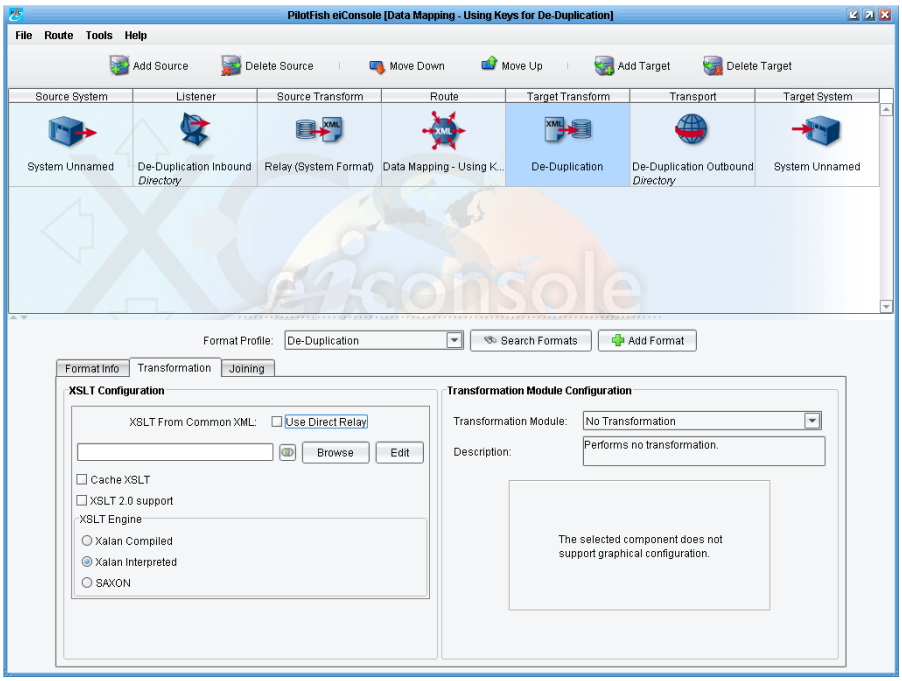
Advanced Mapping – Using Keys for De-Duplication

Overview

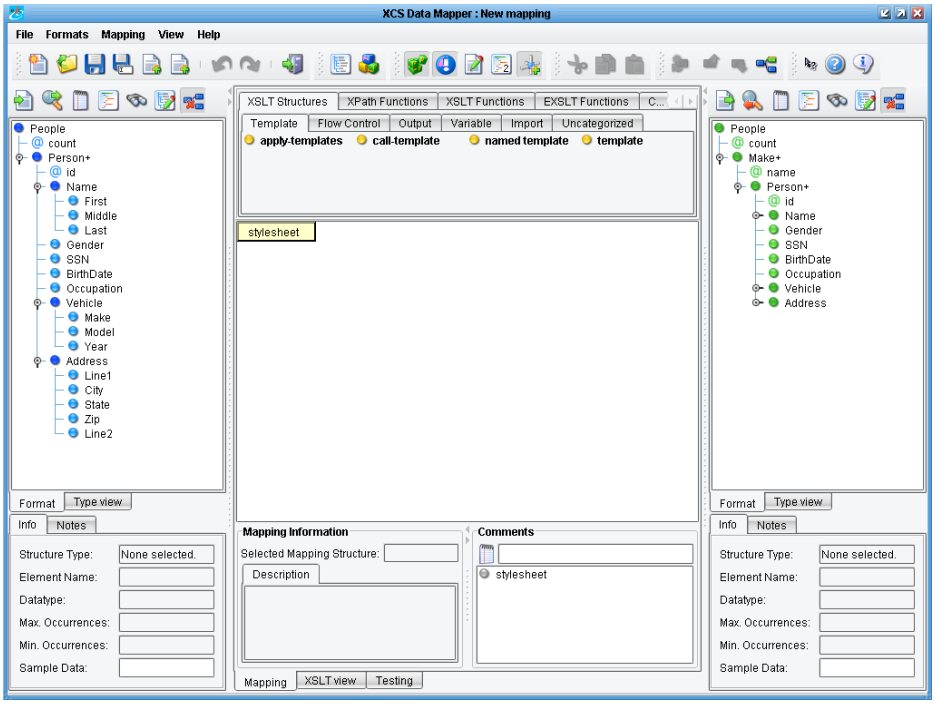
In this tutorial we'll cover the use of keys and the "Muenchian method" to handle de-duplication during XSLT iteration. This tutorial expands on concepts covered in "Data Mapping – Using Templates," so users are expected to be familiar with that material.

Steps

Start by creating and configuration a new Route with a Directory Listener / Transport pair. Add a new Format at the Target Transform staged called "PeopleA to PeopleI":



Click the "Edit" button to open the Data Mapper. Load in "PeopleA.xml" for the Source format and "PeopleI.xml" for the Target format:

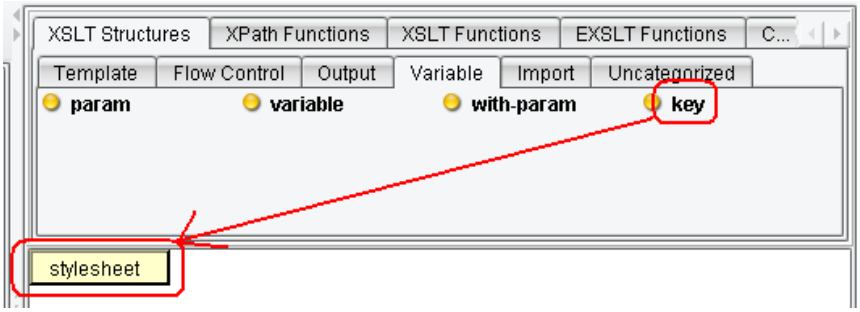


The structure in the Target format is similar to the Source, except that Person elements are grouped under a “Make” element. Each such element represents a Vehicle Make. The goal for this mapping is to group Person elements according to their respective Vehicle Make elements. The intuitive, albeit incorrect, way of doing this would be to iterate over each Person or Make element from the Source and then iterate again for each matching Person. However, this would create duplicate groups, as each such Make might appear multiple times.

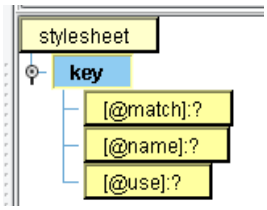
The solution is to use something called the “Muenchian method,” which makes use XSLT keys to iterate uniquely over some set of elements. Conceptually, this approach is somewhat complex, though it is too useful to ignore.

The first thing we'll need to do is to create a “key.” XSLT keys are structures that are conceptually similar to the “Dictionary” in .NET languages (such as C#) or the “Map” in Java. A key is a collection of values “keyed” according to some other value. For example, if we were to create a key mapping “Gender” values to “Person” elements, we would have key entries for “Male” and “Female.” Each such key would point / map to a set of Person elements. Passing in “Male” to the key object (via the key function) would return the Person node set.

For our case, our key will map the “Make” element to “Person” elements. Drag XSLT “Structures” → “Variable” → “key” onto the “stylesheet” element:



The created structure has “match,” “name,” and “use” attributes:



“match” is the expression we wish to match against and use as values in our key object. In our case, that will be “Person” elements. Modify the expression to read:

Person

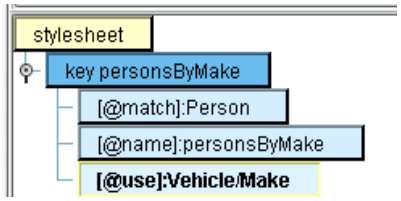
“name” is the name of the key object we'll use to reference it by later. Modify it to read:

personsByMake

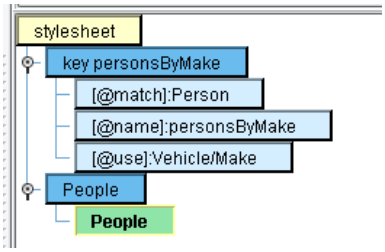
“use” is the expression, relative to the “match” expression (each Person), to use as key entries. Modify it to read:

Vehicle/Make

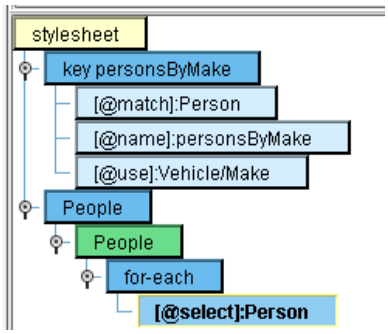
Your key structure should now look like this:



Next, we'll wish to map our basic Source and Target root elements:



We'll want to iterate over each Person in our Source, so provide the for-each expression and its initial expression:



Double-click on the “select” attribute to modify the for-each expression. Change it to read:

Person[generate-id(.) = generate-id(key('personsByMake', Vehicle/Make)[1])]

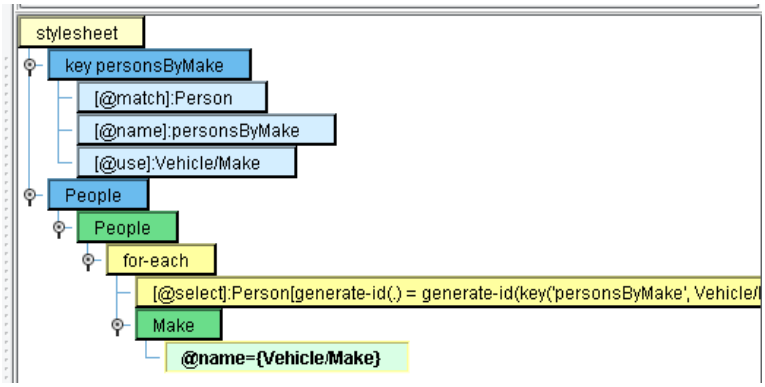
As expressions go, this one is obviously a bit complicated, so we'll break it down. We've added a predicate to Person which tests if the results of the function “generate-id(.)” match the results of another such function. “generate-id(.)” returns a unique ID for the node passed as its argument. The XSLT specification provides no information as to the structure of that ID; only that it is unique for the given node in a particular XML document. That means that, within a given transform, values are unique, but should not be used outside of those purposes, as two unrelated nodes in two disparate documents could share such an ID.

The second part of the expression is the “key” function called against our created key with the current Person's “Make” as the key. This will return a node set containing each Person node with a matching Make element. We then use the “[1]” value to grab the first such node, guaranteed to be in document order. Finally, we get the ID of this node.

The result of all of this is a predicate which tests to see if a given Person is the first such Person with a given Make. Our for-each therefore iterates only once, uniquely, for each Make. We've effectively managed de-duplication as a result.

Next we can map “Make” from our Target onto the “for-each” and populate its “@name” value with the Source “Make”:

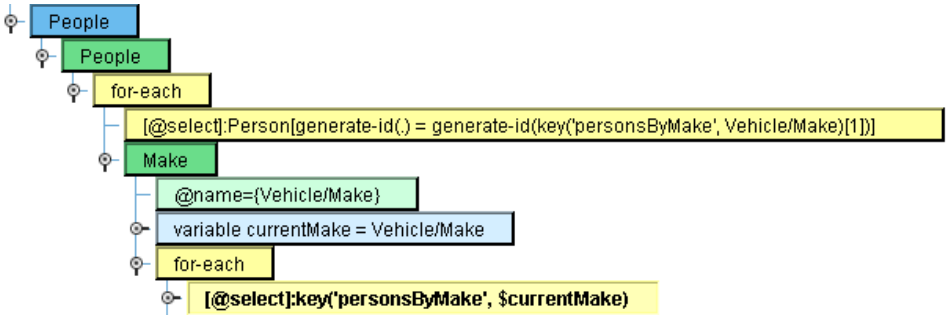
{Vehicle/Make}



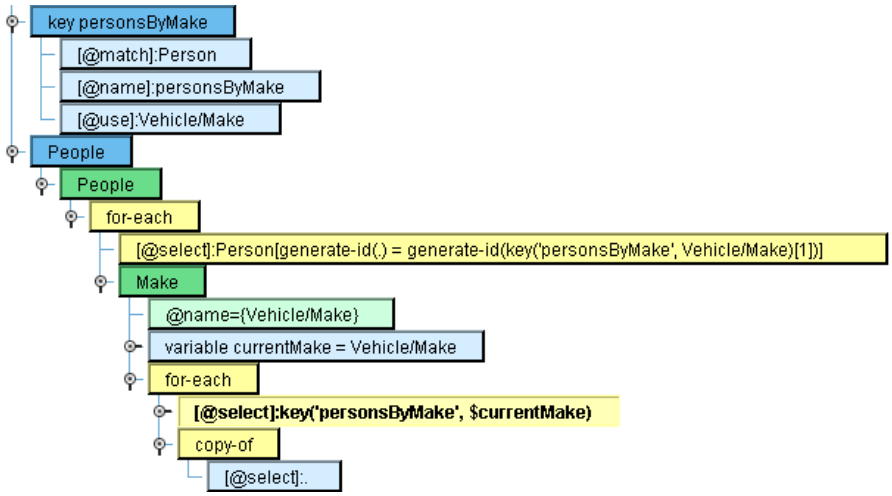
We'll now create a variable to store the current Make, then iterate over each Person with a matching Make:

$currentMake = Vehicle/Make

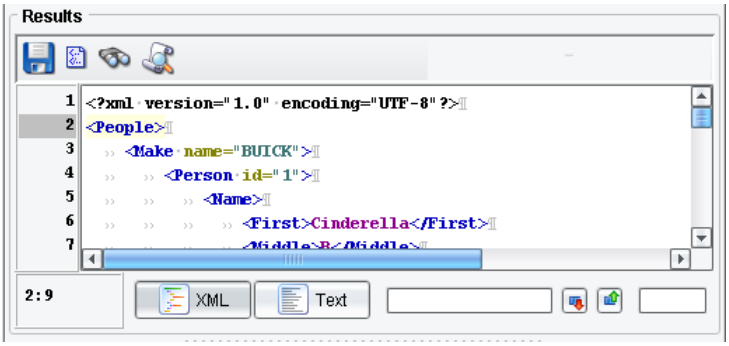
key('personsByMake', $currentMake)



Finally, we can use the XSLT "copy-of" ("XSLT Structures" → "Output") instruction and the current node (.) as an argument to copy whatever Person we're iterating over:



Finally, we can transform to see the results:



Keys and the Muenchian method can be extraordinarily useful for such circumstances, and while the syntax and concepts can be a little difficult to grasp, they are well worth learning.