11 Using Context Variables in XSL Transformations

11.1 Description

This tutorial demonstrates the use of the XSLT assertion with the use of parameters. The XSLT tutorial will give you the basics of using XSLT to search for elements within a document and replace the values from inbound data from the request. The inbound request is an HTTP GET with URL parameters that are used to alter a set XML document. The tutorial utilizes the Loopback Service, HTTP Parameter passing, Context Variables, and XSLT.

11.2 Prerequisites

11.2.1 Environment

1. Layer 7 SecureSpan Gateway *(this tutorial was designed using a version 7.1 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)*
3. Web Browser *(any browser should do for this tutorial)*

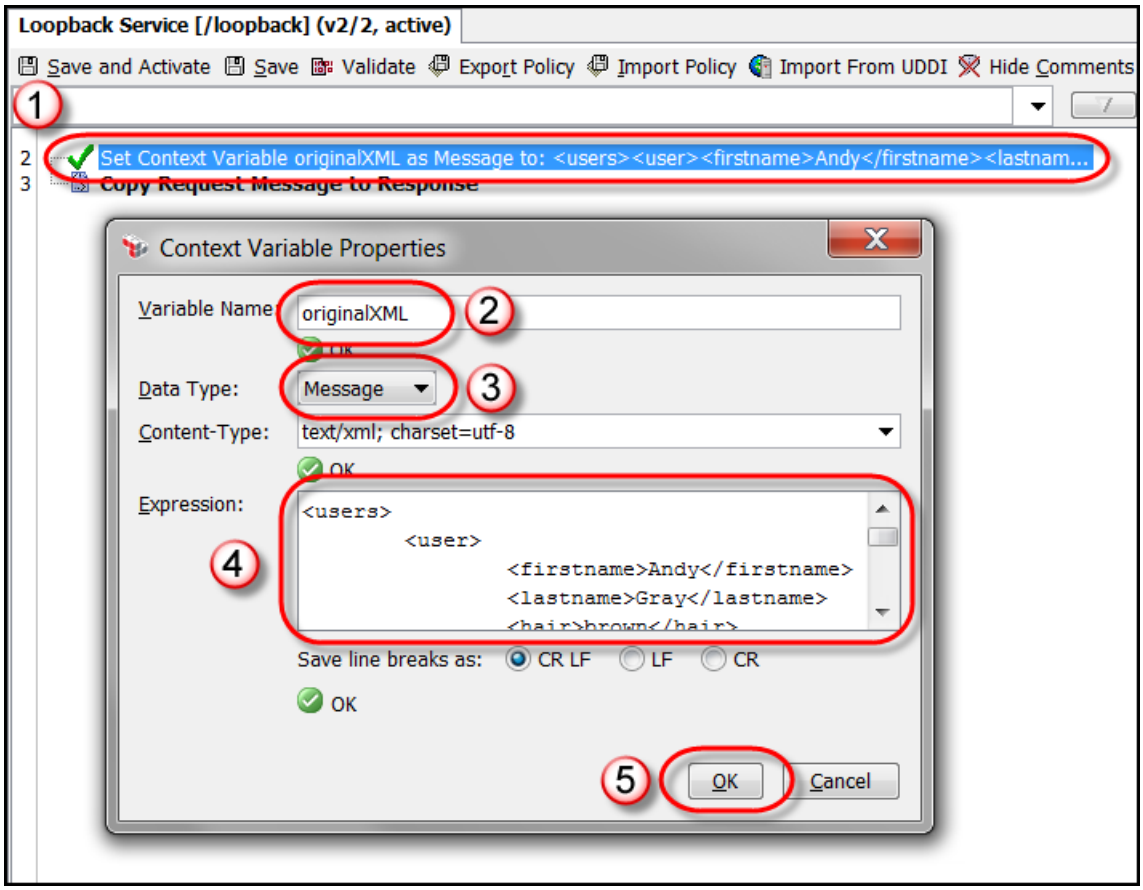
11.2.2 Tutorials

1. Layer 7 Tutorials - Getting Started
2. Tutorial 19 - Publish Loopback Service

11.3 Tutorial Steps

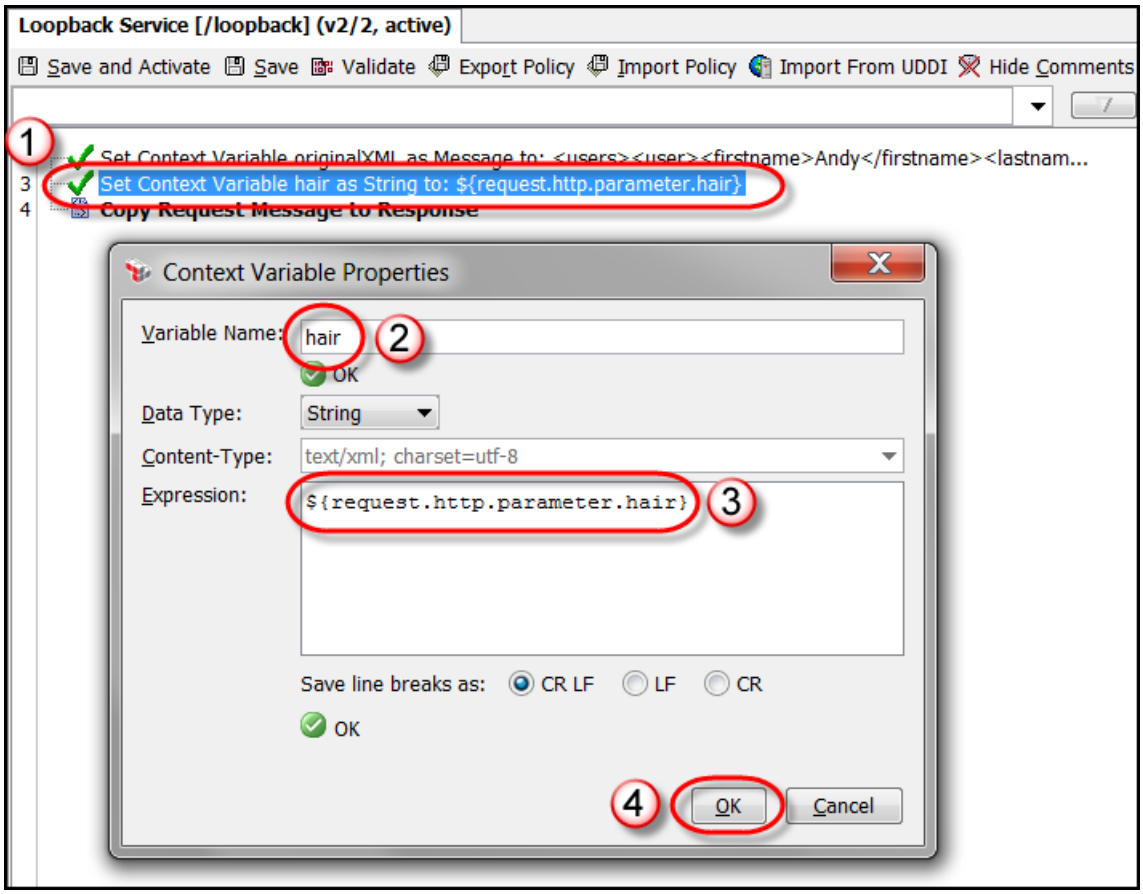
1. Connect to your gateway using Policy Manager (see ***Layer 7 Tutorials - Getting Started***).
2. Per ***Layer 7 Tutorials - Getting Started/Basic Policy Concepts/Policy Authoring/Policy Revisions***, set the active policy version of the ***Loopback Service*** to the version that has been commented with, **Tutorial 19 Complete**.
3. From the policy assertion tree, drag and drop the ***Policy Assertions/Policy Logic/Set Context Variable*** assertion to place it at the top of the ***Loopback Service*** policy in the policy editor, and in the Context Variable Properties dialog, in the Variable Name field, enter ***originalXML***. In the Data Type field, select ***Message***. In the Expression field, copy/paste the following content, and then click the ***OK*** button:

```
<users>
	<user>
		<firstname>Andy</firstname>
		<lastname>Gray</lastname>
		<hair>brown</hair>
		<eye>green</eye>
	</user>
	<user>
		<firstname>Rob</firstname>
		<lastname>Conti</lastname>
		<hair>brown</hair>
		<eye>brown</eye>
	</user>
</users>
```



4. From the policy assertion tree, drag and drop the ***Policy Assertions/Policy Logic/Set Context Variable*** assertion so that it's assertion #3 in the ***Loopback Service*** policy in the policy editor, and in the Context Variable Properties dialog, in the Variable Name field, enter ***hair***. In the Expression field, copy/paste the following content, and then click the ***OK*** button:

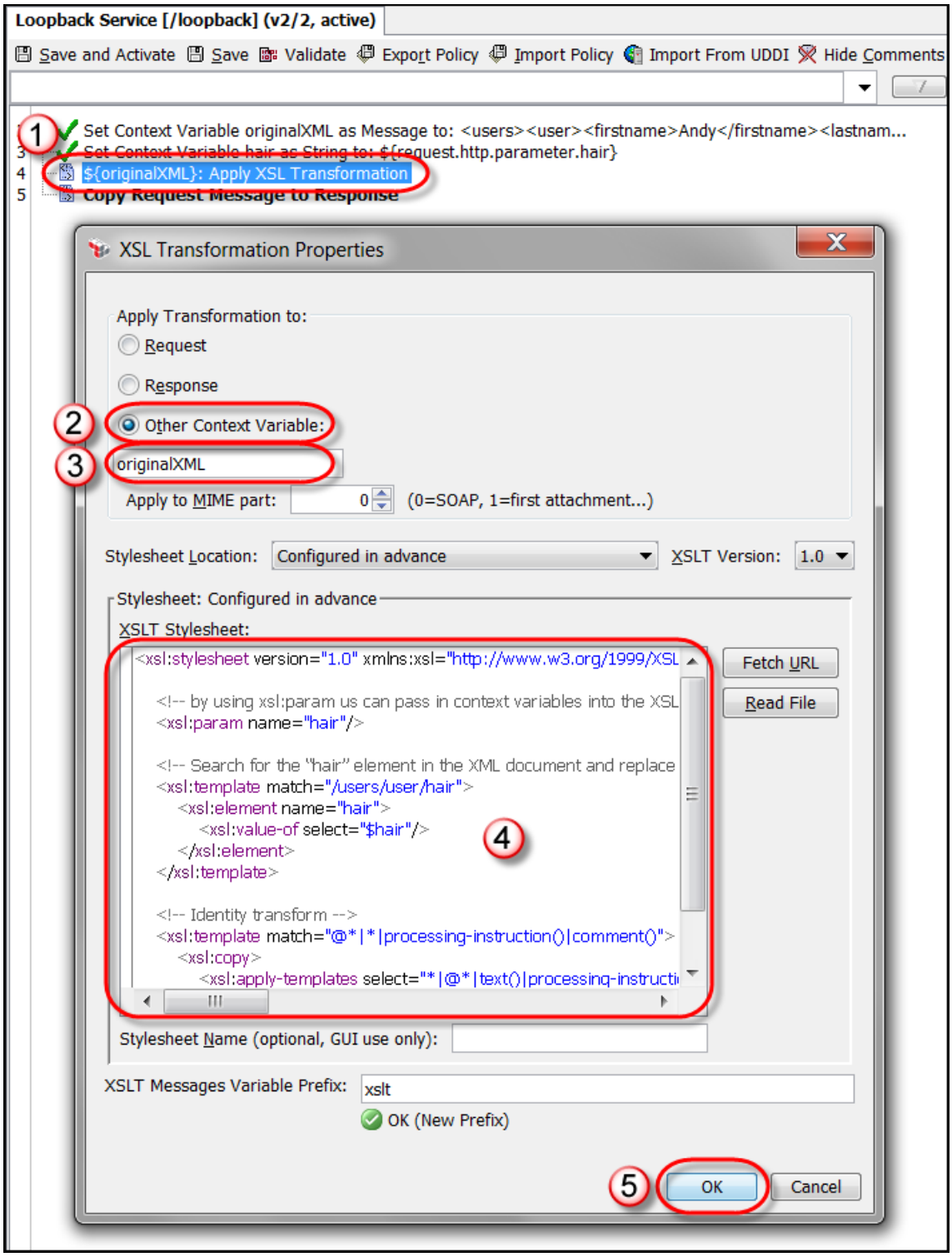
   ${request.http.parameter.hair}



5. From the policy assertion tree, drag and drop the ***Policy Assertions/Message Transformation Validation/Apply XSL Transformation*** assertion so that it's assertion #4 in the ***Loopback Service*** policy in the policy editor, and in the XSL Transformation Properties dialog, in the Apply Transformation to frame, select the ***Other Context Variable*** option and enter ***originalXML***. In the XSLT Stylesheet field, copy/paste the following content, and then click the ***OK*** button:

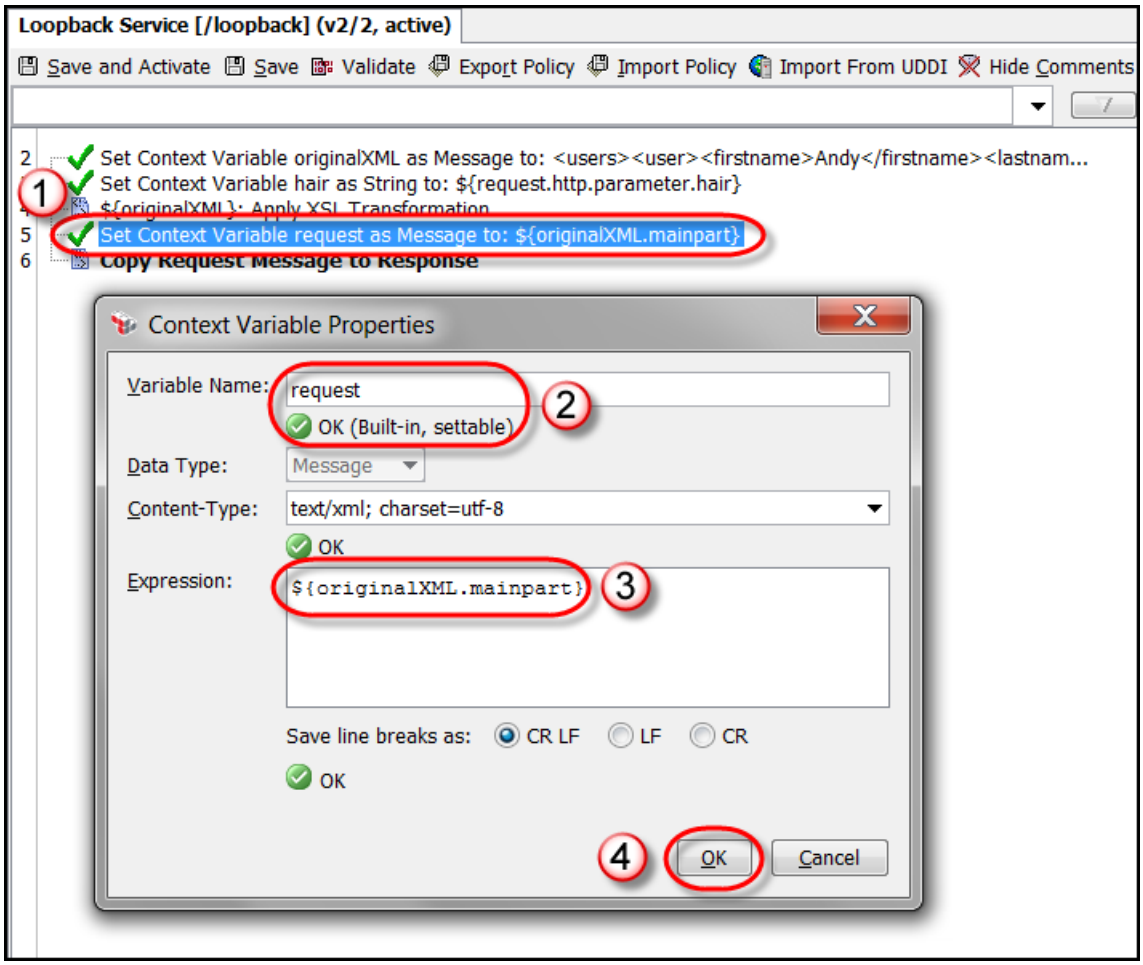
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- by using xsl:param us can pass in context variables into the XSLT transformation. If there exists a context variable prior to the execution of the Apply XSL Transformation assertion that matches the value of the “name” inside the xsl:param the value of that context variable I passed into the XSL transformation -->
  <xsl:param name="hair"/>

  <!-- Search for the “hair” element in the XML document and replace its value with the new value passed in via the browser -->
  <xsl:template match="/users/user/hair">
    <xsl:element name="hair">
      <xsl:value-of select="$hair"/>
    </xsl:element>
  </xsl:template>

  <!-- Identity transform -->
  <xsl:template match="@*|*|processing-instruction()|comment()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()|processing-instruction()|comment()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Note: DO NOT COPY THIS NOTE. See screenshot on next page for additional step detail.


Loopback Service [/loopback] (v2/2, active)
Save and Activate
Save
Validate
Export Policy
Import Policy
Import From UDDI
Hide Comments
1
Set Context Variable originalXML as Message to: <users><user><firstname>Andy</firstname><lastnam...
Set Context Variable hair as String to: ${request.http.parameter.hair}
${originalXML}: Apply XSL Transformation
Copy Request Message to Response
XSL Transformation Properties
Apply Transformation to:
Request
Response
2
Other Context Variable:
3
originalXML
Apply to MIME part:
0
(0=SOAP, 1=first attachment...)
Stylesheet Location:
Configured in advance
XSLT Version:
1.0
Stylesheet: Configured in advance
XSLT Stylesheet:
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL
<!-- by using xsl:param us can pass in context variables into the XSL
<xsl:param name="hair"/>
<!-- Search for the "hair" element in the XML document and replace
<xsl:template match="/users/user/hair">
<xsl:element name="hair">
<xsl:value-of select="$hair"/>
</xsl:element>
</xsl:template>
<!-- Identity transform -->
<xsl:template match="@*|*|processing-instruction()|comment()">
<xsl:copy>
<xsl:apply-templates select="*|@*|text()|processing-instructi
4
Fetch URL
Read File
Stylesheet Name (optional, GUI use only):
XSLT Messages Variable Prefix:
xslt
OK (New Prefix)
5
OK
Cancel

6. From the policy assertion tree, drag and drop the ***Policy Assertions/Policy Logic/Set Context Variable*** assertion so that it's assertion #5 in the ***Loopback Service*** policy in the policy editor, and in the Context Variable Properties dialog, in the Variable Name field, enter ***request***. In the Expression field, copy/paste the following content, and then click the ***OK*** button:

   ${originalXML.mainpart}



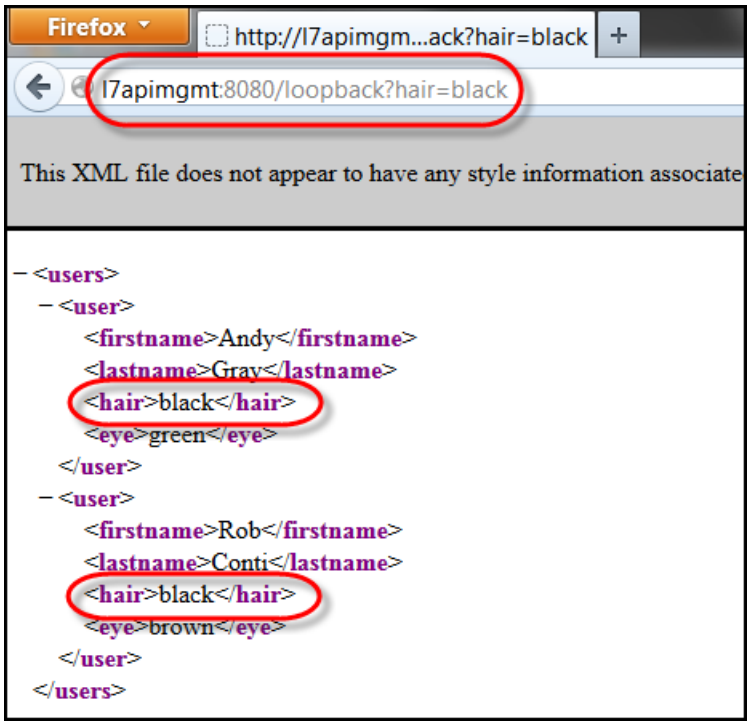
Note: This effectively copies the String value of the originalXML message context variable to the request context which is copied to the response returned to the user later in policy.

7. On the policy editor toolbar, click the ***Save and Activate*** button.

8. In a browser, navigate to the following URL to call the Loopback and to change hair color to black:

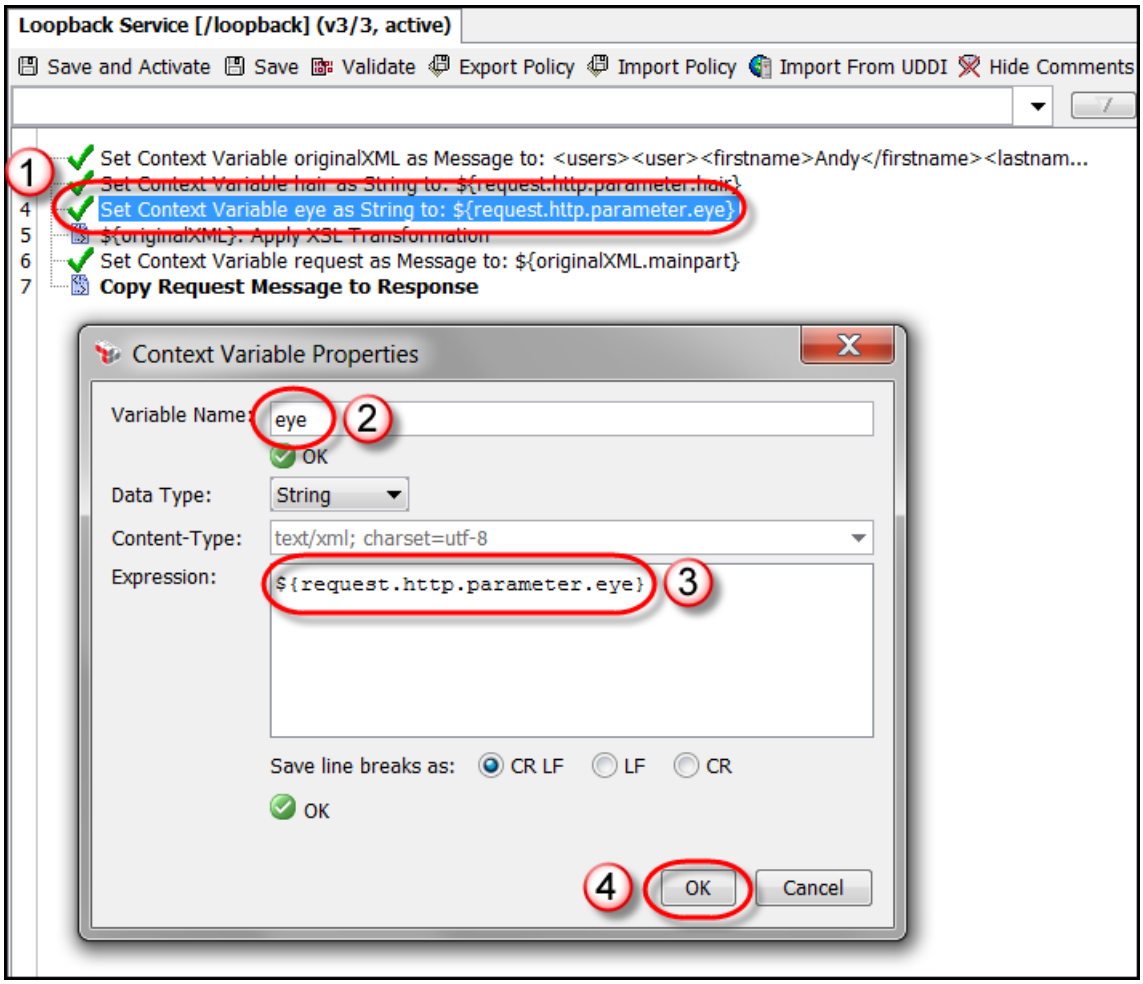
   http://<gateway>:8080/loopback?hair=black

   Replace <gateway> with the host name or IP address of your gateway.



9. In Policy Manager, from the policy assertion tree, drag and drop the ***Policy Assertions/Policy Logic/Set Context Variable*** assertion so that it's assertion #4 in the ***Loopback Service*** policy in the policy editor, and in the Context Variable Properties dialog, in the Variable Name field, enter ***eye***. In the Expression field, copy/paste the following content, and then click the ***OK*** button:

   ${request.http.parameter.eye}



10. In the policy editor, double click on assertion #5, the ***Apply XSL Transformation*** assertion, and in the XSL Transformation Properties dialog, in the XSLT Stylesheet field, copy/paste the following content (changes to the original XSL have been highlighted), and then click the ***OK*** button:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- by using xsl:param us can pass in context variables into the XSLT transformation. If
there exists a context variable prior to the execution of the Apply XSL Transformation
assertion that matches the value of the “name” inside the xsl:param the value of that context
variable I passed into the XSL transformation -->
  <xsl:param name="hair"/>
  <xsl:param name="eye"/>

  <!-- Search for the “hair” element in the XML document and replace its value with the new
value passed in via the browser -->
  <xsl:template match="/users/user/hair">
    <xsl:element name="hair">
      <xsl:value-of select="$hair"/>
    </xsl:element>
  </xsl:template>

  <!-- Search for the “eye” element in the XML document and replace its value with the new
value passed in via the browser -->
  <xsl:template match="/users/user/eye">
    <xsl:element name="eye">
      <xsl:value-of select="$eye"/>
    </xsl:element>
  </xsl:template>

  <!-- Identity transform -->
  <xsl:template match="@*|*|processing-instruction()|comment()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()|processing-instruction()|comment()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

11. On the policy editor toolbar, click the ***Save and Activate*** button.

12. In a browser, navigate to the following URL to call the Loopback and to change hair color to black:

    http://<gateway>:8080/loopback?hair=black&eye=blue

    Replace <gateway> with the host name or IP address of your gateway.

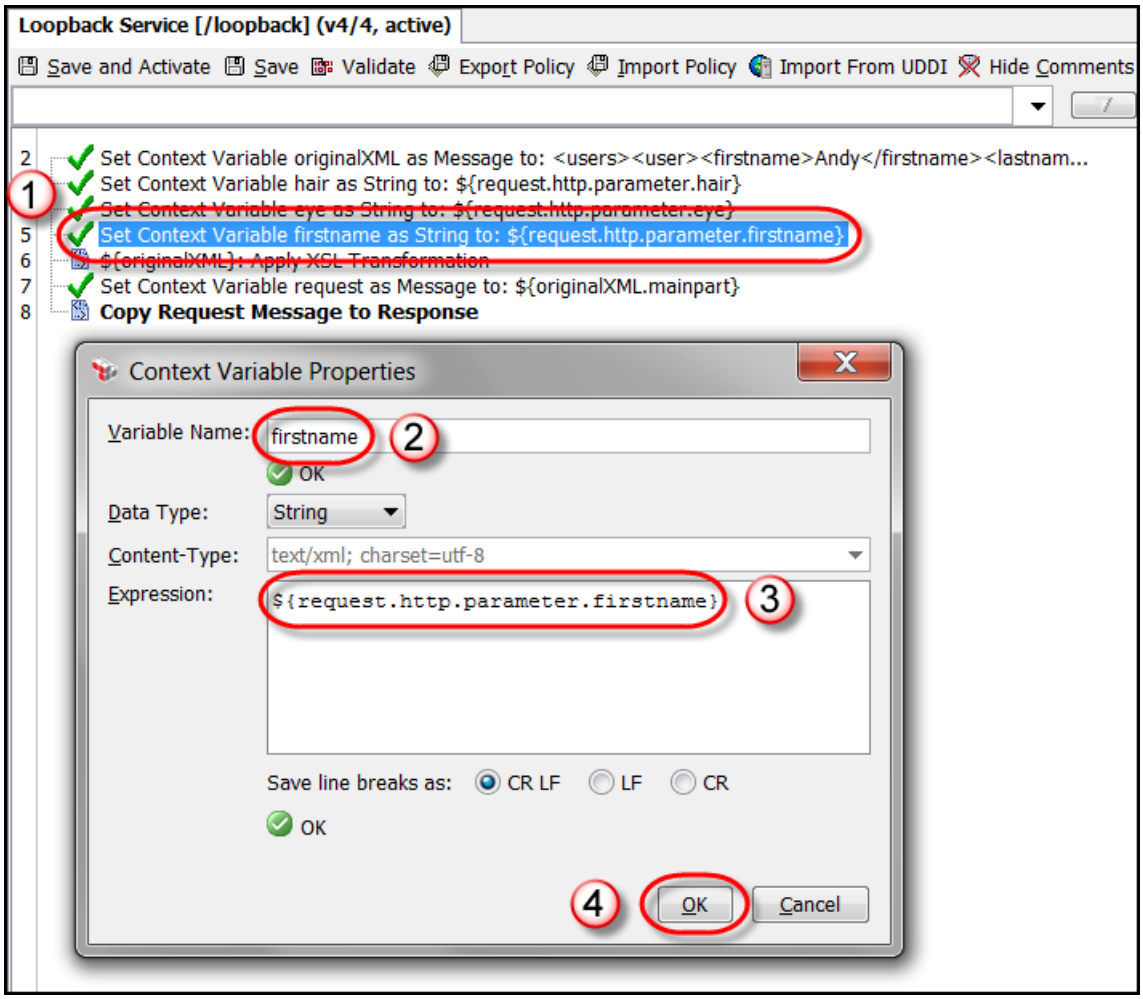
    Try different combinations of hair and eye color. For example:

    http://<gateway>:8080/loopback?hair=black&eye=yellow
    http://<gateway>:8080/loopback?hair=blonde&eye=blue
    http://<gateway>:8080/loopback?hair=red&eye=green

    Notice that all users in the originalXML document get their eye color and hair color changes. This is because we're matching on and transforming all hair and eye elements regardless of user. Next we will expect an additional HTTP parameter to identify the user, and we'll selectively change just that user's hair and eye colors.

13. In Policy Manager, from the policy assertion tree, drag and drop the ***Policy Assertions/Policy Logic/Set Context Variable*** assertion so that it's assertion #5 in the ***Loopback Service*** policy in the policy editor, and in the Context Variable Properties dialog, in the Variable Name field, enter ***firstname***. In the Expression field, copy/paste the following content, and then click the ***OK*** button:

    ${request.http.parameter.firstname}



14. In the policy editor, double click on assertion #6, the ***Apply XSL Transformation*** assertion, and in the XSL Transformation Properties dialog, in the XSLT Stylesheet field, copy/paste the following content (changes to the original XSL have been highlighted), and then click the ***OK*** button:

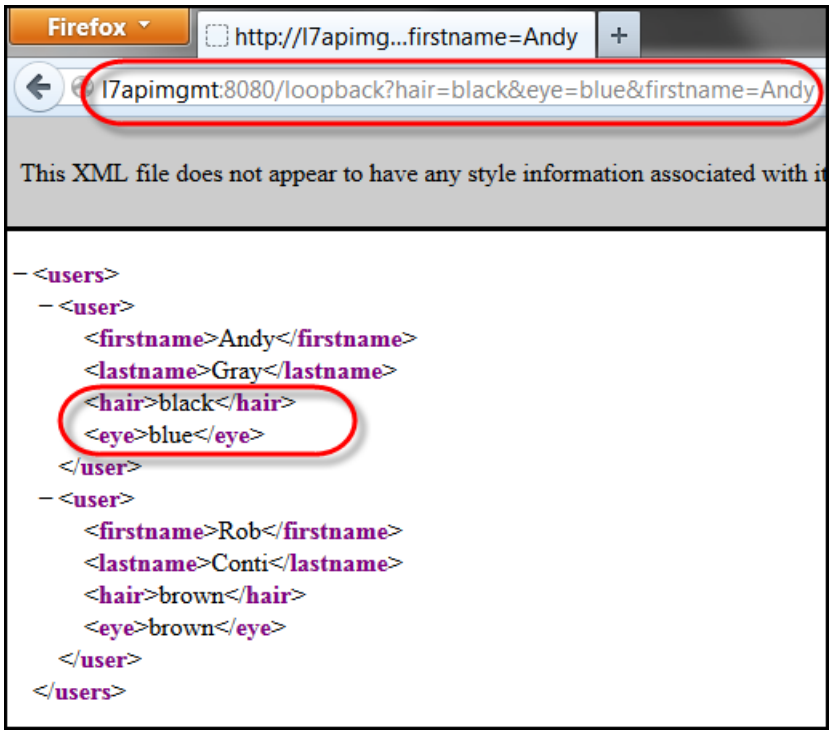
```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- by using xsl:param us can pass in context variables into the XSLT transformation. If there exists a context variable prior to the execution of the Apply XSL Transformation assertion that matches the value of the “name” inside the xsl:param the value of that context variable I passed into the XSL transformation -->
  <xsl:param name="hair"/>
  <xsl:param name="eye"/>
  <xsl:param name="firstname"/>

  <!-- Search for the “firstname” element in the XML document equal to the inbound $firstname and replace its value with the new value passed in via the browser. We are searching with xpath all the user elements and then comparing the firstname within that element with the inbound $firstname and then copying the firstname and lastname while changing the eye and hair color to the inbound values of $eye and $hair -->
  <xsl:template match="/users/user">
    <xsl:element name="user">
      <xsl:choose>
        <xsl:when test="./firstname=$firstname">
          <xsl:element name="firstname"><xsl:value-of select="./firstname"/></xsl:element>
          <xsl:element name="lastname"><xsl:value-of select="./lastname"/></xsl:element>
          <xsl:element name="hair"><xsl:value-of select="$hair"/></xsl:element>
          <xsl:element name="eye"><xsl:value-of select="$eye"/></xsl:element>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="*"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:element>
  </xsl:template>

  <!-- Identity transform -->
  <xsl:template match="@*|*|processing-instruction()|comment()">
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()|processing-instruction()|comment()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

15. On the policy editor toolbar, click the ***Save and Activate*** button.

16. In a browser, navigate to the following URL to call the Loopback and to change hair color to black:

    http://<gateway>:8080/loopback?hair=black&eye=blue&firstname=Andy

    Replace <gateway> with the host name or IP address of your gateway.



17. Per ***Layer 7 Tutorials - Getting Started/Basic Policy Concepts/Policy Authoring/Policy Revisions***, and as demonstrated at the end of ***Tutorial 1 - Deploy Tutorial Services***, comment the active policy revision of the ***Loopback Service*** with the comment, **Tutorial 11 Complete**.
18. You are done with this tutorial.

11.4 Additional Context

None