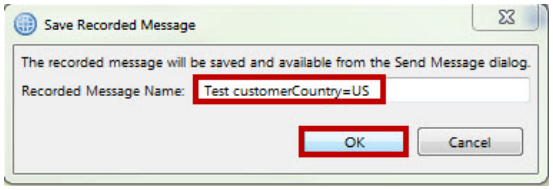
__52. Note that the Recorded Message does reflect the change you made.



__53. Save this recorded message by clicking the save icon in the upper right corner.

__54. Call this recorded message “Test customerCountry=US.”

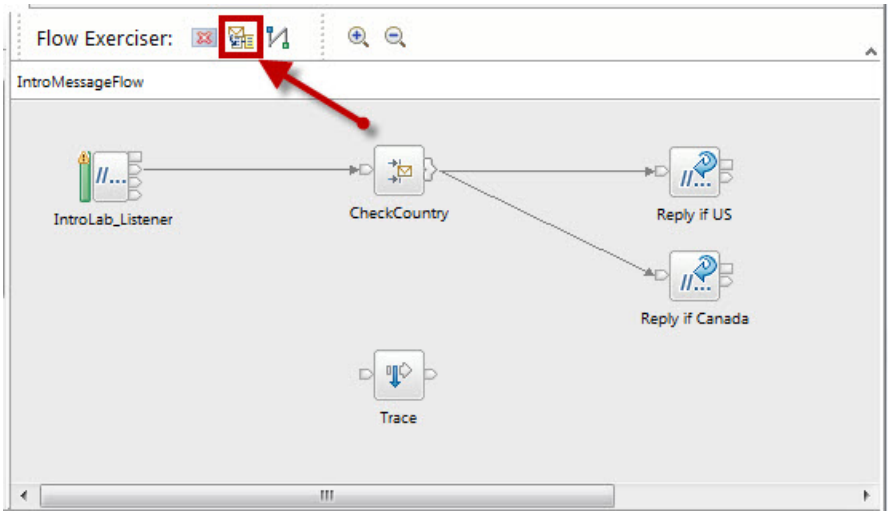
Click **OK**.



__55. Close the Recorded Message window.

__56. Repeat these steps to test the Canada path in the message flow.

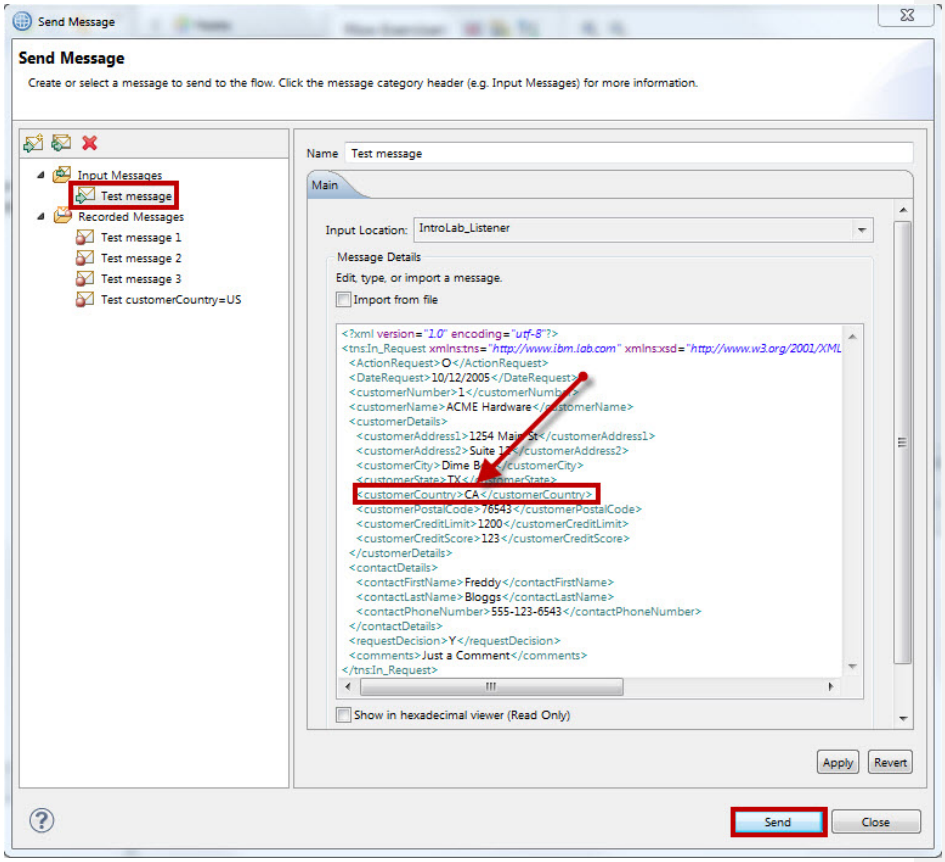
Click **Send** to configure another message.



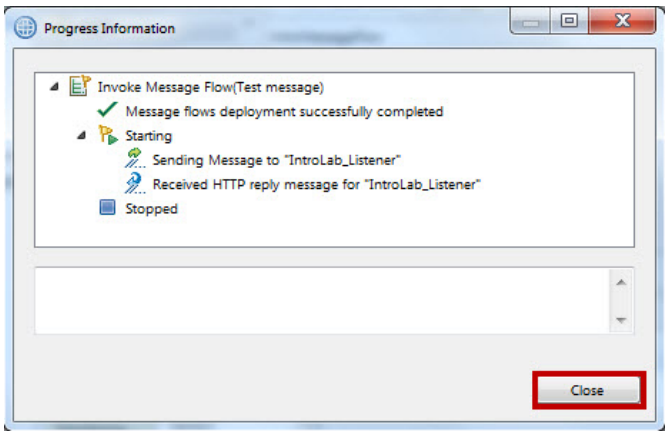
__57. Select **Test message**.

Change customerCountry to “CA”

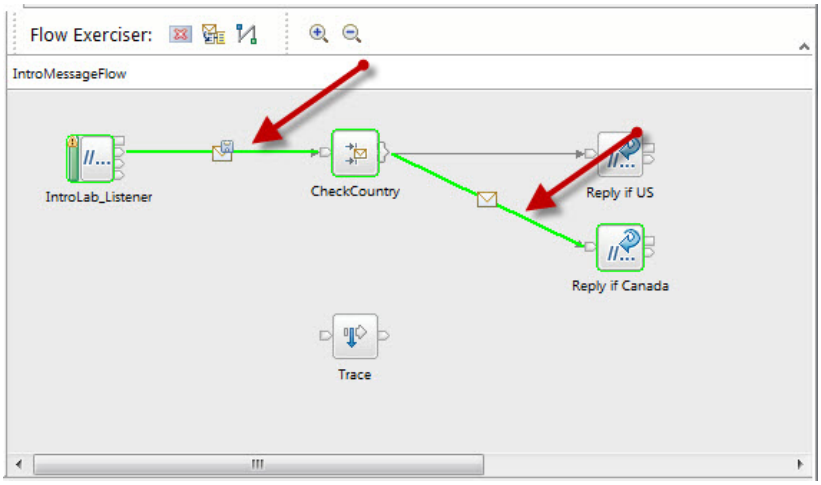
Click **Send**.



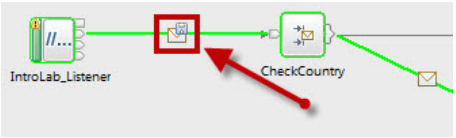
__58. The Flow Exerciser will run. Click **Close** after it stops.



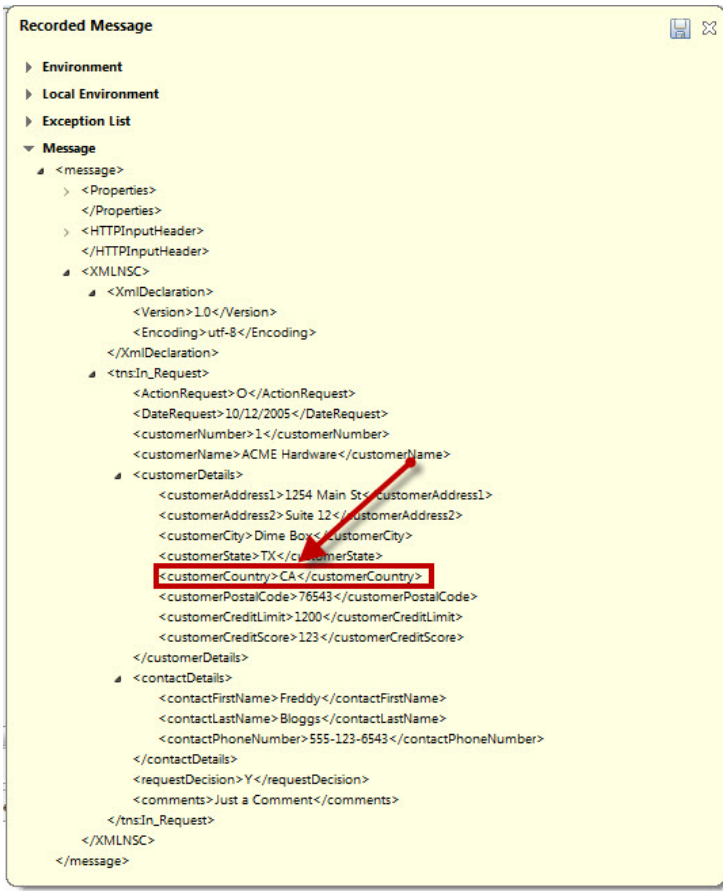
__59. This time the Flow Exerciser shows that the **CA** path was taken.



__60. Click the recorded message to view it.



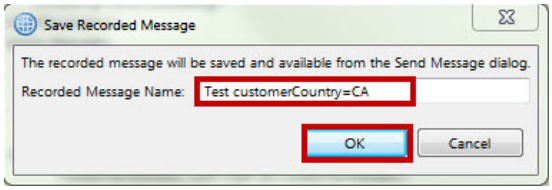
__61. Note that the Recorded Message does reflect the change you made.



__62. Save this recorded message by clicking the save icon in the upper right corner.

__63. Call this recorded message “Test customerCountry=CA.”

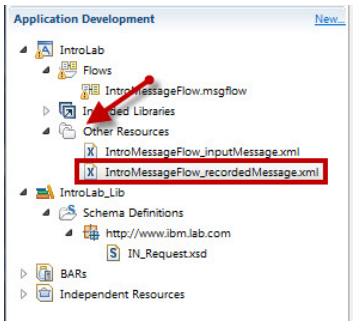
Click **OK**.



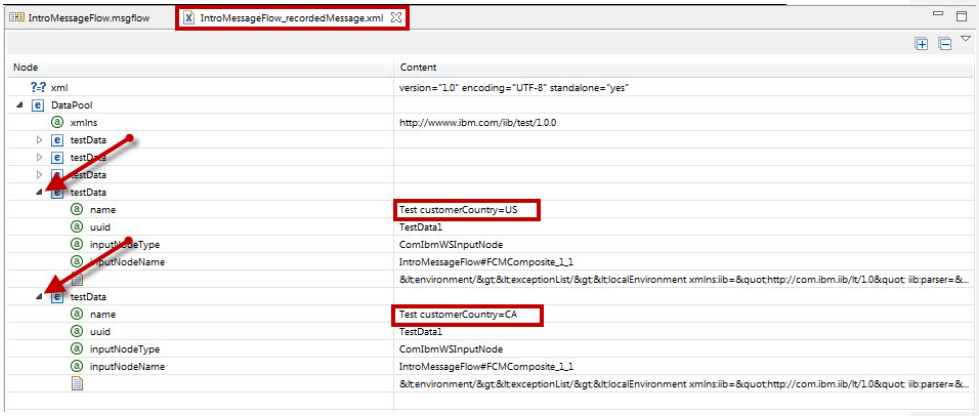
__64. Close the Recorded Message window.

__65. In the Application Development pane, expand **Other Resources**.

Double-click **IntroMessageFlow_recordedMessage.xml** to open it.



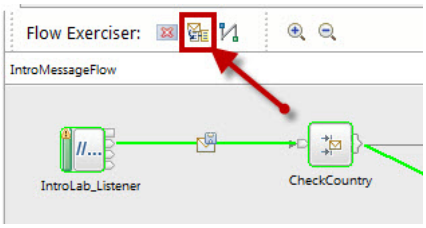
__66. In the XML Editor, expand the two most recent **testData** entries. You should see the recorded messages for US and CA.



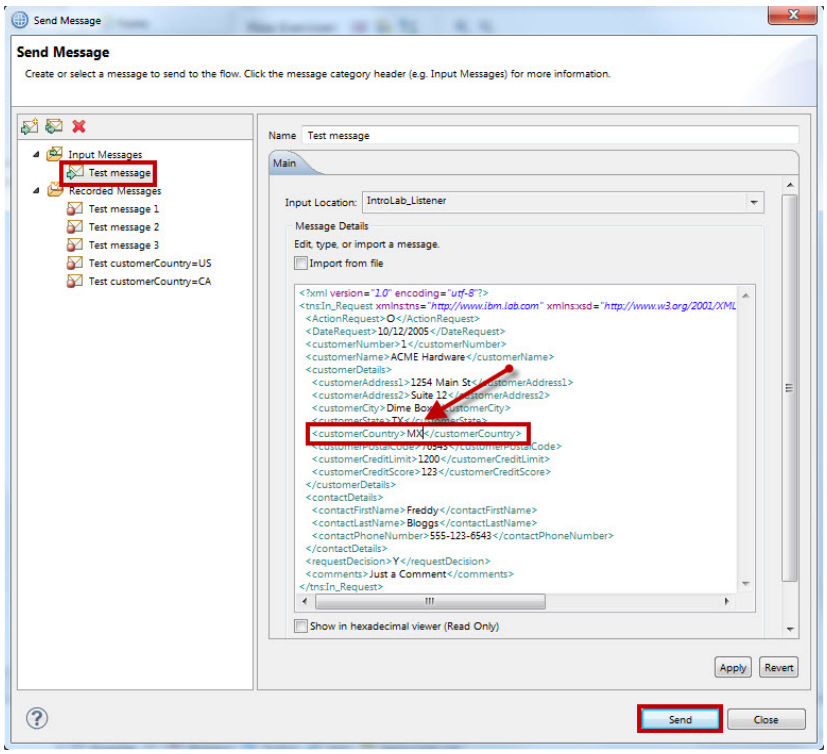
__67. Close the XML Editor.

__68. You have tested your two expected paths (US and CA). But what will happen if some other customerCountry value is received by the message flow?

__69. Click **Send** to configure another message.

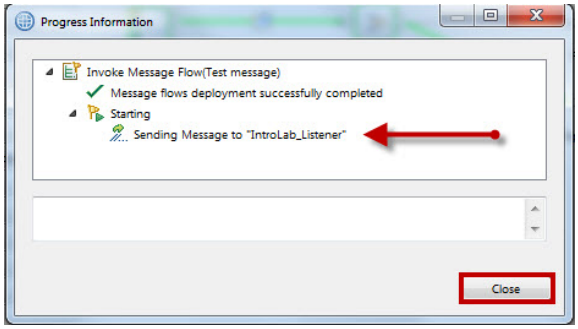


__70. Click **Test message**. Change the customerCountry to “MX”. Click **Send**.

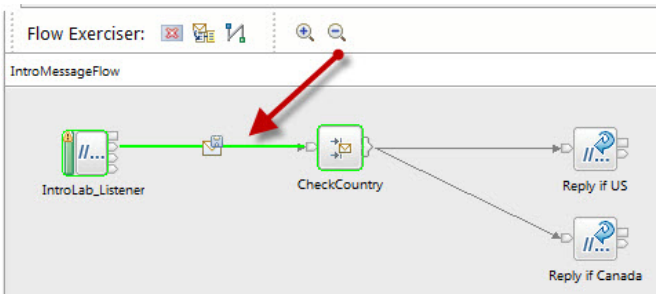


__71. Note that the message was sent but no HTTP Reply was received.

Click **Close**.

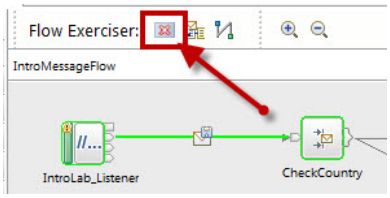


__72. In the Flow Exerciser, notice that control reached the CheckCountry node. But it stopped there. Why did it stop?

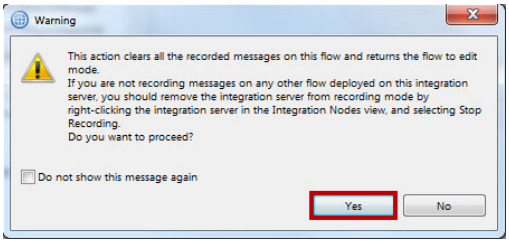


__73. To answer that, look again at the flow.

Stop the Flow Exerciser.

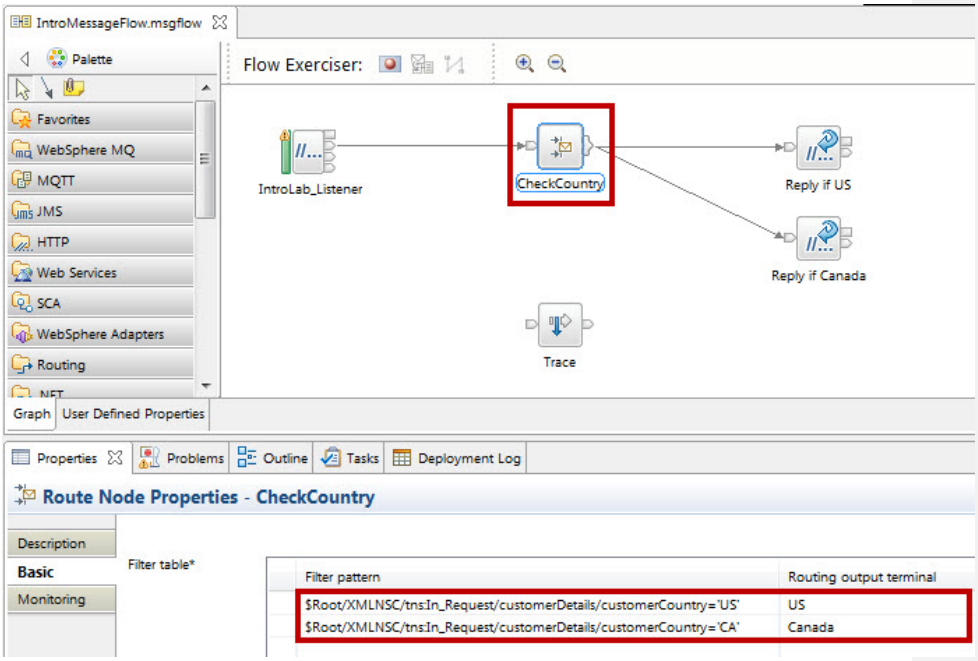


__74. Click **Yes** on the pop-up.



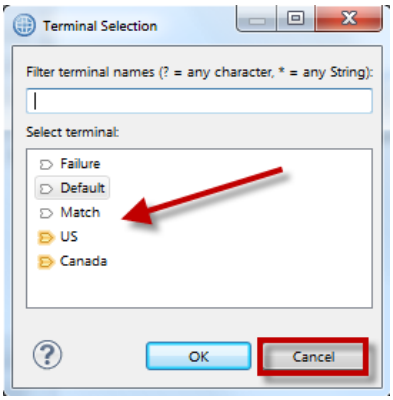
__75. The Flow Exerciser is stopped, and the message flow is returned to editable mode.

Click the **CheckCountry** route node. Only US and CA are handled.



__76. Click the **Output Terminal** group.

__77. Review the available output terminals.



- US and Canada are wired. But MX is not going to match either of those.
- There was neither a match nor a failure.
- Default would be the terminal normally taken when there was no match. But this was not wired. Therefore, the message was discarded. This might not be what you want to happen.

One approach would be to wire the Default terminal to an “Unsupported Country” path.

But many things could potentially go wrong in a message flow. Rather than wire every possible exception path on every node in a flow, you will explore approaches to dealing with situations like this one in a later lab.

__78. Click **Cancel** to cancel the unattached connector.

In a later lab you will explore one approach for dealing with unexpected conditions such as this one, as well as how to implement some basic flow-level exception handling.

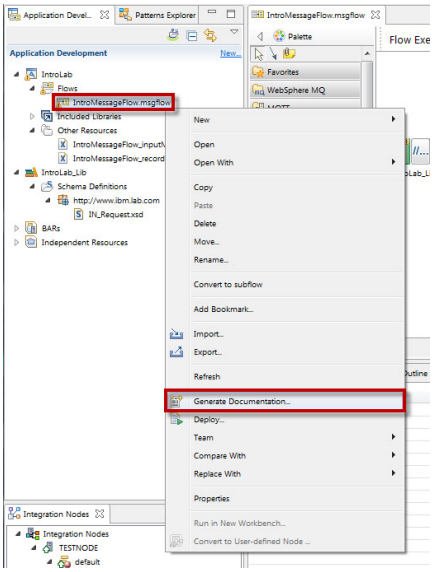
2.4 Generating documentation from your application or service

In this section, you will see how the App Connect Enterprise Toolkit can be used to generate documentation from application and service artifacts such as message flows.

The Document Generation feature of the toolkit uses information such as metadata from flows and other development artifacts, as well as description information specified by solution developers.

This documentation can be combined with test messages and results to create a package of documentation, testing artifacts and scenarios.

__79. In the Application Development pane, select the **IntroMessageFlow.msgflow** file, right-click the file, and select **Generate Documentation** from the menu.



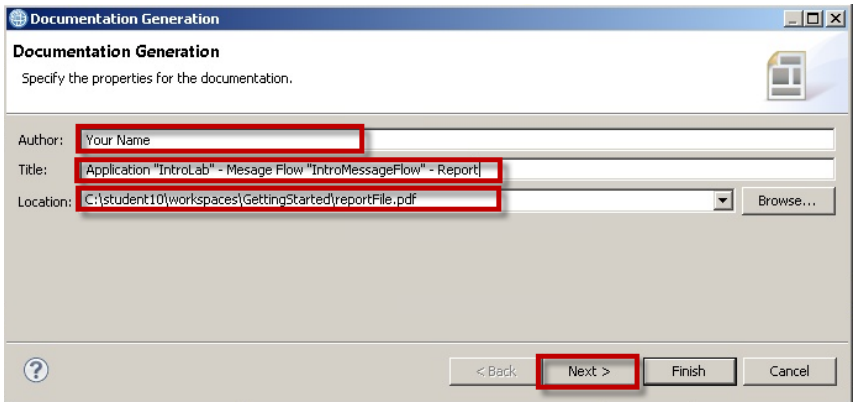
__80. In the Document Generation pop-up, specify:

a. **Author** (“Your Name”)

b. A suitable **Title**

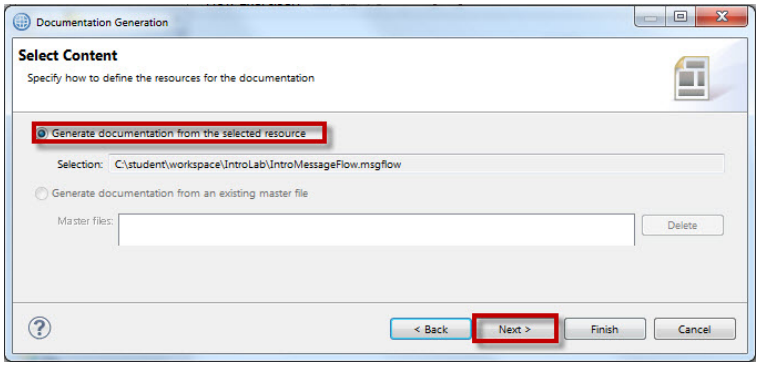
c. A **Location** and name for the generated file (by default this is the current workspace)

Click **Next.**

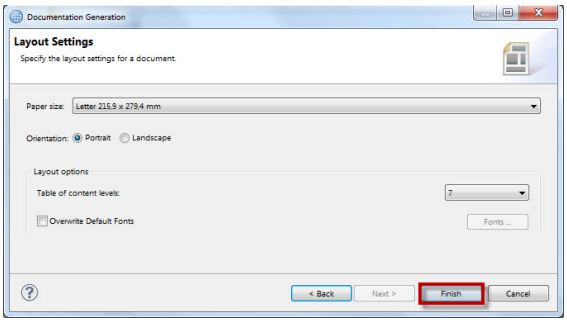


__81. Make sure **Generate documentation from the selected resource** is selected.

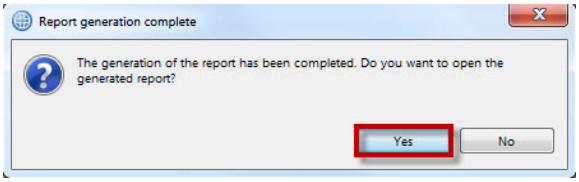
Click **Next.**



__82. Accept the default layout and click **Finish**.



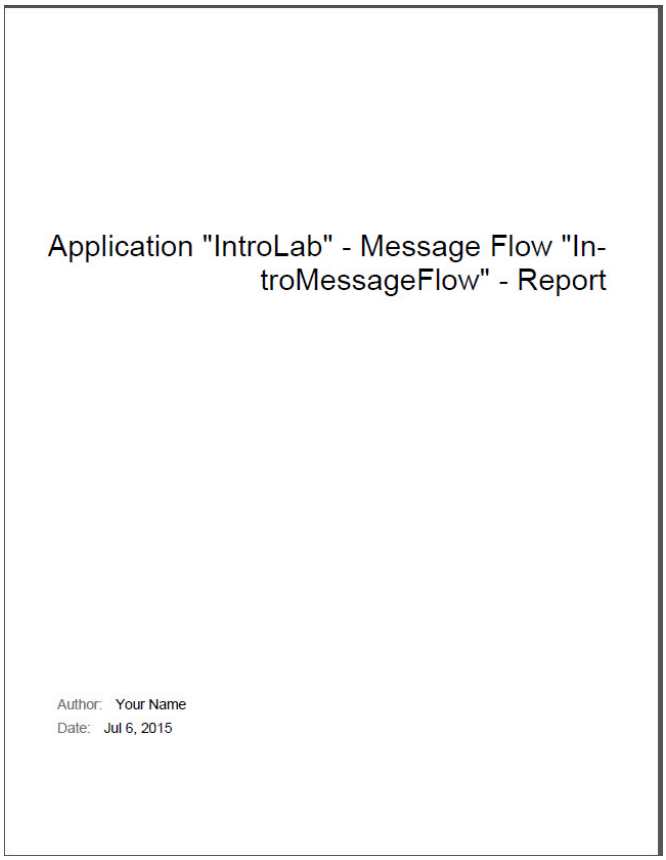
__83. Click **Yes** to view the generated report.



__84. Review the generated report (the first page of which is shown here).

Notice how the Toolkit uses both flow metadata as well as descriptions you entered, version numbers, and additional information to generate the report.

Many of these fields were left blank in the labs you did, but a best practice would be for you to make the fullest use possible of these fields, in order to make your integration solutions as well-documented and maintainable as possible.



__85. Close Adobe Reader.

The next section will take a closer look at the deployment process.

2.5 A closer look at the deployment process

In the first two labs, you have been utilizing the Flow Exerciser to initiate your unit testing and it has been handling the deployment process for you “behind the scenes”. To finish this lab, you will briefly examine the deployment process, and you will manually do your own deploy.

2.5.1 Key Idea: The deployment process

Key Idea: The deployment process

Deployment of integration solutions up to this point have been done *implicitly* – by virtue of your using the Flow Exerciser. This section will explore other approaches to solution deployment.

Commented [EL3]: Should this be "integration service"? There are a couple other instances in this document of referring to "integration solution" but it's not clear if they should all be "integration service" instead. Please verify.

When you create an application, Integration Service or REST API project in the IBM ACE Toolkit, you must distribute them to the Integration Nodes where you want them to run. Artifacts for message flows and associated resources are packaged in a broker archive (BAR) file before being deployed to the integration server.

You can initiate a deployment in the following ways:

- From the IBM ACE Toolkit
- From the IBM Integration Web UI
- By using the **mqsideploy** command
- By using functions defined by the IBM Integration API

Depending on your work patterns, you might use all these methods at different times.

The ACE Toolkit provides an Integration Nodes view in the lower left hand corner of the Integration Development perspective. If you expand an Integration Node, all the Integration Servers in that node are displayed, as well as deployed applications, services, REST APIs, and their underlying message flows and their associated resources. You can drag an application or library, message flow, or a broker archive (BAR) file from the Application Development view onto an Integration Server to deploy it. Alternatively, you can right-click an Integration Server to select an application or library, message flow, or BAR file to deploy to the selected Integration Server.

If you are working with an application and want to deploy and test it quickly, you can deploy just that resource. Drag the resource onto the Integration Server to which you want to deploy it. A BAR file is generated automatically and deployed. If static libraries are referenced, they are added automatically to the BAR file and deployed. If a message flow contains a subflow that is defined in a ".subflow" file, the subflow is automatically included in the BAR file, and deployed with the message flow. If you drag a flow that is contained in an application or library, you will see a message saying that the whole application or library will be deployed, because you cannot deploy a message flow on its own if it belongs to an application or library.

2.5.2 Key Concept: The BAR file

Key Concept: Broker archive (BAR) files.
The unit of deployment to an Integration Server is the Broker Archive (BAR) file. This section will explore what BAR files are and how you make use of them.

The Broker Archive (BAR) file is a .zip file that contains the flows, models, .jar files, maps, and any other resources in the workspace needed to run your applications. The BAR file also contains a deployment descriptor .xml file, which exposes flow and node properties for override at build or deploy time. The following sequence of events illustrates how to deploy with a BAR file:

1. Create a broker archive.
2. Add applications, libraries, services, REST APIs, message flows and other flow dependencies to the broker archive.
3. If necessary, edit the configurable properties of the message flows or applications in the broker archive.
4. Deploy the BAR file by sending it to an Integration Server.

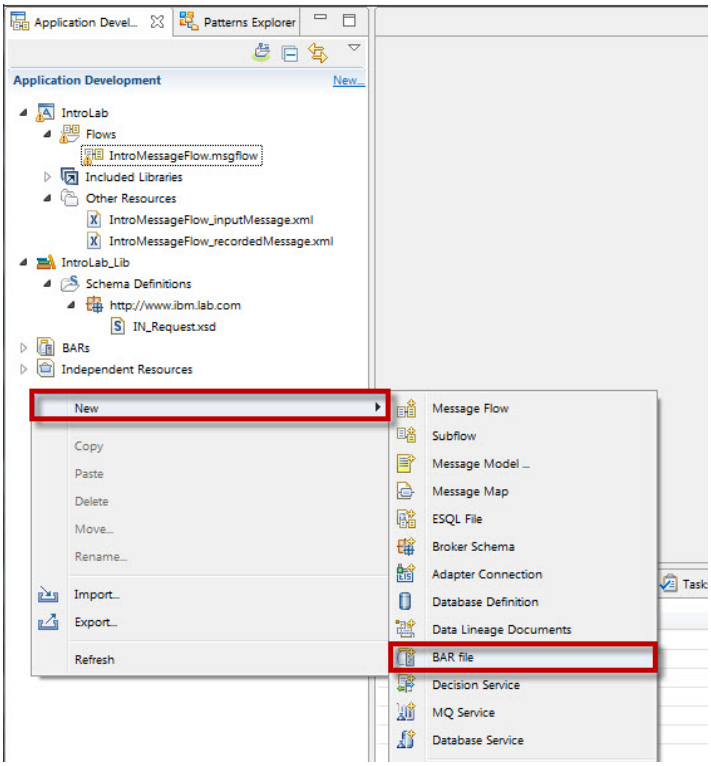
A BAR file can be deployed in two ways:

- Incremental BAR file deployment. Deployed files are added to the execution group. Files that exist in the execution group are replaced by the new version.
- Complete BAR file deployment. Files that are already deployed to the execution group are removed before the entire contents of the BAR file are deployed. Therefore, nothing is left in the execution group from previous deployments.

__86. Return to the ACE Toolkit.

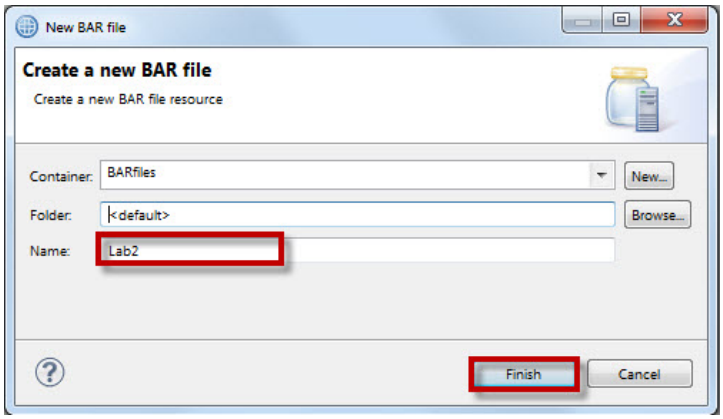
In the Application Development pane, right-click in the whitespace.

__87. Select **New > BAR file** from the menu.

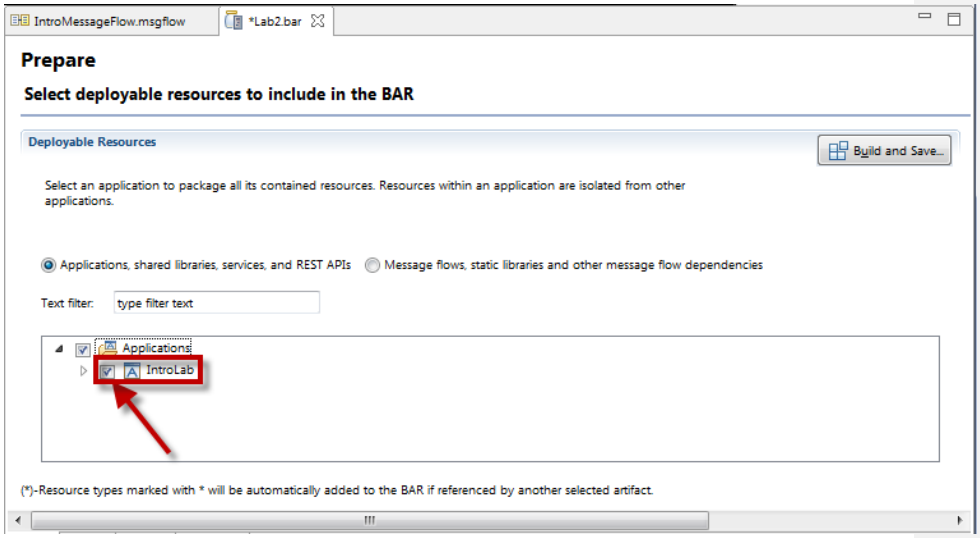


__88. Enter “Lab2” as the name of the new broker archive file.

Click **Finish**.

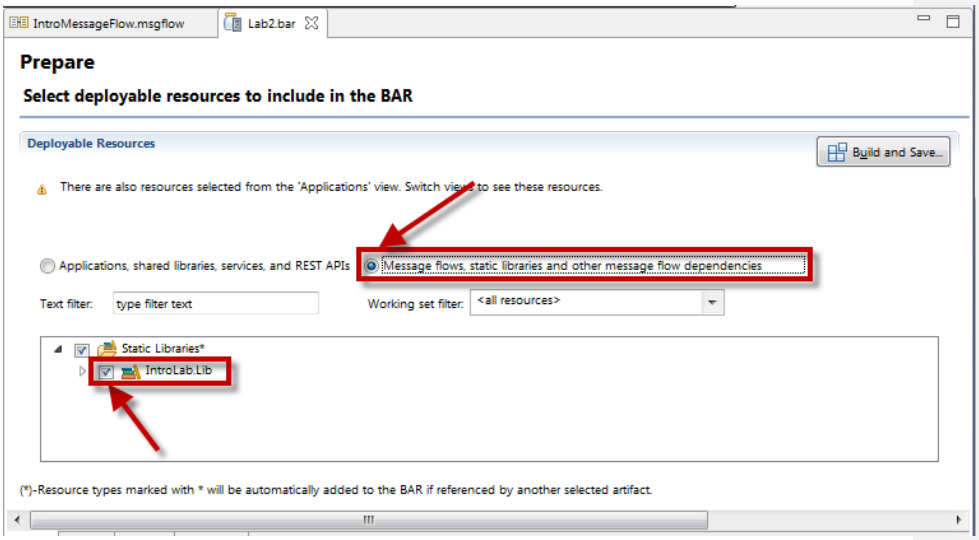


__89. In the BAR editor, select the **IntroLab** checkbox.

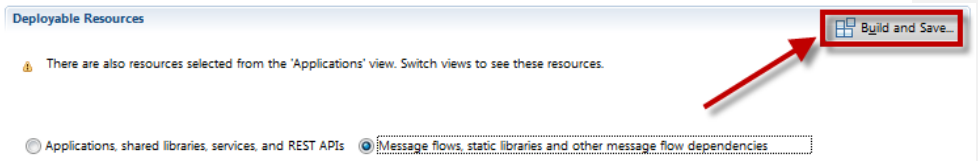


__90. Select the **Message flows, static libraries and other message flow dependencies** radio button.

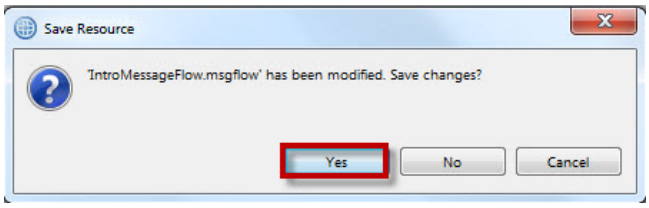
__91. Select the **IntroLab_Lib** checkbox.



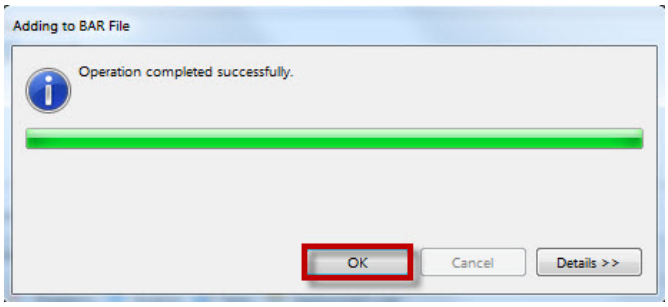
__92. Click the **Build and Save** button.



__93. If prompted to save the message flow, click **Yes**.



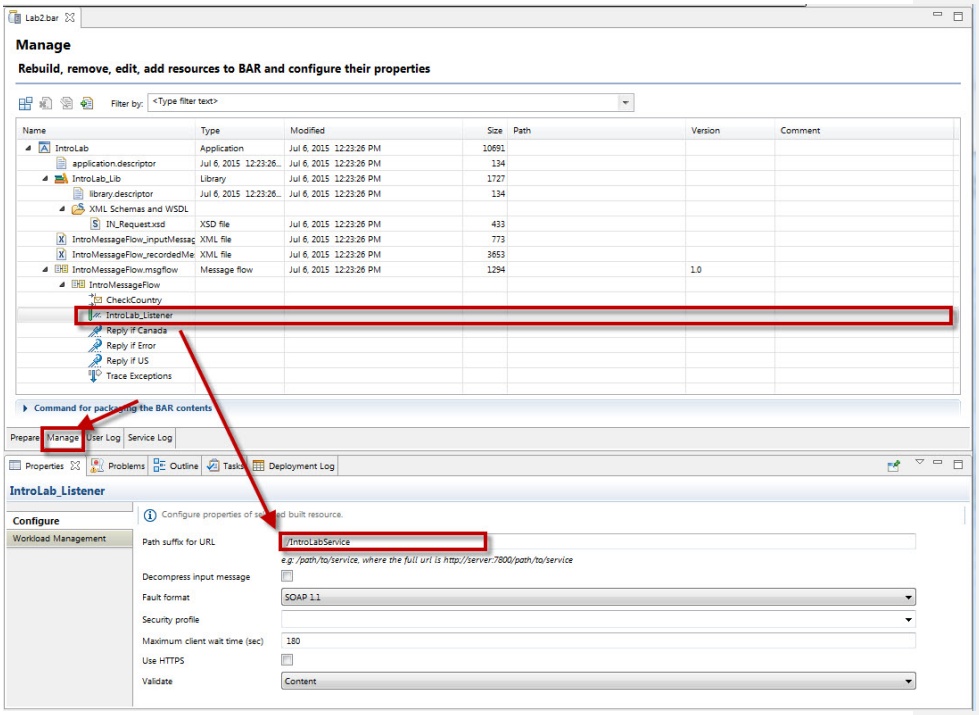
__94. Click **OK**.



__95. In the BAR editor, select the **Manage** tab.

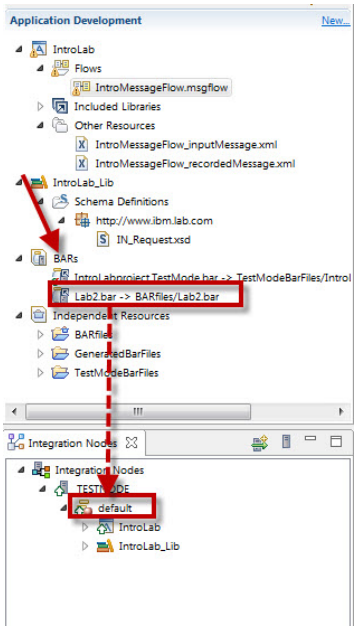
Expand the **IntroLab** app and select the various resources.

Look at the **Properties view** below to see what properties are exposed in order to be overridden within the BAR file. For example, select the **IntroLab_Listener** node. You can see that the "Path suffix for URL" property can be overridden at deployment time.

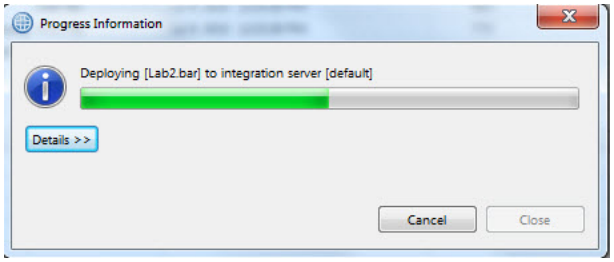


To deploy, find the **Lab2.bar** file in the navigator in the **BARs** container.

Drag and drop it onto the **myaceworkdir** integration server.



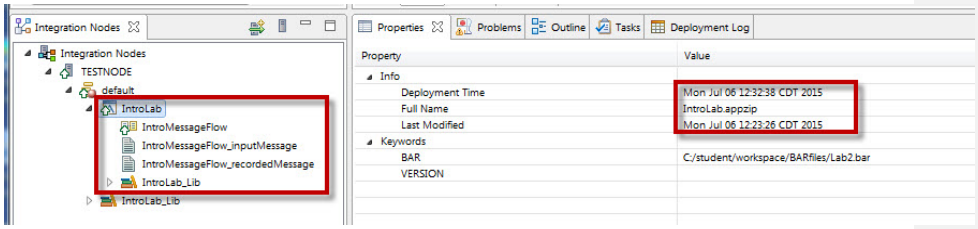
__96. Wait for deployment to complete.



__97. Once deployment is complete, click the **myaceworkdir** integration server to open it and see the assets deployed to it.

The IntroLab application has been deployed, which included the IntroMessageFlow message flow. The IntroLab_Lib static library, which contains the IN_Request XML Schema, has also been deployed.

Note too that the deployment date and time should reflect the actual time of deployment.



Close all the open editor tabs but leave the Toolkit running.

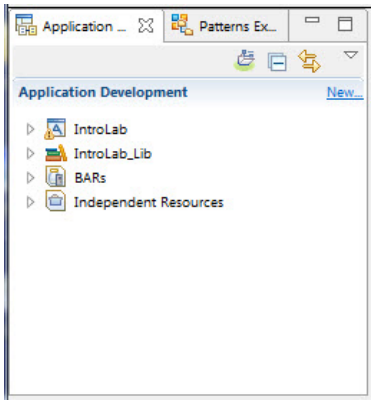
2.6 Lab clean-up

Optionally, you can clean up the resources you created in Labs 1 and 2.

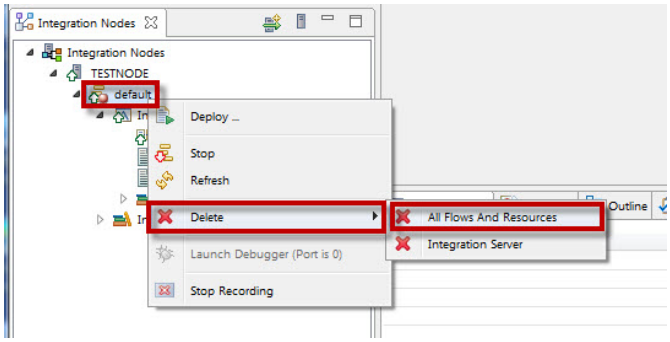
The following steps will delete the run-time artifacts from the server.

This will make things less cluttered while you continue through the remainder of the labs.

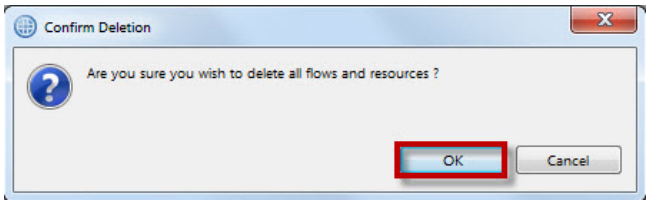
__98. In the Application Development pane, collapse any folders that are currently expanded.



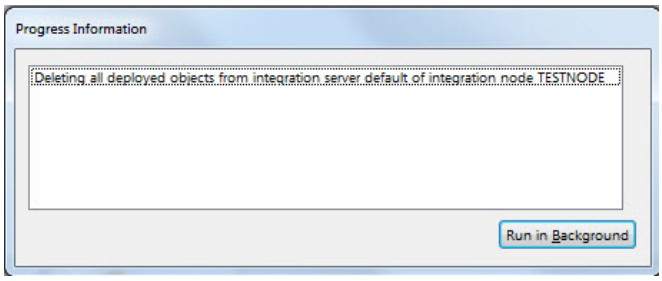
__99. In the Integration Nodes pane, right-click the **Integration Server**, and select **Delete > All Flows And Resources**.



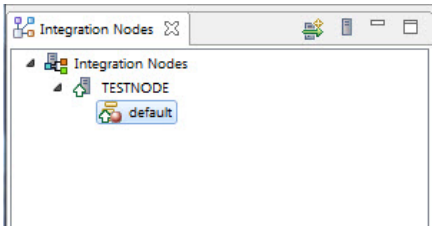
__100. Click **OK** to proceed.



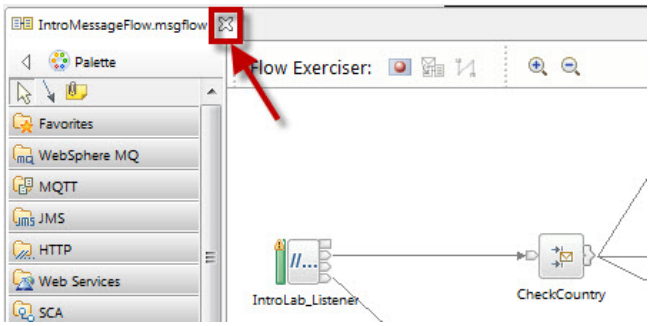
__101. Wait while the myaceworkdir integration server is cleared.



__102. All deployed objects should be cleared from the Integration Server.



__103. Close all the open editor tabs but leave the Toolkit running.



END OF LAB 2

Lab 3 Using patterns to work with files

In this lab will you will learn how to start from, configure and instantiate a pattern. You will also implement file processing, content-based routing and file-to-HTTP protocol switching. The lab implementation reads JSON files, routes them based upon the value of a `country` field, and calls an HTTP web service that can be provided either by a Node.js application or an ACE application.

You will use the integration server named **`myaceworkdir`**.

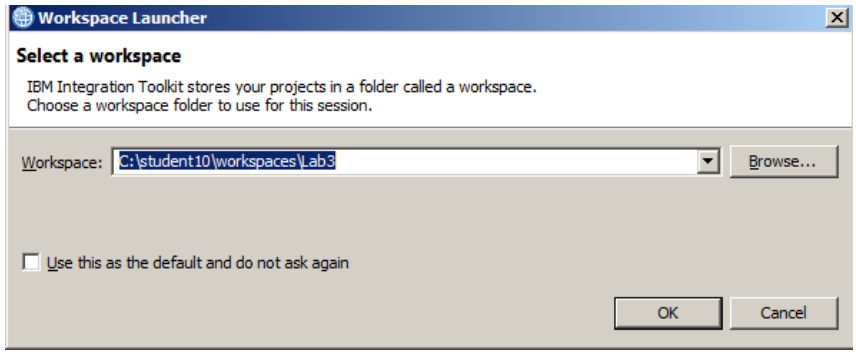
3.1 Selecting and configuring the filesystem location

We need to create directories on a local filesystem, where ACE will read and write the input file. We suggest that location to be the filesystem path of your integration server (c:\myaceworkdir). Inside that directory, create a subdirectories named `out` and `file_to_http_sample`. If you chose a different base location, please contact your instructor, because there are certain changes to the flow configuration which need to be made in this case.

3.2 Preparing the ACE Toolkit workspace

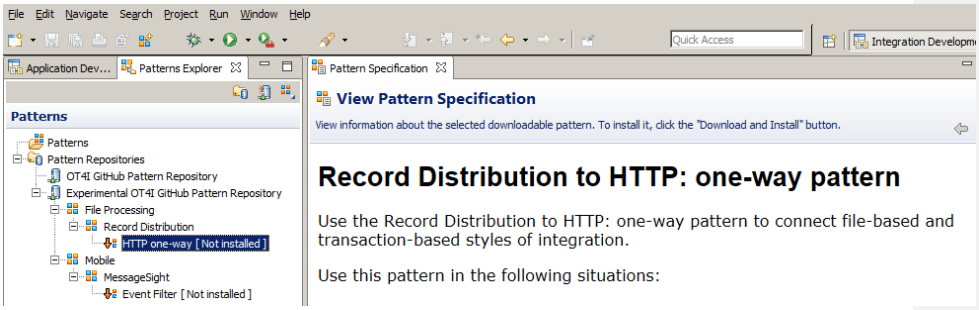
In the section you will prepare an empty ACE Toolkit workspace with the pattern.

__104. Double-click the desktop shortcut for the ACE V11 Toolkit. When prompted for a workspace, enter a path for an empty workspace then click **OK**.

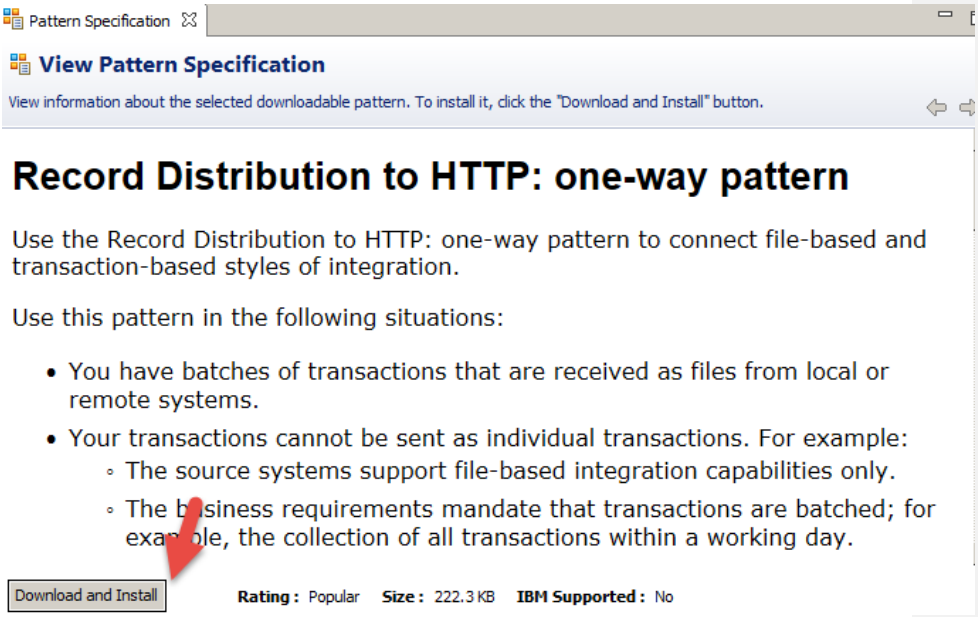


__105. Close the **Welcome** panel.

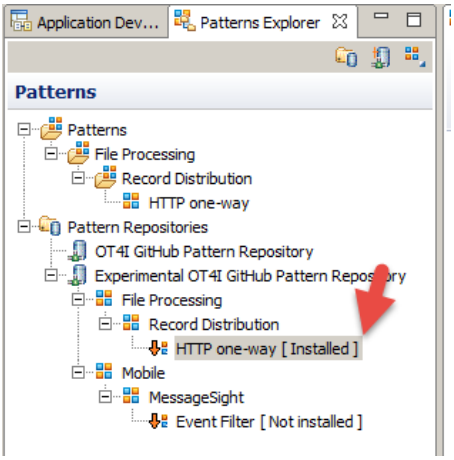
__106. In the Toolkit's upper left, click the **Patterns Explorer** tab. Then select **Pattern Repositories > OT4I GitHub Pattern Repository > File Processing > Record Distribution > HTTP one-way** to open the Pattern Specification editor for `Record Distribution to HTTP: one-way pattern`. The Patterns Explorer view should indicate that this pattern currently is not installed.



__107. In the lower right of the Pattern Specification editor, click **Download and Install**.



__108. Let the download complete. If prompted with a security warning, click **OK** to proceed. Upon completion of the download, the Patterns Explorer should indicate that the HTTP one-way pattern is now installed.



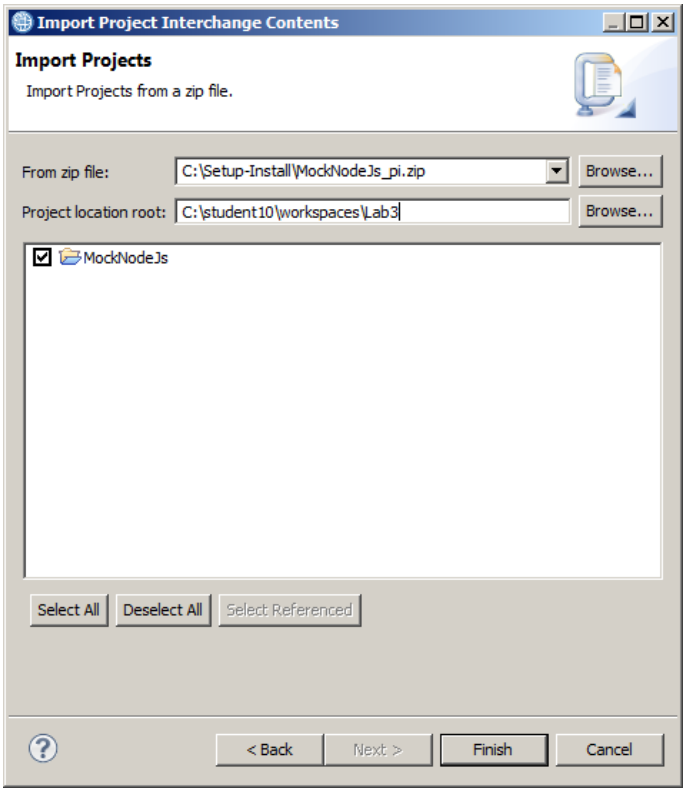
__109. Read all of the information on the Pattern Specification view for this pattern, including the links in the Message Flows and Related Tasks sections.

3.3 Deploying the MockNodeJs ACE application

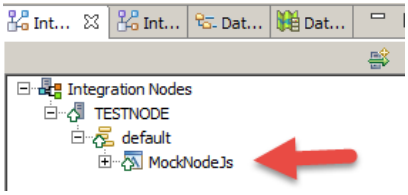
Follow the instructions in this section to deploy an ACE application that serves as a server for this lab.

__110. On your local disk, locate the project interchange file **MockNodeJs_pi.zip**. It should be in the `C:\student10\FileProcessing` directory.

__111. Within the ACE Toolkit workspace, import that project interchange file. Navigate to **File > Import > IBM Integration > Project Interchange** and click **Next**. On the Import Project Interchange Contents panel, set `From zip file:` to the full path of the project interchange file and make sure that the MockNodeJs project is selected. Click **Finish**.

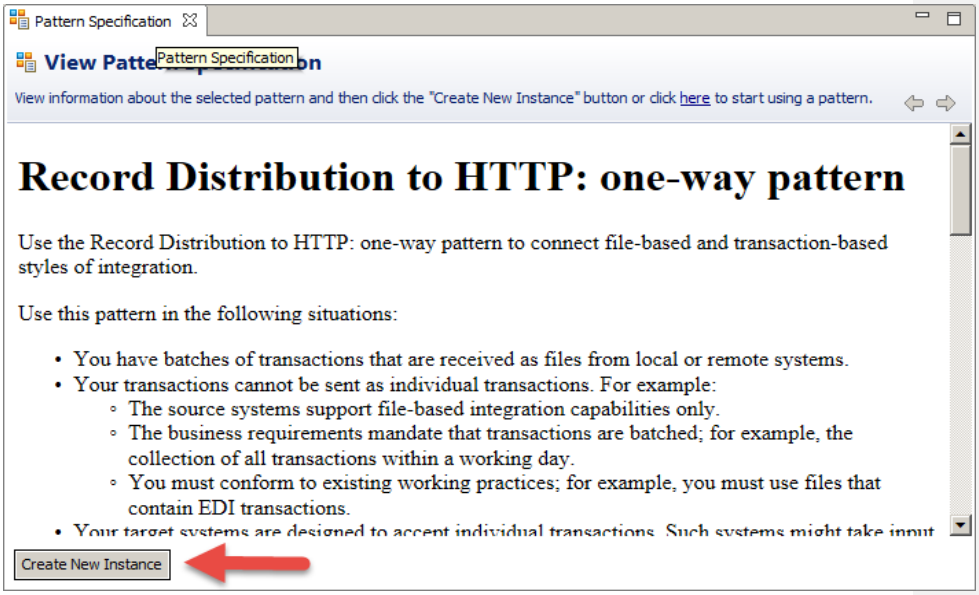


__112. In the Application Development view in the upper left of the toolkit, select the **MockNodeJs** application, right-click, select **Deploy**, select the **myaceworkdir** integration server on the Deploy panel, and click **Finish**. You should see the MockNodeJs application deployed to the **myaceworkdir** integration server.

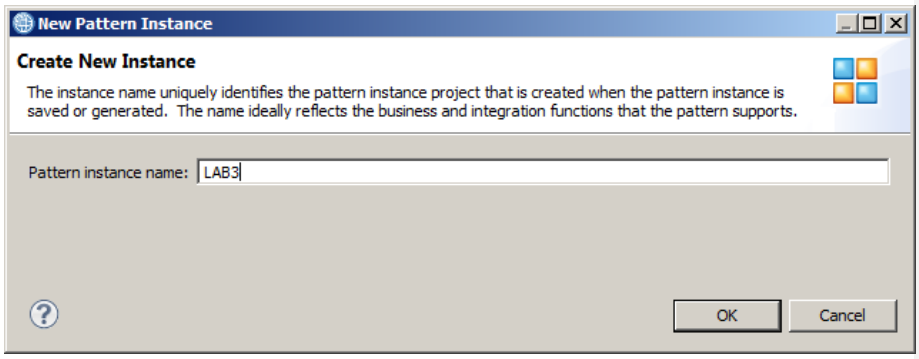


3.4 Configuring and generating the pattern instance

__113. In the Patterns Explorer, select **Patterns > File Processing > Record Distribution > HTTP one-way** and single-click. On the resulting Pattern Specification page, click **Create New Instance**. (Note: The correct path is under the Patterns folder, not the sibling folder named Pattern Repositories.)

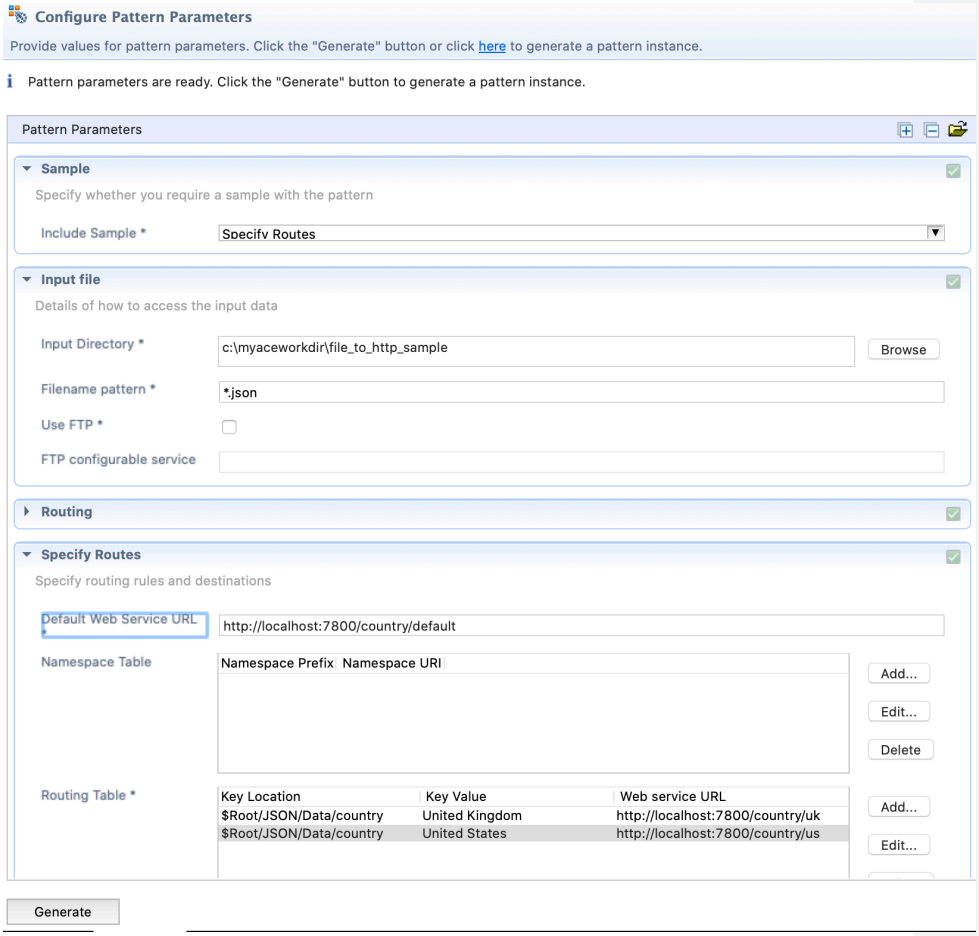


__114. In the New Pattern Instance window, specify a name.



__115. On the Pattern Configuration editor, specify exactly the values shown in the screen capture below. Modify the port number in each of the three URLs below with the value **7800 (which is the default http listener port of your application**.

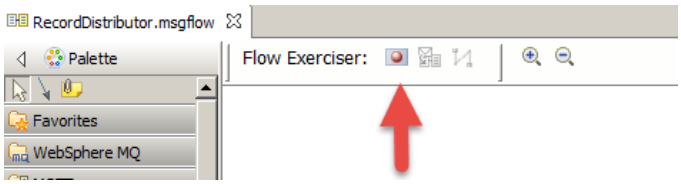
Then click **Generate** in the lower left of the panel and wait until pattern generation is complete.



3.5 Running the generated ACE application

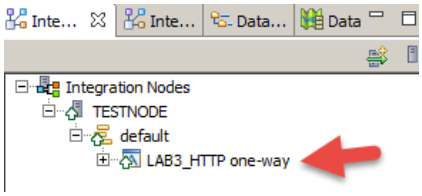
__116. In the Application Development view, expand the **<pattern instance name>_HTTP one-way** application, select the **RecordDistributor** message flow, and double-click to open it in a message flow editor. Review the flow, its node, their settings, and the Pattern Specification to ensure that you understand what this flow does: reads JSON files from the `file_to_http_sample` directory, parses them as JSON, routes them based upon the country specified in the JSON, and submits an HTTP request to the URL appropriate for that country.

__117. With the RecordDistributor message flow still open in a message flow editor, click the red **Flow Exerciser** button. If the toolkit presents a `Ready to record message panel`, select the box for `Do not show this message again` then click **Close**.



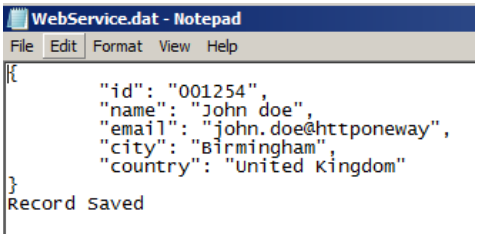
The application should be deployed, and flow recording should start.

__118. In the Integration servers view in the Toolkit's lower left, confirm that the **<pattern instance name>_HTTP one-way** application deployed.



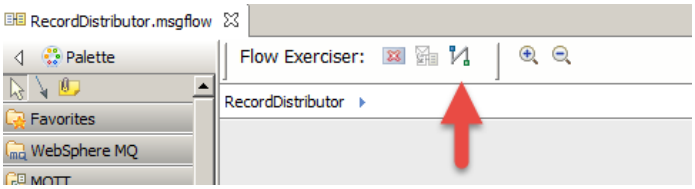
__119. Run the flow. In the Application Development view, navigate to **Independent Resources > <pattern instance name>_HTTP-oneway_sample** and select the `Record1.json` file. Right-click then select **Copy** to copy the file. Paste the file into the `file_to_http_sample` directory polled by the flow's File Input node. The RecordDistributor flow will read the file, route it based upon the `country` value in the JSON, and send the request to the corresponding URL.

__120. Confirm that the web service has received the request sent by the ACE flow. Open the `WebService.dat` file in the `out` subdirectory to see output from that web service.

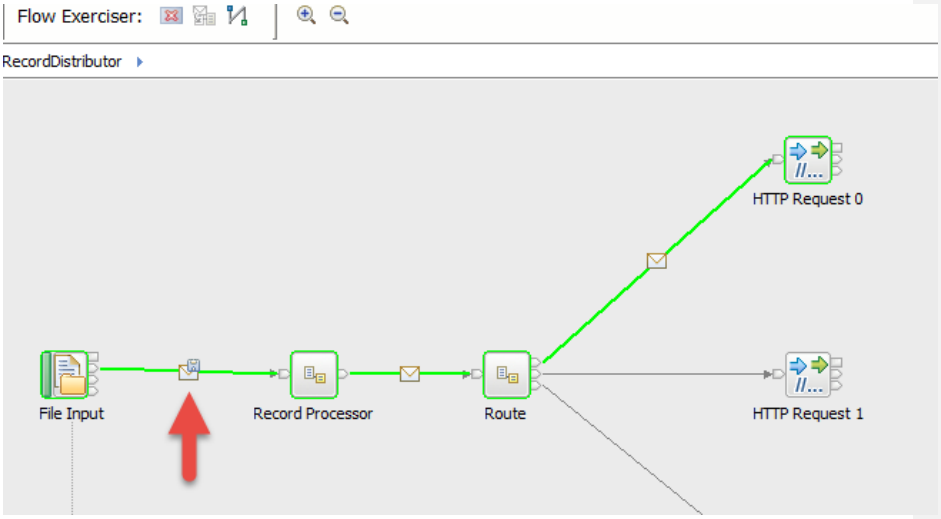


Either of these outputs indicates successful execution of the message flow.

__121. Examine the path that the file data took through the message flow. At the top of the message flow editor for the RecordDistributor flow, click the **View Path** icon.



Green highlighting will appear to indicate the path the previous file data took through the flow. Click the **envelope icon** on the wire between the File Input and Record Processor nodes.



View the message data in the Recorded Message dialog. In particular, notice the value of the `country` field.

Click the **envelope icon** on the green wire between the Route node and the HTTP Request 0 node. View the data in the new Recorded Message dialog. The Route subflow routed the message through this path of the flow based upon the value of the `country` field.

__122. Rerun the RecordDistributor flow with different files. Repeat the prior three steps using the files `Record2.json` and `Record3.json`. Because each of these files has a different country value, they will take different paths through the flow.

3.6 Summary

This lab has provided an illustration of how to start from, configure, and instantiate a pattern. It has also illustrated file processing, content-based routing, and file-to-HTTP protocol switching.

END OF LAB 3