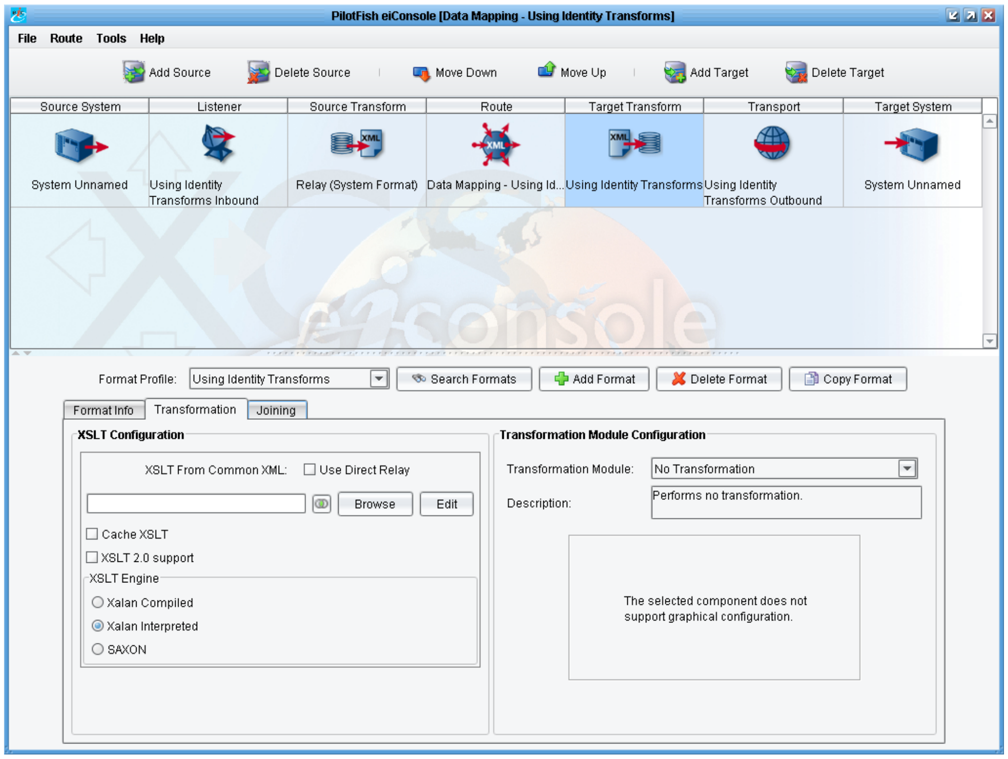
Data Mapping – Using Identity Transforms

Overview

In this tutorial we'll cover the use of identity transforms to make discrete modifications to XML documents without needing redundant mapping. This tutorial expands on concepts from "Data Mapping – Using Templates," so users are expected to be familiar with that content.

Steps

Start by creating and configuring a new Route with a Directory Listener / Transport pair. Create a new Format on the Target Transform stage called "PeopleA to PeopleH":



Format Profile: Using Identity Transforms

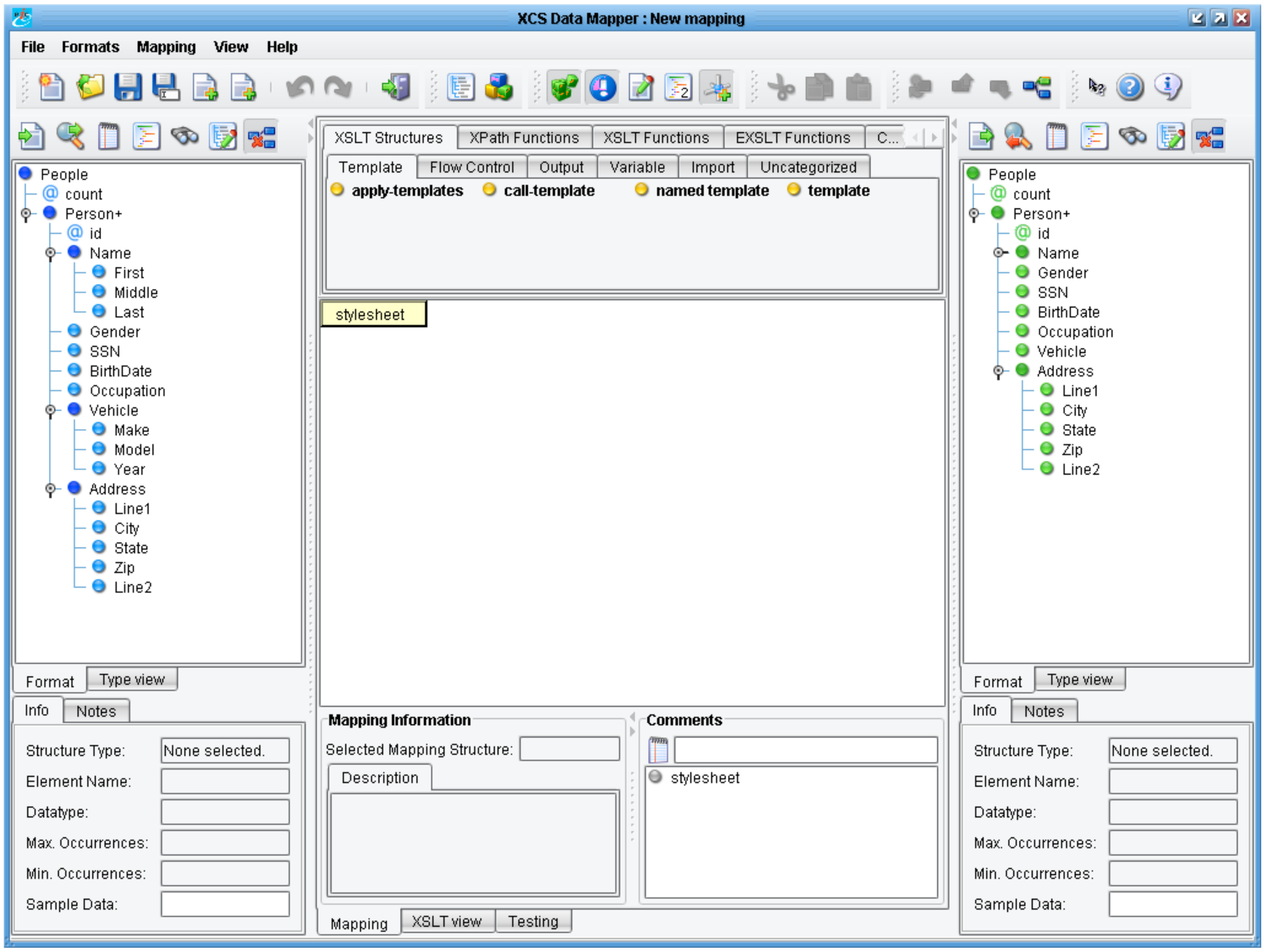
XSLT From Common XML: ☐ Use Direct Relay

☐ Cache XSLT
☐ XSLT 2.0 support
XSLT Engine
○ Xalan Compiled
● Xalan Interpreted
○ SAXON

Transformation Module: No Transformation
Description: Performs no transformation.

The selected component does not support graphical configuration.

Click “Edit” to open the Data Mapper. Load “PeopleA.xml” as the Source format and “PeopleH.xml” as the Target format:

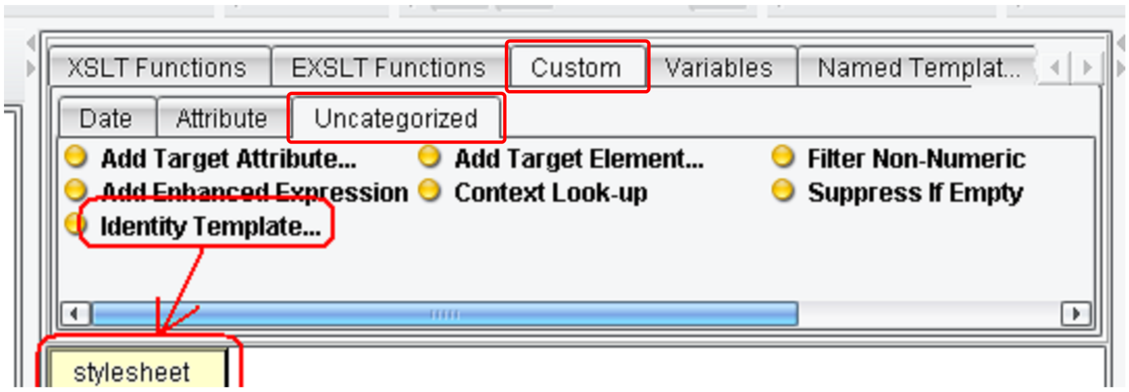


Structure Type:	None selected.
Element Name:	
Datatype:	
Max. Occurrences:	
Min. Occurrences:	
Sample Data:	

Our Source and Target formats are very similar in this case. The only difference is that the Target format's “Vehicle” element is condensed into a single tag, which is a concatenation of Make and Model. In previous tutorials, we handled such mappings by completely mapping the Source and Target formats along with all of their elements. However, there's a concept in XSLT called an “identity transform,” which is a template that matches and copies all elements, attributes, and text recursively, as-is. Any templates added in addition to this template are the “exceptions to the rule;” their elements are handled according to those added templates.

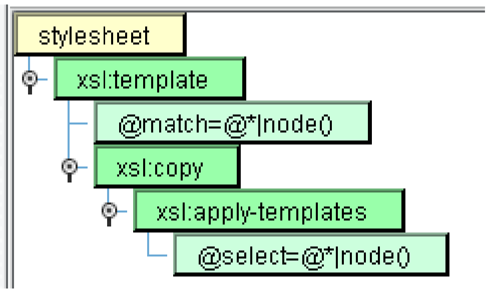
You may have to temporarily match “People” into your “stylesheet” before you can add the ‘Identity Template’.

Drag the template from “Custom” → “Uncategorized” → “Identity Template” onto the “stylesheet” element:



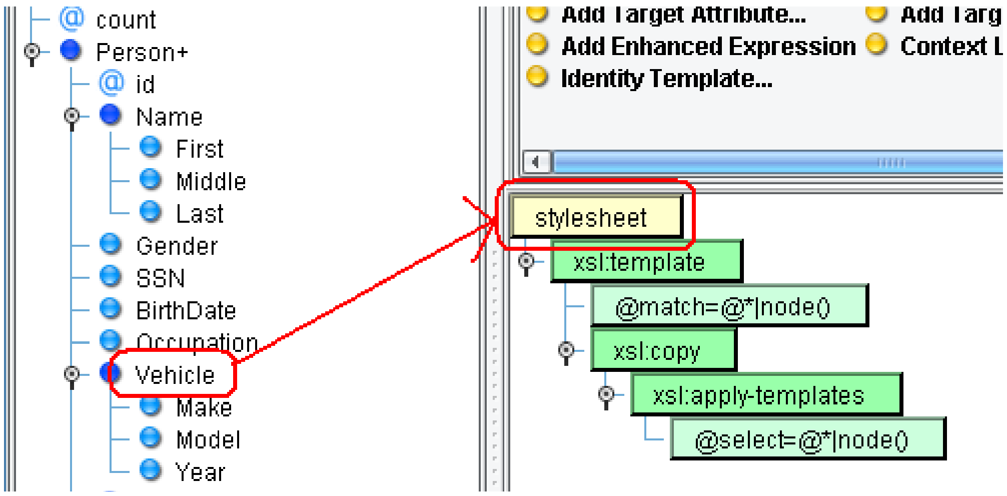
This template's expression matches all attributes (@), all child elements (*), and all nodes (node()). The “|” part of the expression is a “join” between the two halves (@* and node()).

The “copy” element under the template copies the current node, while the “apply-templates” and expression underneath recursively evaluate all child nodes of the current node.

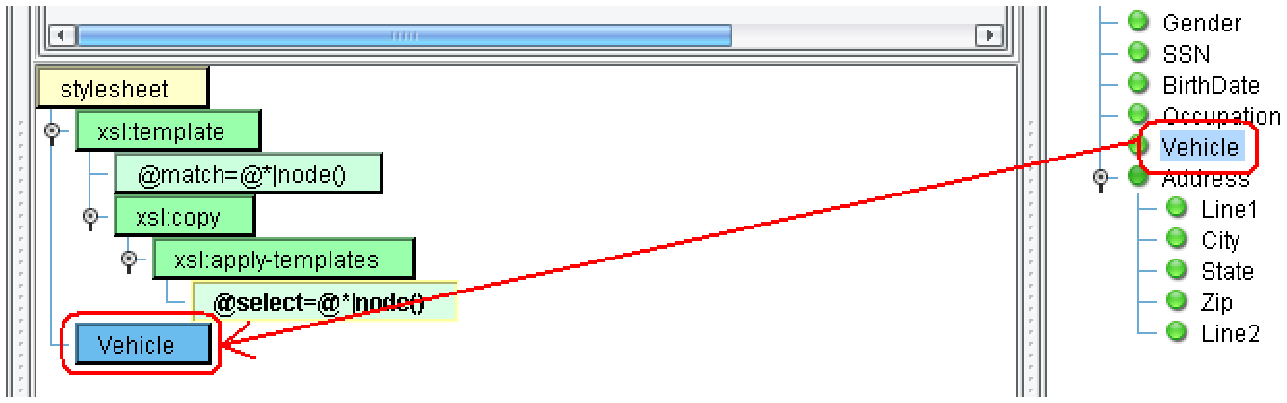


If you were to run the transform now, you would see that the result document is identical to the source document; the identity transform creates a perfect copy.

To make changes to the mapping, we need only create templates for the elements we wish to alter. To change the handling of Vehicle, drag it from the Source onto the stylesheet:

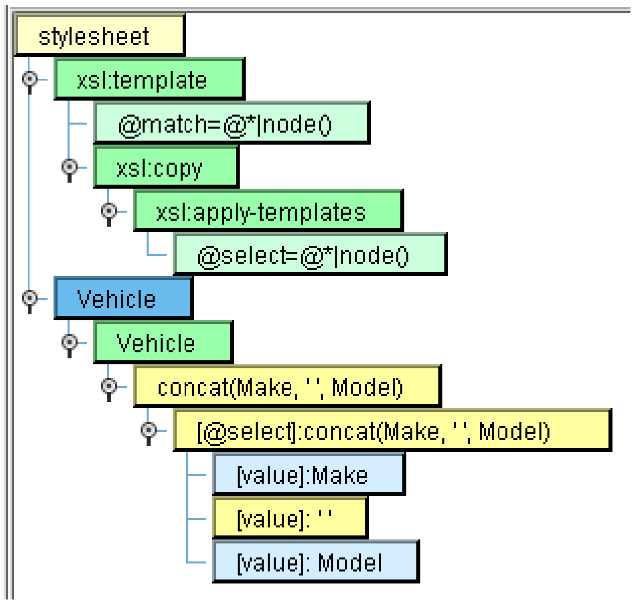


If you were to run the transform now, you would see that the result document is identical except that the “Vehicle” elements are all missing; we've given “Vehicle” special handling, but have not created a new Vehicle element in its stead. Drag “Vehicle” from the Target format onto the “Vehicle” template:



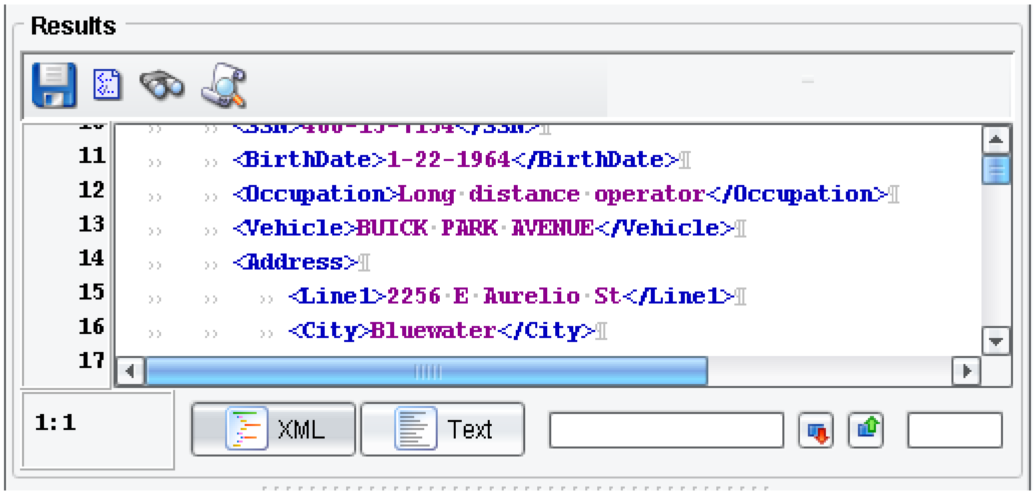
Our mapping will now create a copy of the document with an empty Vehicle element. We are copying all nodes that are not Vehicle. Now we just need to concatenate our Make and Model. Drag the “concat” function from the tool palette onto the Vehicle element and modify the expression to combine the Make and Model (separated by a space, of course):

concat(Make, ' ', Model)



```
stylesheet
  xsl:template
    @match=@*|node()
    xsl:copy
      xsl:apply-templates
        @select=@*|node()
  Vehicle
    Vehicle
      concat(Make, ' ', Model)
        [@select]:concat(Make, ' ', Model)
          [value]:Make
          [value]: ' '
          [value]: Model
```

If we now execute our transform, we should see only the Vehicle element altered:



```
<BirthDate>1-22-1964</BirthDate>
<Occupation>Long distance operator</Occupation>
<Vehicle>BUICK PARK AVENUE</Vehicle>
<Address>
    <Line1>2256 E Aurelio St</Line1>
    <City>Bluewater</City>
```

Identity transforms are an extremely useful tool in transformation. They allow you to make discrete, tactical modifications to a document without needlessly replicating mapping. In the eiConsole, they are most frequently used on the Target Transform stage to “tweak” a previously transformed document so that it conforms to minor changes required by a particular target system.