13 Content-Based Routing

13.1 Description

This tutorial demonstrates using the content of messages to make routing decisions about where to send service requests.

13.2 Prerequisites

13.2.1 Environment

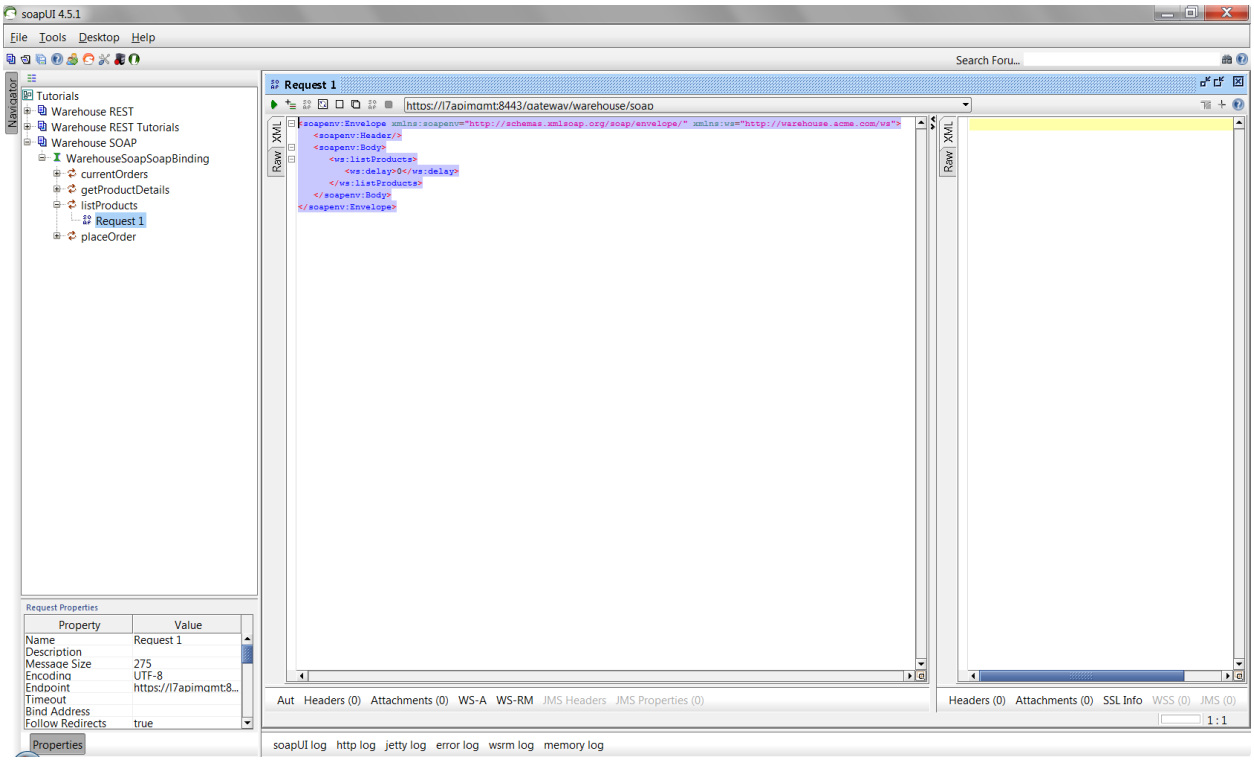
1. Layer 7 SecureSpan Gateway *(this tutorial was designed using a version 7.0 gateway; it may or may not work with earlier versions; it should work with later versions)*
2. Layer 7 Policy Manager *(this tutorial uses the Policy Manager software installation; the software installation version must match the gateway version; alternatively, users can use the Policy Manager browser-based version which always matches the gateway version that is connected to)*

13.2.2 Tutorials

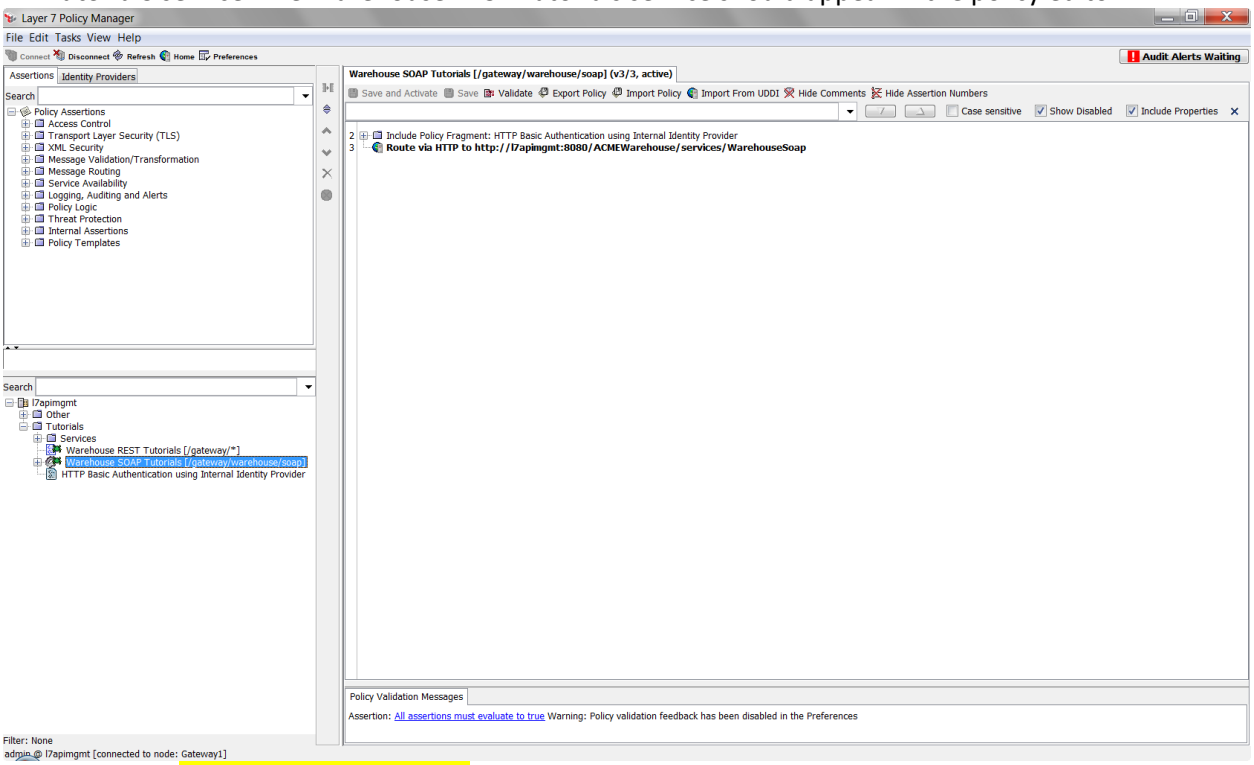
1. 1 General Information
2. 2 Deploy Tutorial Services
3. 3 Test Tutorial REST Service
4. 4 Test Tutorial REST Service
5. 5 Publish SOAP Service
6. 6 Publish REST Service
7. 8 Basic Authentication
8. 11 Policy Fragments

13.3 Tutorial Steps

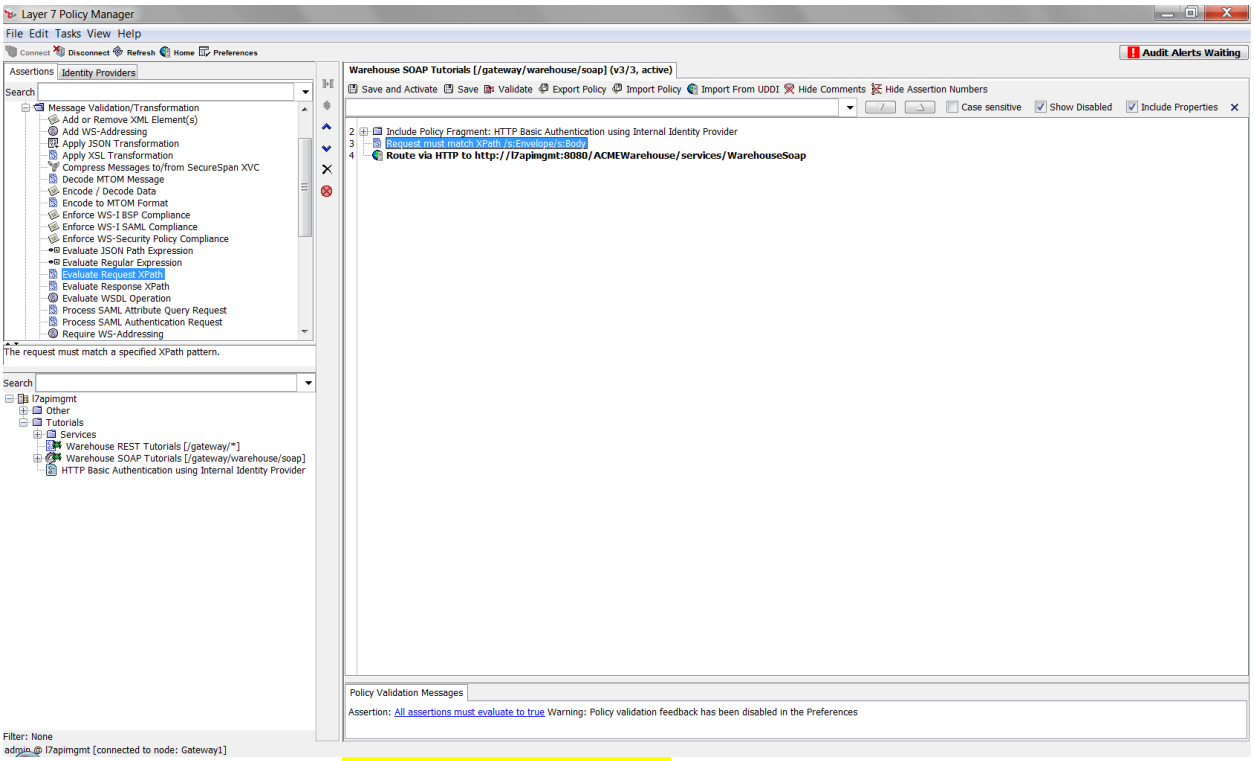
1. Connect to your gateway using Policy Manager (see tutorial ***1 General Information***).
2. Per ***1 General Information/Basic Policy Concepts/Policy Authoring/Policy Revisions***, set the active policy version of the ***Warehouse SOAP Tutorials*** service to the version that has been commented with, **Tutorial 11 Complete**.
3. Open soapUI and use the ***Warehouse SOAP*** project created in the ***3 Test Tutorial SOAP Service*** tutorial and updated in the 11 Policy Fragments tutorial.
4. In the soapUI Navigator window, expand the ***Warehouse SOAP/WarehouseSoapSoapBinding/listProducts*** operation, and double-click on the ***Request 1*** test message. Highlight and copy the entire request message.



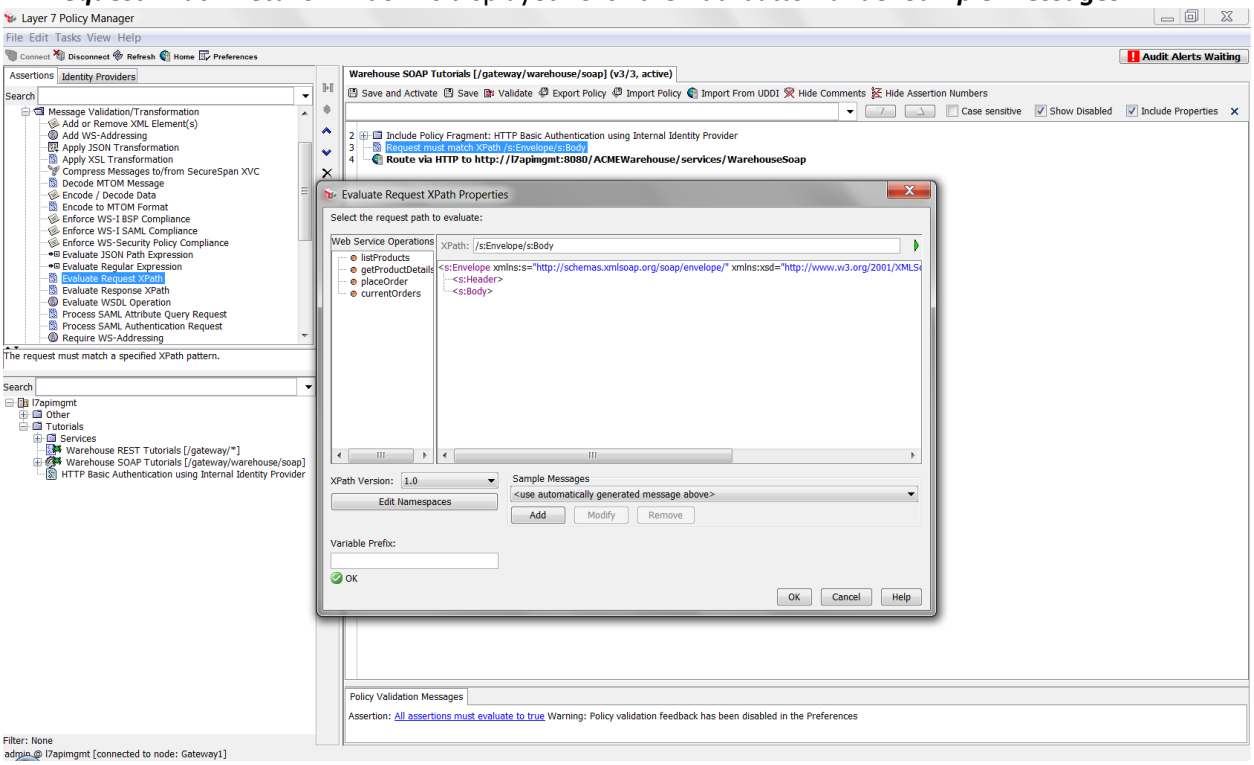
5. Using the Policy Manager, on the services and policy tree, double-click on the ***Warehouse SOAP Tutorials*** service. The Warehouse REST Tutorials service should appear in the policy editor.



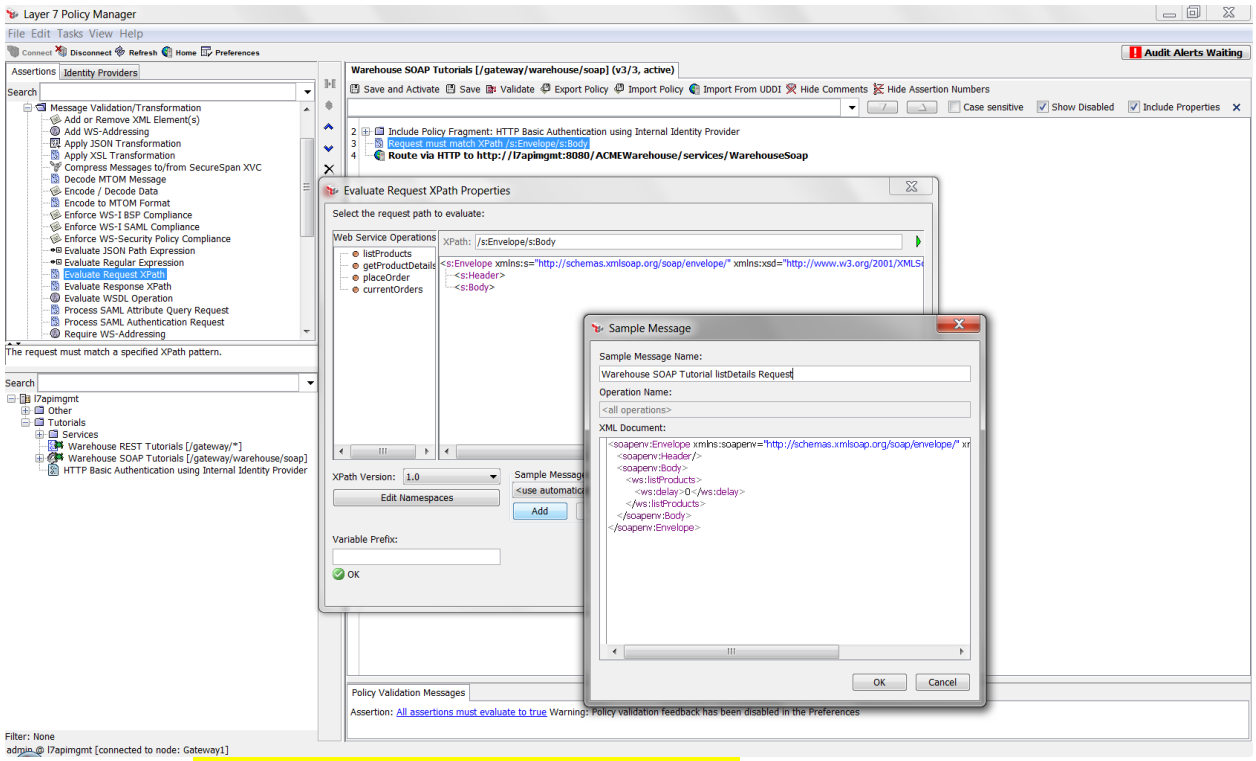
6. Locate the ***Evaluate Request XPath*** assertion in the ***Message Validation / Transformation*** folder of the assertion palette. Drag and drop the assertion to the policy editor, and place it between the ***Include Policy Fragment*** and ***Route*** assertions.



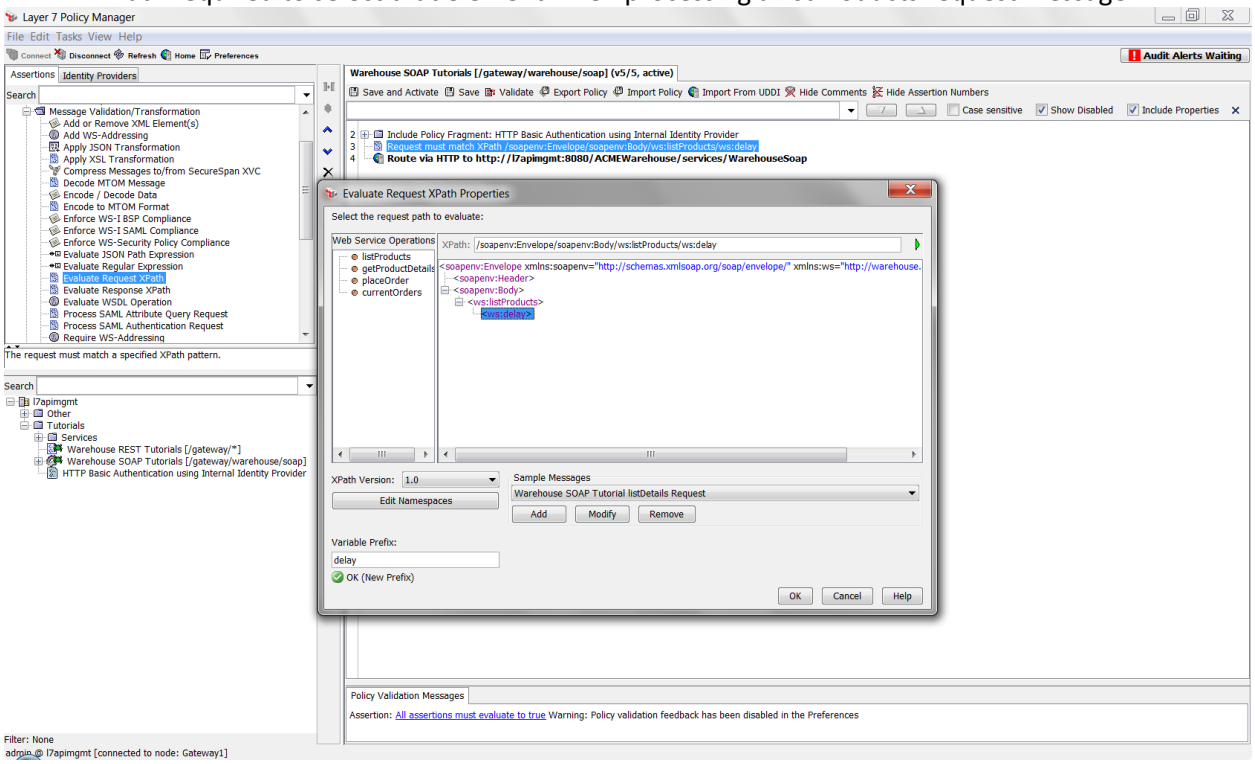
7. Double-click on the new ***Request must match XPath*** assertion in the policy editor. The ***Evaluate Request XPath Details*** window is displayed. Click the ***Add*** button under ***Sample Messages***.



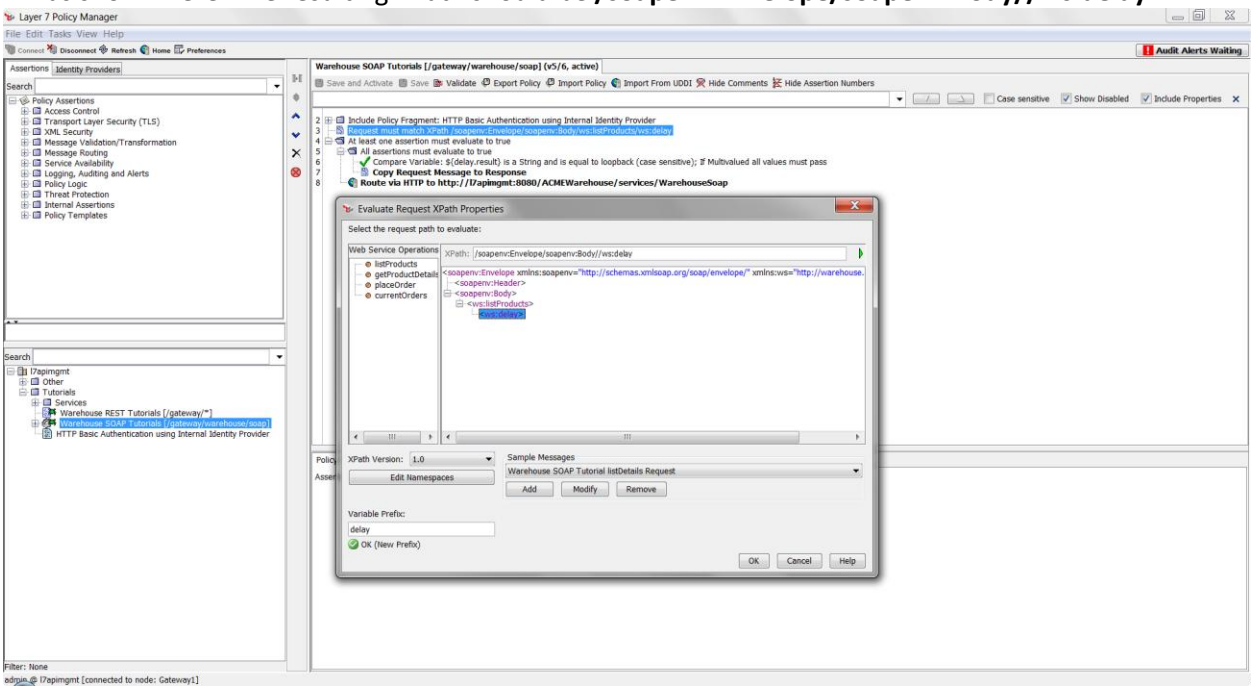
8. The ***Sample Message*** window is displayed. Overwrite the default ***XML Document*** with the document copied in step 4 above and enter a ***Sample Message Name*** such as “Warehouse SOAP Tutorial listDetails Request”. Click the ***OK*** button to add the message and close the window.



9. The sample ***Warehouse SOAP Tutorial listDetails Request*** should now be displayed in the ***XML Document*** entry pain below the XPath statement. If it is not, select ***Warehouse SOAP Tutorial listDetails Request*** from the ***Sample Messages*** dropdown.
10. In the ***XML Document*** entry pain, click on the <ws:delay> element to automatically generate the XPath required to select that element when processing a listProducts request message.

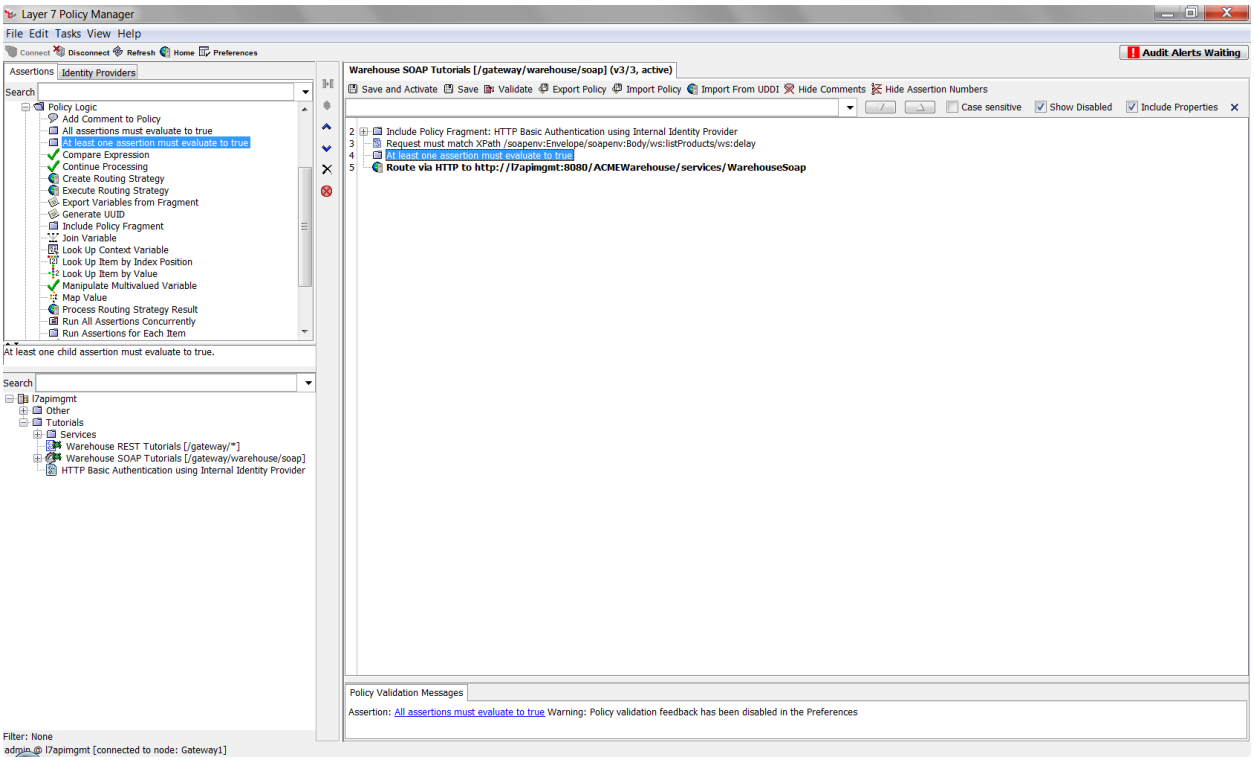


Edit the ***XPath*** produced by clicking on the <ws:delay> element to remove the reference to the ws:listProducts node and create and XPath that will find a ws:delay node in any SOAP document as shown here. The resulting XPath should be **/soapenv:Envelope/soapenv:Body//ws:delay**



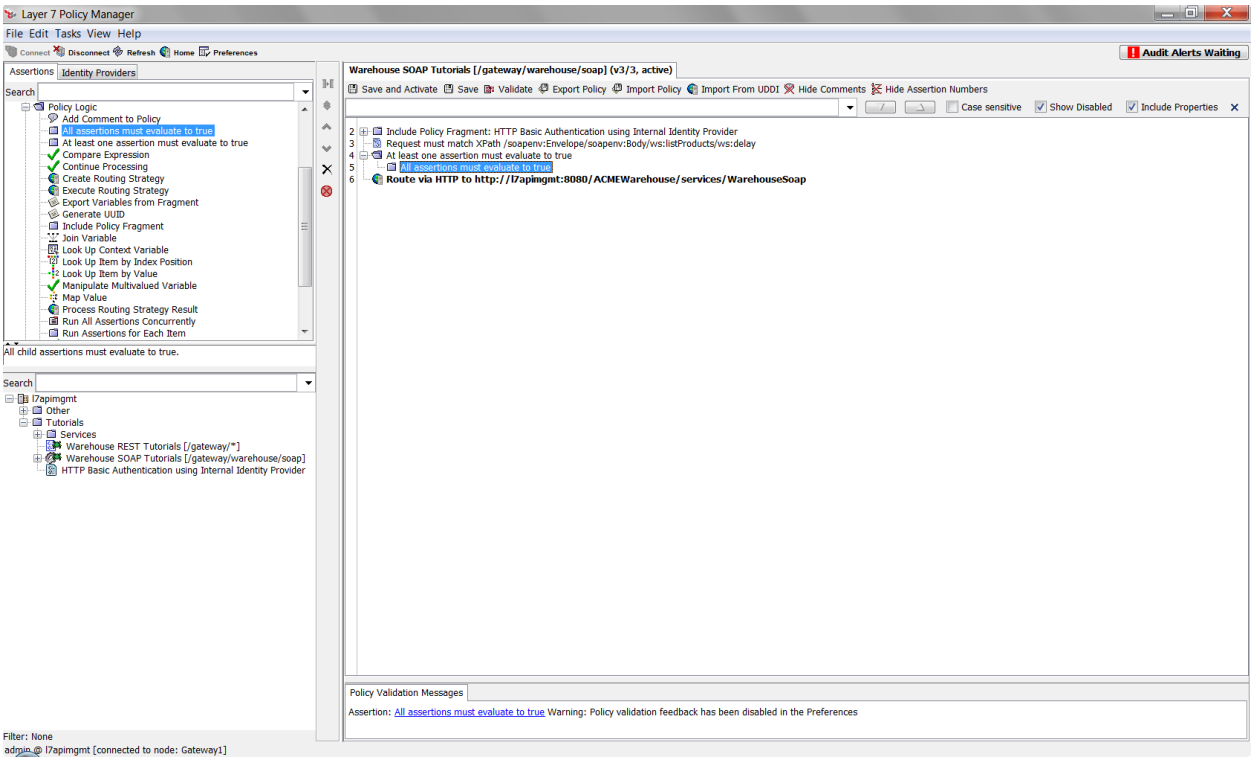
11. Enter “delay” in the ***Variable Prefix*** field. Click the ***OK*** button to save the changes to the Request must match XPath assertion and exit the window.

12. Locate the ***At least one assertion must evaluate to true*** assertion in the ***Policy Logic*** folder of the assertion palette. Drag and drop it onto the policy editor between the ***Request must match XPath*** assertion and the ***Route*** assertion.



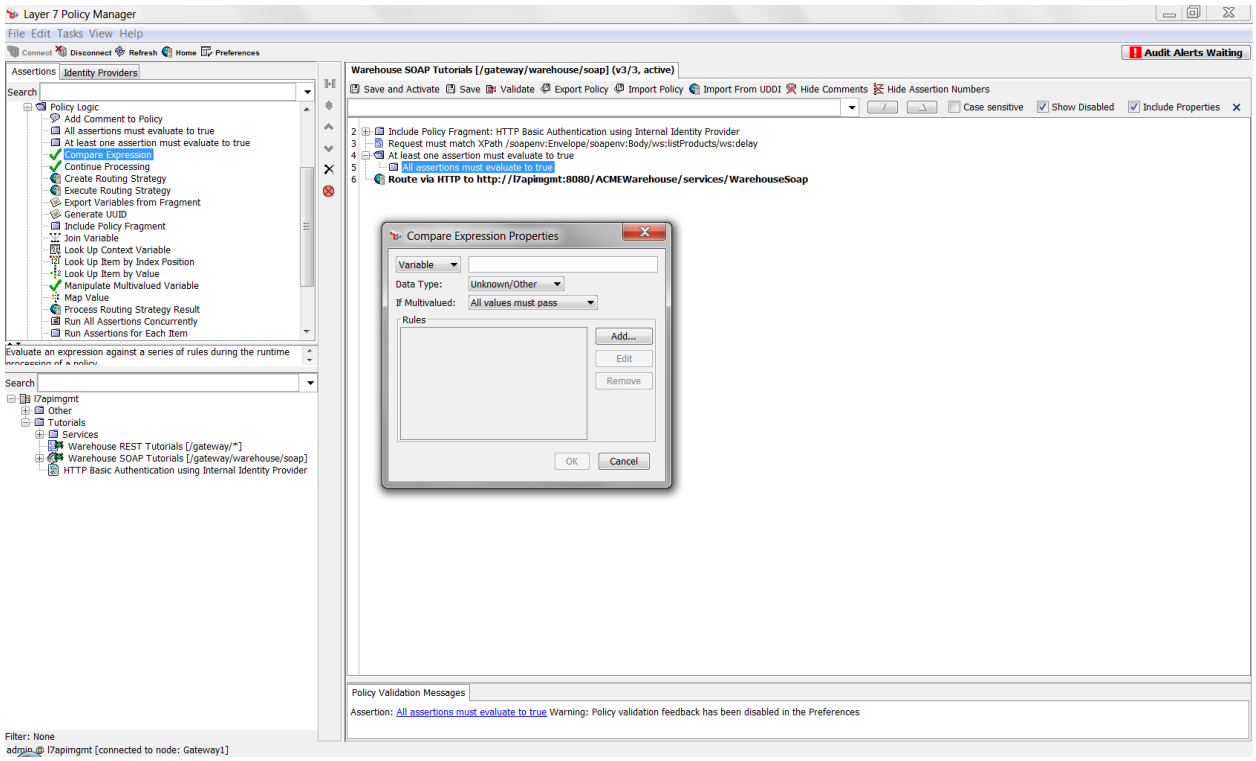
Note: the **At least one assertion must evaluate to true** *assertion is a folder into which other assertions are placed. It associates the assertions contained within it in a logical relationship. Contained assertions execute in order, from top to bottom. Each assertion will either succeed or fail, as per usual. However, unlike in the main processing flow outside of the folder, if an assertion fails inside of an* **At least one assertion must evaluate to true** *folder, processing continues at the next assertion inside the folder. Processing in the folder concludes only when one of the contained assertions succeeds, or all assertions have executed without a single success. In the case where no assertions succeed, the* **At least one assertion must evaluate to true** *assertion is considered to have failed. In the case where an assertion succeeds, processing inside the folder concludes on that first success, the* **At least one assertion must evaluate to true** *assertion is considered to have executed successfully, and control immediately passes to the next assertion in line outside of the folder without evaluation of any further assertions inside the further.*

13. Locate the ***All assertions must evaluate to true*** assertion in the same ***Policy Logic*** folder and drag and drop it onto the ***At least one assertion must evaluate to true*** assertion in the policy editor, taking care that the end result is a nested structure in the policy editor like the one shown here. If you make a mistake and the nested structure does not initially appear as shown, click and hold on the new ***All assertions must evaluate to true*** assertion, wherever it has landed on the policy editor, and move it onto the ***At least one assertion must evaluate to true*** assertion.

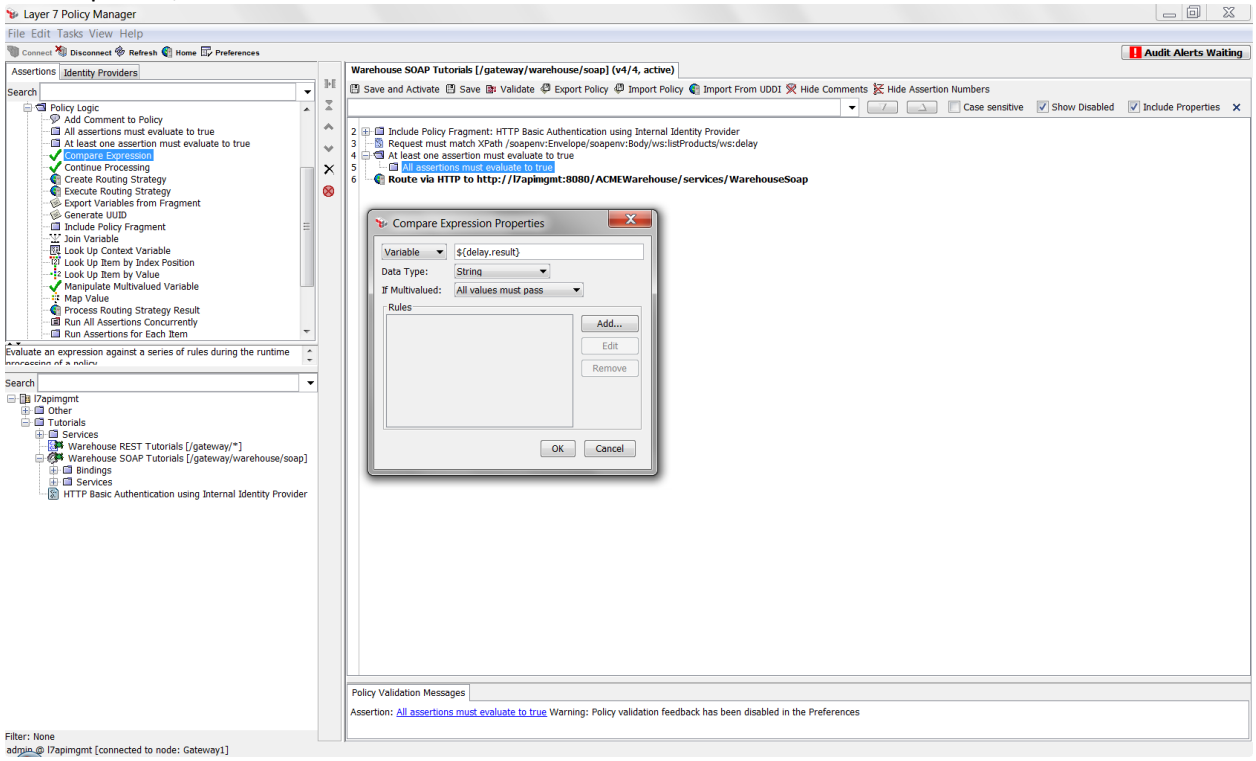


Note: The **All assertions must evaluate to true** *assertion is also a folder into which other assertions can be placed. Like the* **At least one assertion must evaluate to true** *assertion, it binds together the assertions that it contains into a logical relationship. Assertions execute in order, from top to bottom. Processing inside the* **All assertions must evaluate to true** *folder ends when either all contained assertions execute successfully, or one of them fails. If one fails, processing inside the folder immediately ceases, no further assertions are evaluated inside the folder, and the* **All assertions must evaluate to true** *assertion itself is considered to have failed. The* **All assertions must evaluate to true** *assertion only succeeds if all contained assertions succeed.*

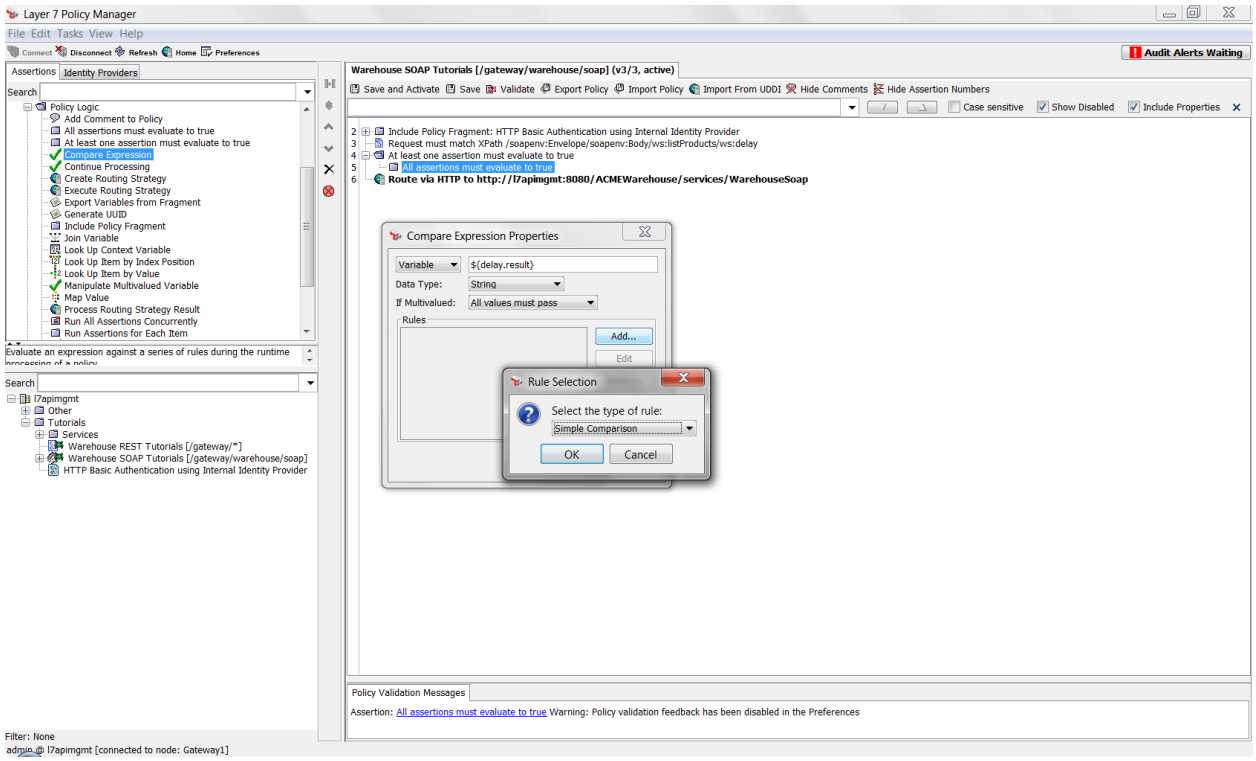
14. Locate the ***Compare Expression*** in same ***Policy Logic*** folder. Drag and drop it onto the ***All assertions must evaluate to true*** assertion. The ***Compare Expression Properties*** window automatically opens.



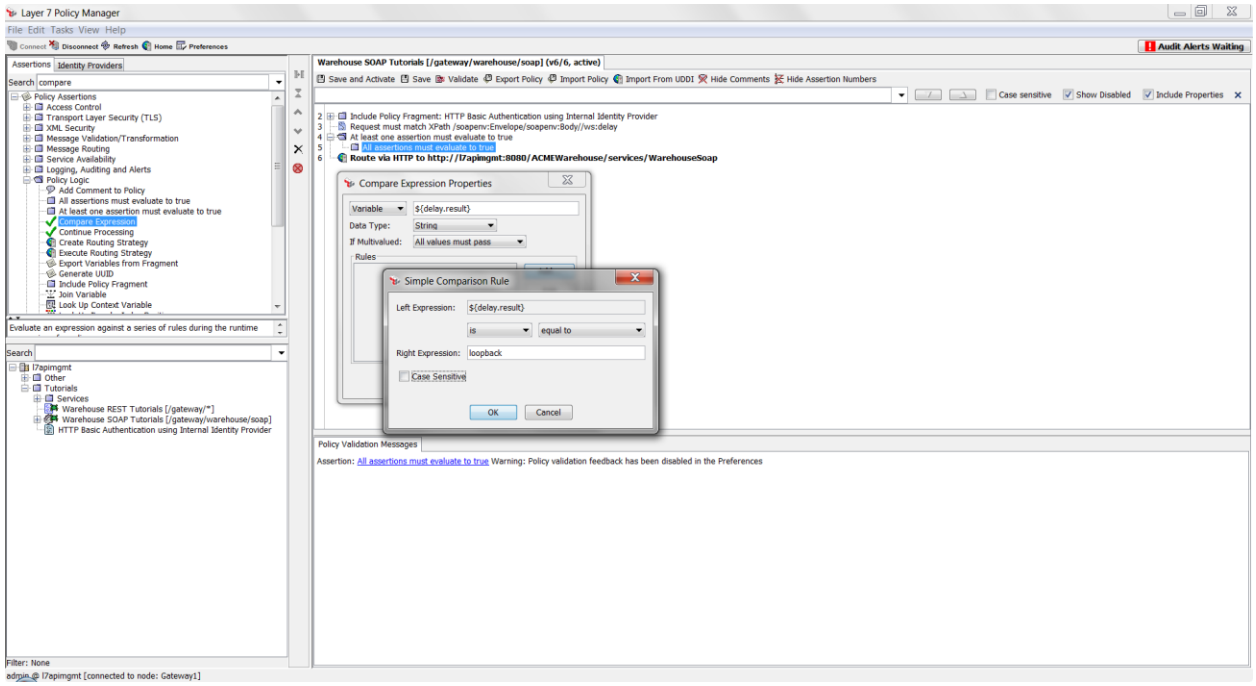
15. Enter "${delay.result}" in the ***Variable*** entry field. Select the ***String*** from the ***Data Type*** dropdown, and click the ***Add*** button.



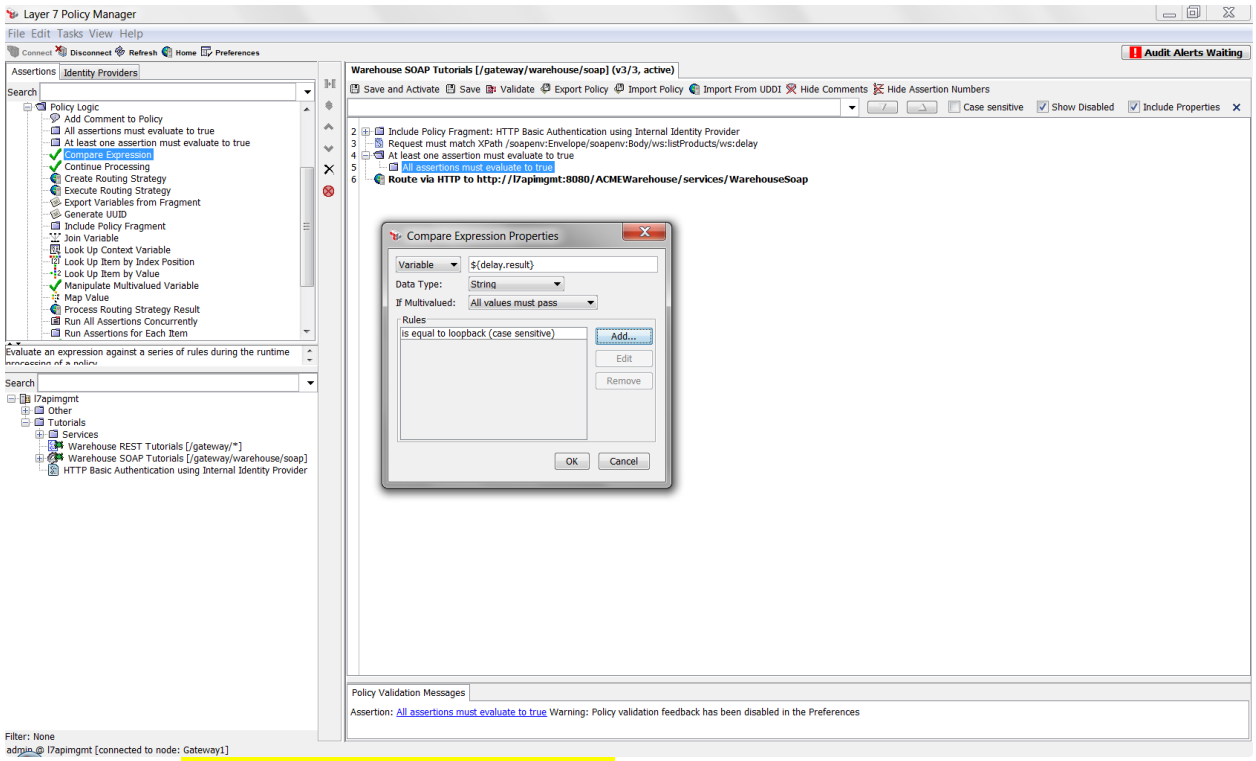
16. Click the ***OK*** button to select the default Simple Comparison rule.



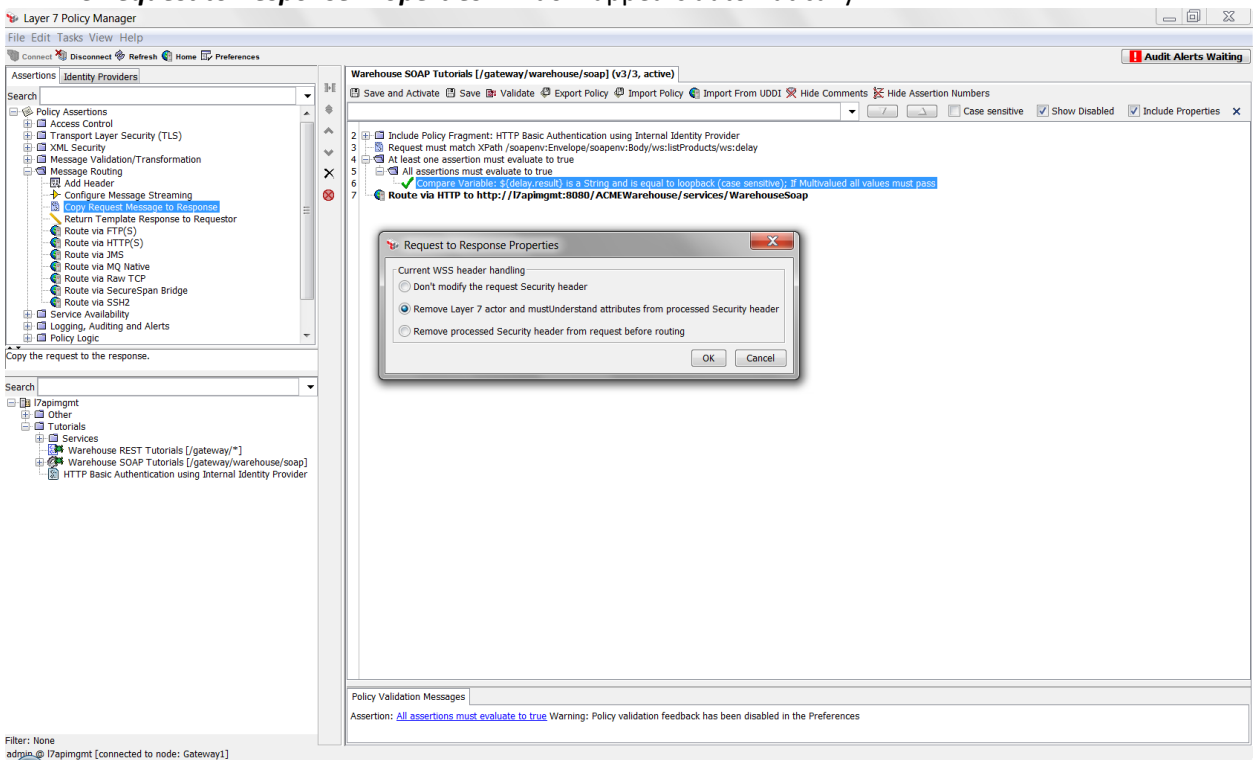
17. The ***Simple Comparison Rule*** window should open. Enter “loopback” into the ***Right Expression*** entry field, and un-check the ***Case Sensitive*** checkbox. Click the ***OK*** button to save changes and close the window.



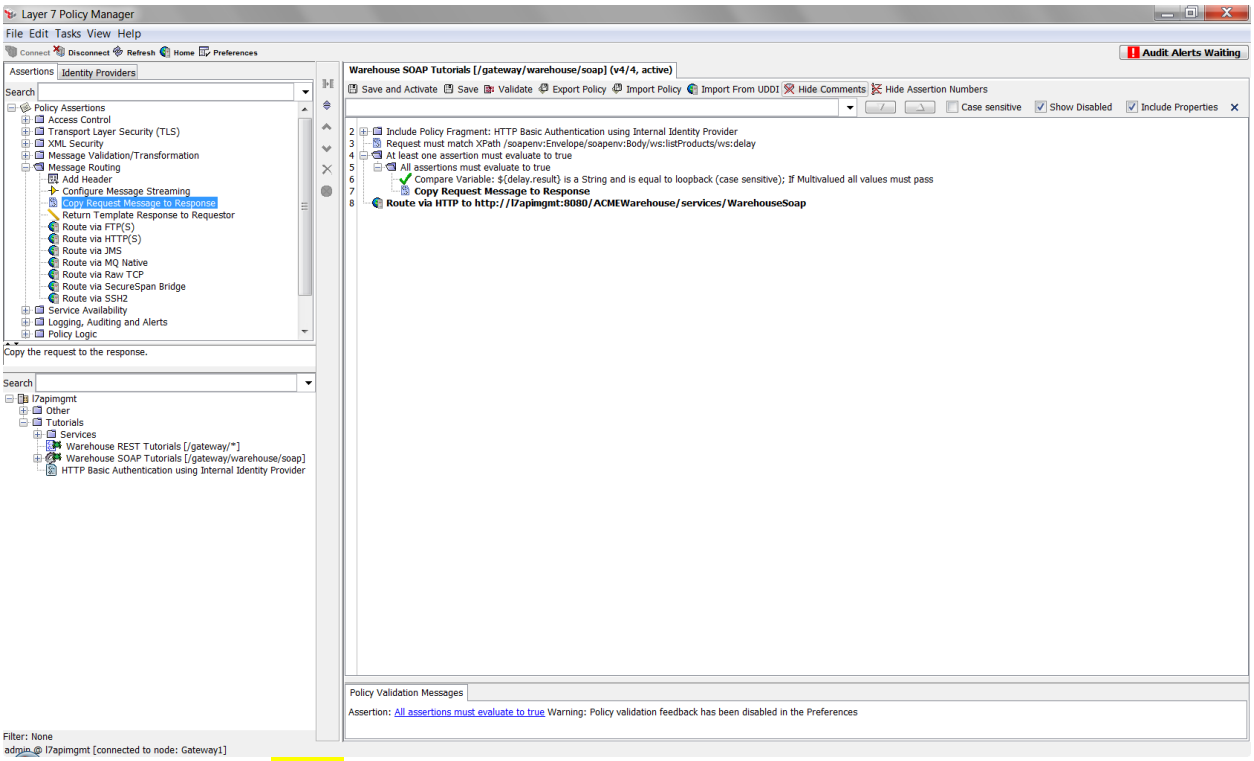
18. Click the ***OK*** button to save the completed expression and close the window.



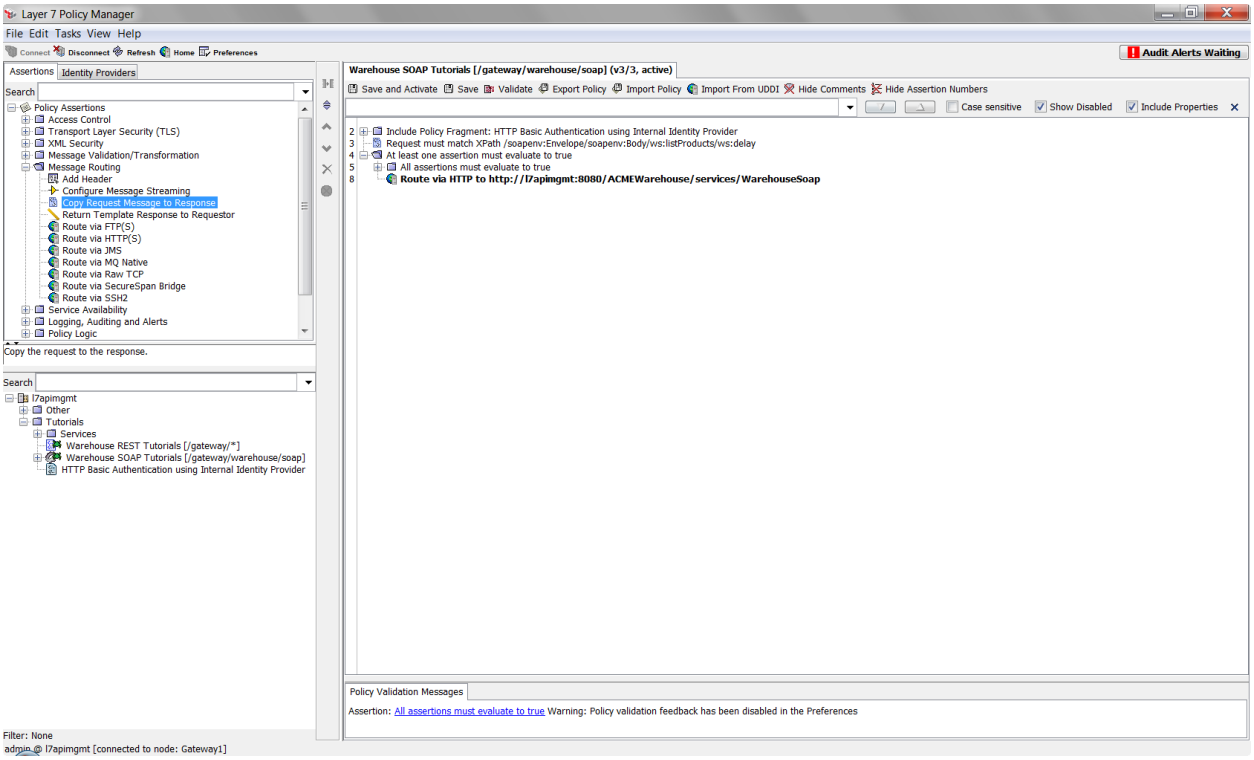
19. Locate the ***Copy Request Message to Response*** assertion in the ***Message Routing*** folder of the assertion palette. Drag and drop it under new ***Compare Variable*** assertion in the policy editor. The ***Request to Response Properties*** window appears automatically.



20. Click the ***OK*** button to accept the default option. The policy should now look like this one:

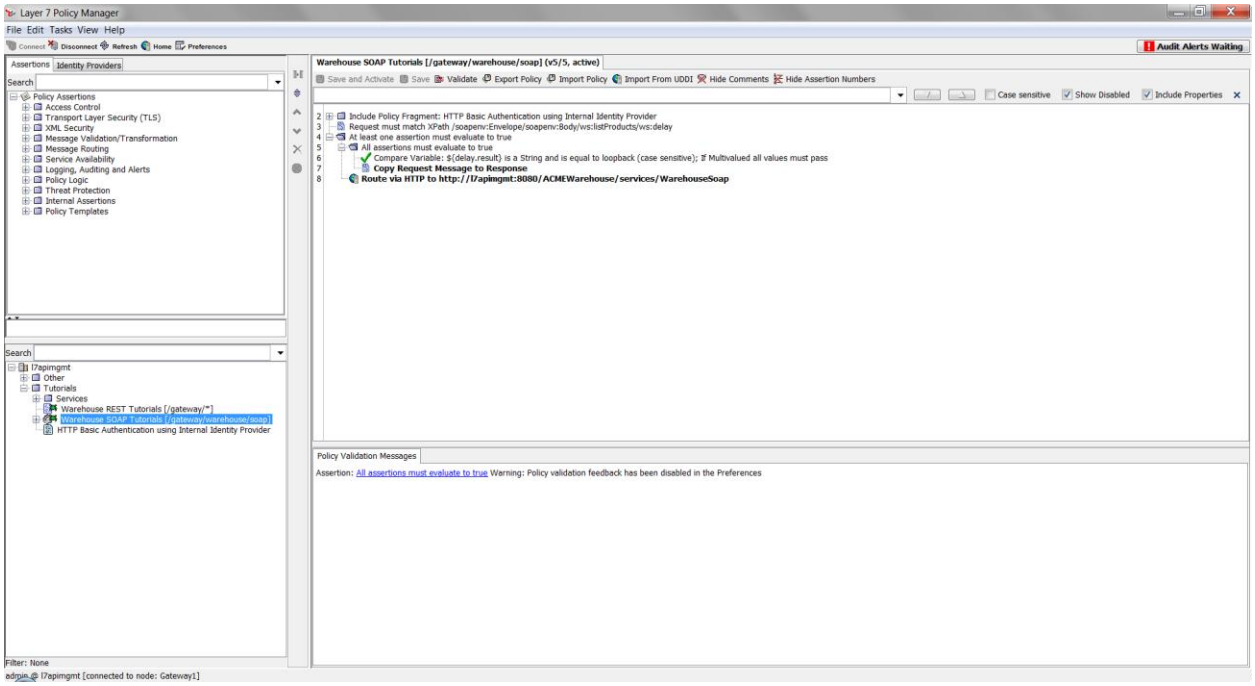


21. Drag and drop the ***Route*** assertion into the ***At least one assertion must evaluate to true*** assertion and use the ***arrow buttons*** to the left of the policy editor to position it as a sibling of the ***All assertions must evaluate to true*** assertion, after that assertion. The result is shown here with the All ***assertions must evaluate to true*** assertion folder minimized.



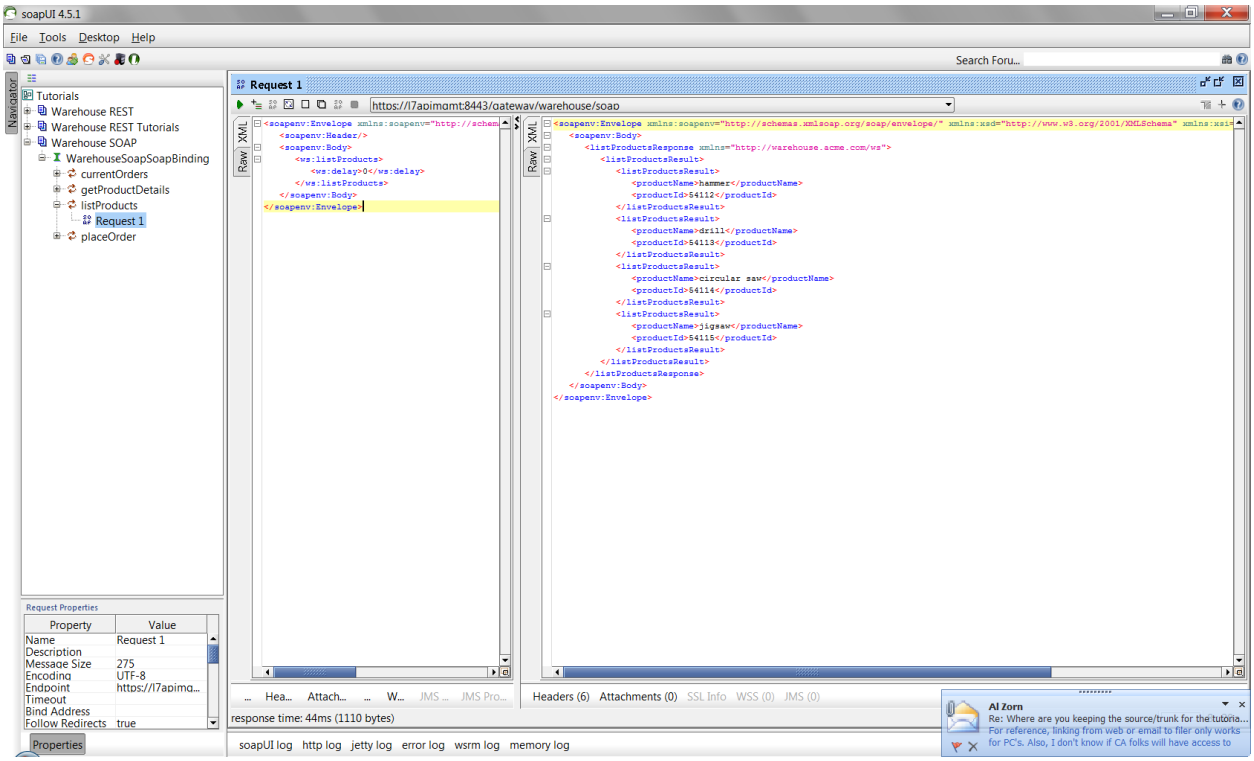
22. Click the ***Save and Activate*** button.

Let's pause for a moment to consider what we have constructed. Recall that the **At least one assertion must evaluate to true** *folder acts somewhat like an OR statement that attempts contained assertions consecutively until it finds one that works. And the fully-expanded policy should look like this:*

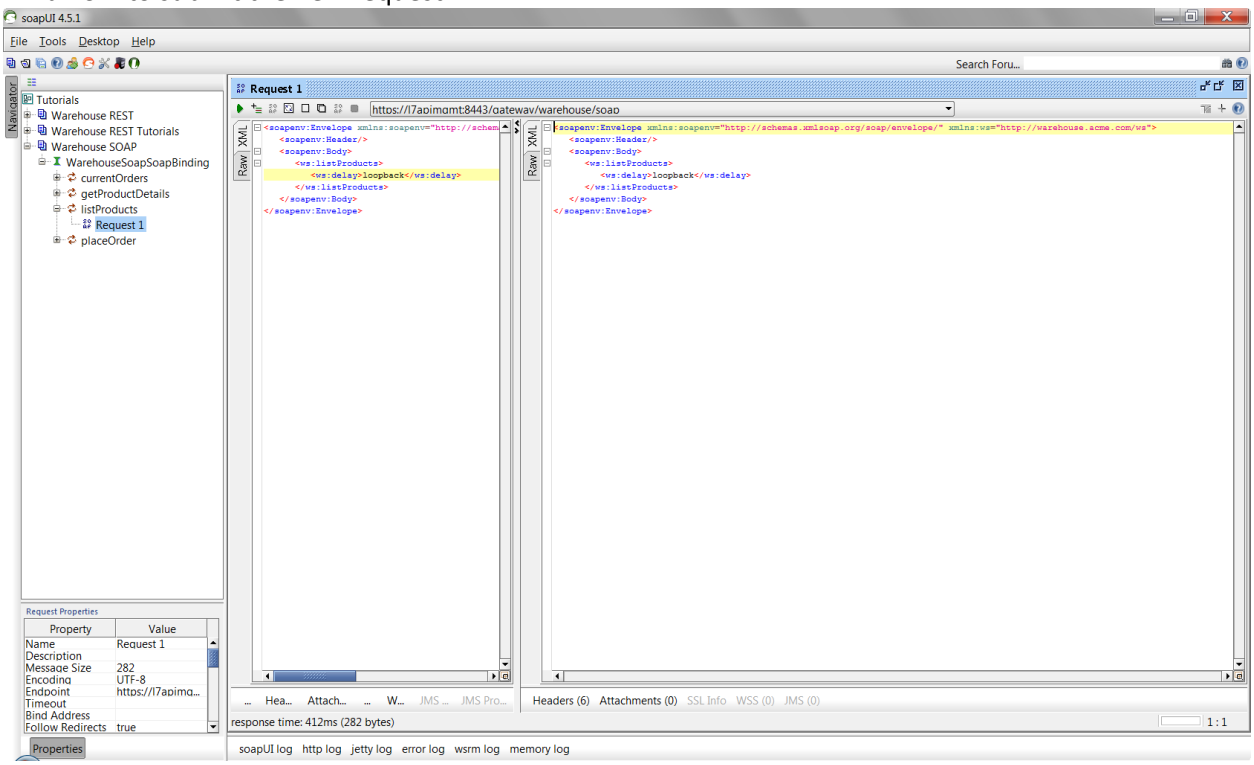


You have created a policy that will check for a particular value – the word "loopcack" – in the incoming request message. If it finds that value, it will copy the request message into the response context, and then skip routing to the service provider. If on the other hand that value is not found, the policy will route to the service provider as per usual.

23. Return to soapUI and the ***Warehouse SOAP*** project. In the soapUI Navigator window, if it is not already visible, expand the ***Warehouse SOAP/WarehouseSoapSoapBinding/listProducts*** operation, and double-click on ***Request 1*** to open the test message. Ensure that the request is targeted to the correct URL as shown, then click on the ***green arrow*** to submit the request to the gateway.



24. Now, overwrite the contents of the <ws:delay> element with “loopback” and click the ***green arrow*** to submit the new request.



25. Try testing other operations in the ***Warehouse SOAP*** project. You should be able to skip backend routing and loop the requests back to soapUI using the policy that you just created.

26. Per ***1 General Information/Basic Policy Concepts/Policy Authoring/Policy Revisions***, and as demonstrated at the end of tutorial ***2 Deploy Tutorial Services***, comment the active policy revision of the ***Warehouse SOAP Tutorials*** service with the comment, **Tutorial ????? Complete**.

13.4 Additional Context

This tutorial demonstrates the basic use of logical assertions to make a decision about whether to route messages to the service provider, or to loop the original requests back to the client. Although this function may be useful in test environments, a much more practical application of content-based routing is to use the content of messages to select different backend service providers for routing. A popular example is to provide enhanced service levels for higher value transactions or clients, perhaps routing transactions over a certain value to the "gold" service provider, while routing all others to a "silver" provider that might not return responses or fulfill orders as quickly as the gold provider. This would certainly have been possible for us to do in our example, but for the lack of multiple service endpoints to route to on the back end.

This tutorial used the ***Warehouse SOAP Tutorial*** instead of the ***Warehouse REST Tutorial*** because the SOAP standard prescribes a consistent form for delivering request content: a SOAP Envelope with a body containing business information inside of an HTTP POST. Many REST requests, by contrast, use a variety of ways to transmit information about the request. GET requests use query string parameters, for instance, while PUT messages include message content, but may also include query string parameters. So for the purposes of keeping the tutorial as simple as possible, then, it was constructed using the Warehouse SOAP Tutorial.

That is not to say, however, that content-based routing is exclusive XML content, or even to the transport payload. JSON requests can be interrogated using the ***Evaluate JSON Path Expression*** assertion in much the same manner as the SOAP messages are probed using XPath. Other message formats can be scanned using the ***Evaluate Regular Expression*** assertion and in other ways as well.

Headers are available in context variables like ${request.http.header.<header-name>} or ${request.http.allheadervalues}, and query string parameters are available in context variables like ${request.http.parameter.<parameter-name>} and ${request.http.parameters[*n*]}, all of which can contain valuable information on which to make routing decisions.