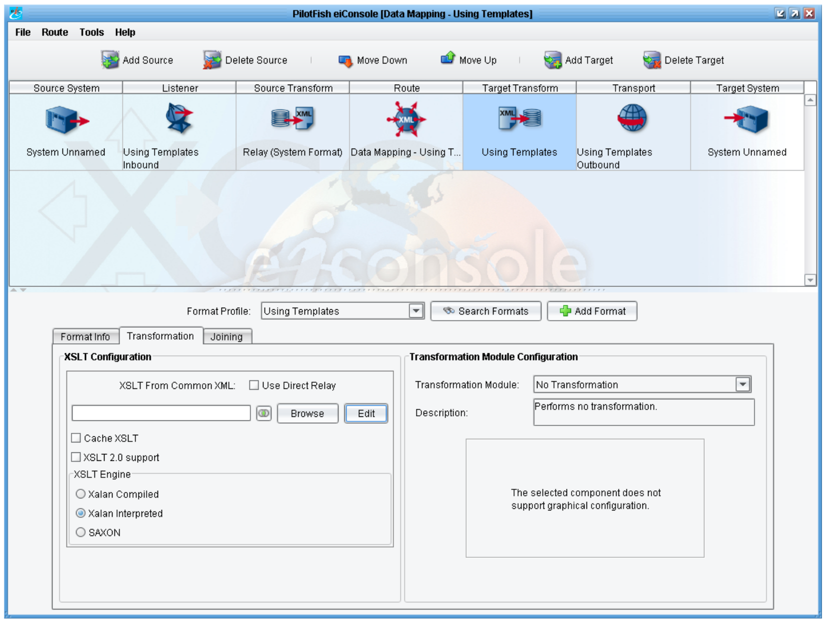
Data Mapping – Using Templates

Overview

In this tutorial we'll cover the use of templates in the Data Mapper and XSLT. This tutorial expands on concepts demonstrated in "Data Mapping – Using Variables," so users are expected to be familiar with that content. We'll show how to use templates as "callable functions," as well as how to structure XSLT somewhat less linearly.

Steps

Create and configure a new Route with a Directory Listener / Transport pair. Add a new Format on the "Target Transform" stage; we'll be naming ours "PeopleA to PeopleC":



Format Profile: Using Templates

XSLT From Common XML: ☐ Use Direct Relay

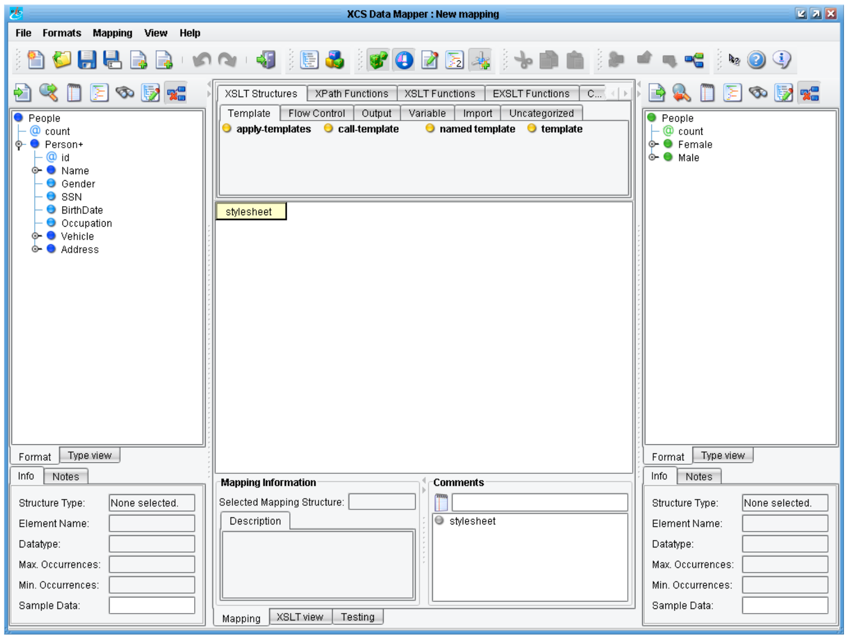
☐ Cache XSLT
☐ XSLT 2.0 support
XSLT Engine
○ Xalan Compiled
● Xalan Interpreted
○ SAXON

Transformation Module: No Transformation
Description: Performs no transformation.

The selected component does not support graphical configuration.

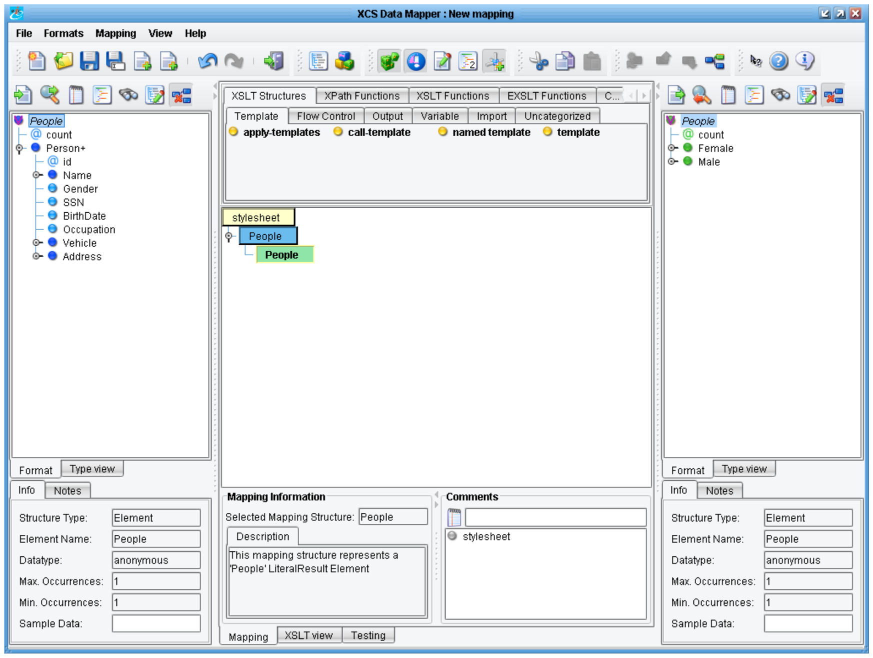
Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

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



As usual, begin by mapping the root element from the Source onto the “stylesheet” element, and then the root element from the Target onto that:

Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

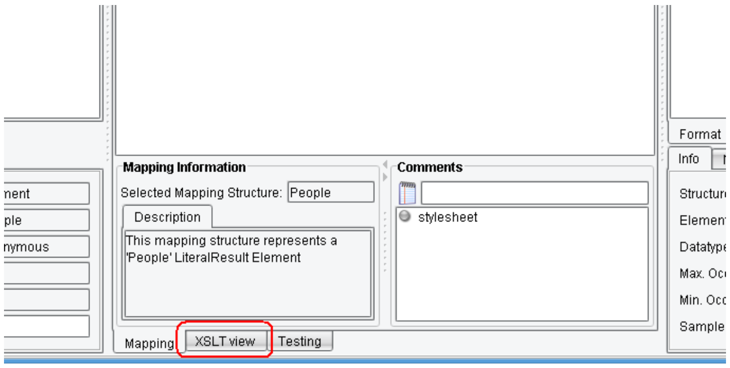

XCS Data Mapper : New mapping
File Formats Mapping View Help
People
count
Person+
id
Name
Gender
SSN
BirthDate
Occupation
Vehicle
Address
XSLT Structures XPath Functions XSLT Functions EXSLT Functions
Template Flow Control Output Variable Import Uncategorized
apply-templates call-template named template template
stylesheet
People
People
count
Female
Male
Format Type view
Info Notes
Mapping Information
Comments
Mapping XSLT view Testing

Structure Type: Element
Element Name: People
Datatype: anonymous
Max. Occurrences: 1
Min. Occurrences: 1
Sample Data:

Selected Mapping Structure: People
Description
This mapping structure represents a 'People' LiteralResult Element

stylesheet

In previous tutorials, we've immediately starting mapping items from the Target format, sometimes directly and sometimes using constructs such as "for-each." We'll do things a bit differently this time. Click the "XSLT View" tab:

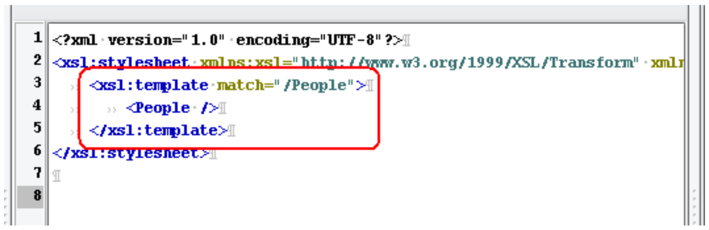


Selected Mapping Structure: People
Description
This mapping structure represents a 'People' LiteralResult Element

stylesheet

If you look at the XSLT we've thus far generated using the graphical mapping view, it should look something like this:

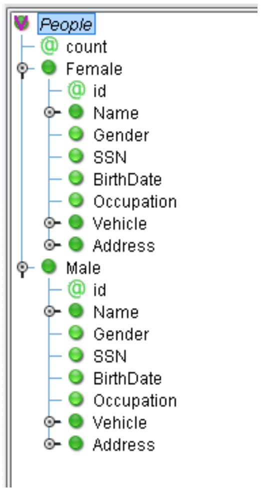
Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlr
3     <xsl:template match="/People">
4         <People />
5     </xsl:template>
6 </xsl:stylesheet>
7
8
```

When we dragged the Source element, People, onto “stylesheet,” it created an XSLT instruction called “template” with a “match” attribute. In XSLT, a template is a handler for a particular XPath expression. In this case, we're handling root-level elements named “People,” of which there should only be one. This template is therefore evaluated only a single time, producing a single “People” element in the result document (line 4). Most of our mapping so far has taken place inside this template. However, there's another, more modular approach to XSLT, which is to use templates to break your XSLT into portions which call one another.

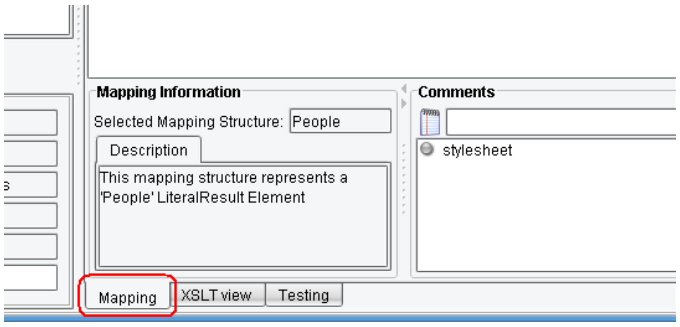
If we examine our Target format, we can see that the Person elements from the Source are replaced with individual instances of genders: “Male” and “Female”:



```
People
├─ @ count
├─ Female
│  ├─ @ id
│  ├─ Name
│  ├─ Gender
│  ├─ SSN
│  ├─ BirthDate
│  ├─ Occupation
│  ├─ Vehicle
│  └─ Address
└─ Male
   ├─ @ id
   ├─ Name
   ├─ Gender
   ├─ SSN
   ├─ BirthDate
   ├─ Occupation
   ├─ Vehicle
   └─ Address
```

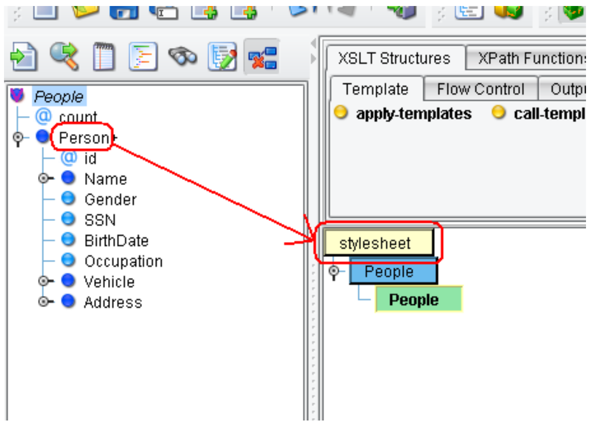
Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

In previous tutorials, we handled this by creating two separate for-eaches or using a single for-each with a choose / when / otherwise block. But let's support we want to do this using templates. Return to the graphical view:

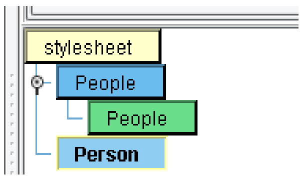


This mapping structure represents a 'People' LiteralResult Element

This time, drag "Person" from the Source onto the "stylesheet" element:



This will create another "template" element, which in the graphical view will show up as a blue "Person" element, sibling to "People":



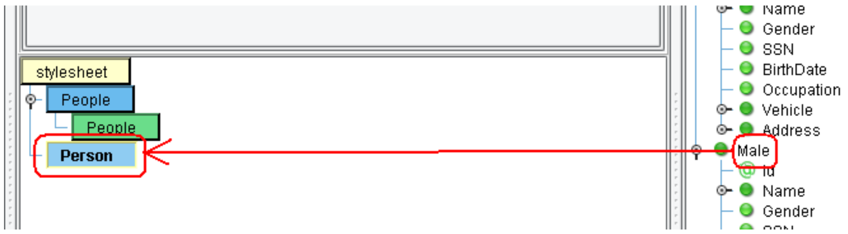
Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

However, let's alter the template match. Double-click on the Person element and change it to read:

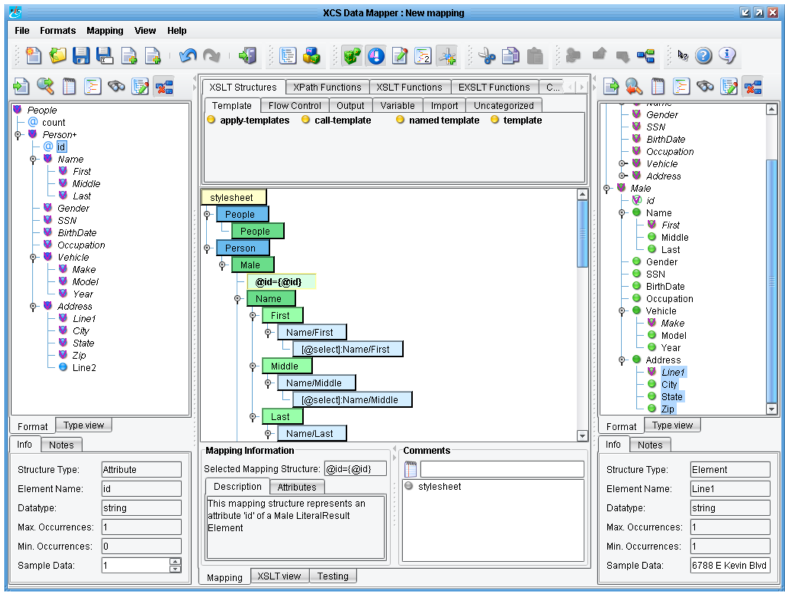
    Person[Gender = 'Male']

Remember to hit enter / return to save the changes. We've modified the expression so that, instead of just grabbing Person elements, it only grabs Person elements where that Person's "Gender" element is equal to the literal value "Male." The [ ] brackets define what is known in XPath as a "predicate" - a condition.

Now, drag "Male" from the Target onto the Person element:



Using the same rules as previous mappings, map the contents of Person from the Source onto the contents of Male from the Target:


XCS Data Mapper : New mapping
File Formats Mapping View Help
XSLT Structures XPath Functions XSLT Functions EXSLT Functions
Template Flow Control Output Variable Import Uncategorized
apply-templates call-template named template template
stylesheet
People
Person
Male
@id={@id}
Name
First
Name/First
[@select]:Name/First
Middle
Name/Middle
[@select]:Name/Middle
Last
Name/Last
Format Type view
Info Notes
Mapping XSLT view Testing

Structure Type:	Attribute
Element Name:	id
Datatype:	string
Max. Occurrences:	1
Min. Occurrences:	0
Sample Data:	1

Mapping Information

Selected Mapping Structure: @id={@id}

Description

This mapping structure represents an attribute 'id' of a Male LiteralResult Element

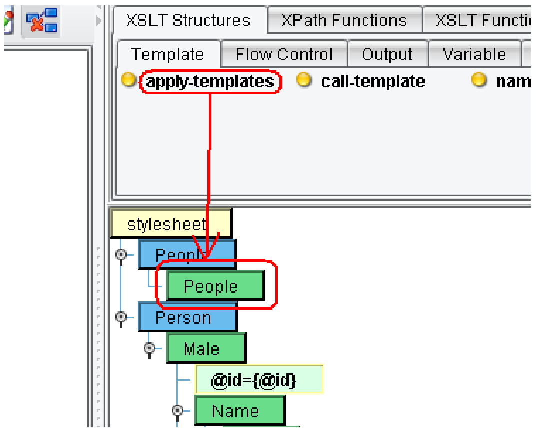
Comments

stylesheet

Structure Type:	Element
Element Name:	Line1
Datatype:	string
Max. Occurrences:	1
Min. Occurrences:	1
Sample Data:	6788 E Kevin Blvd

Repeat the exercise with the Female element; map another Person, change its expression so that the predicate checks for "Female," and then map Female and its values from the Source and Target.

Now, as-is, this mapping won't make use of the new template we've added. The first template that matches is evaluated. We need to add a call for our new template. Drag XSLT Structures → Template → apply-templates onto the green "People" element in our mapping:

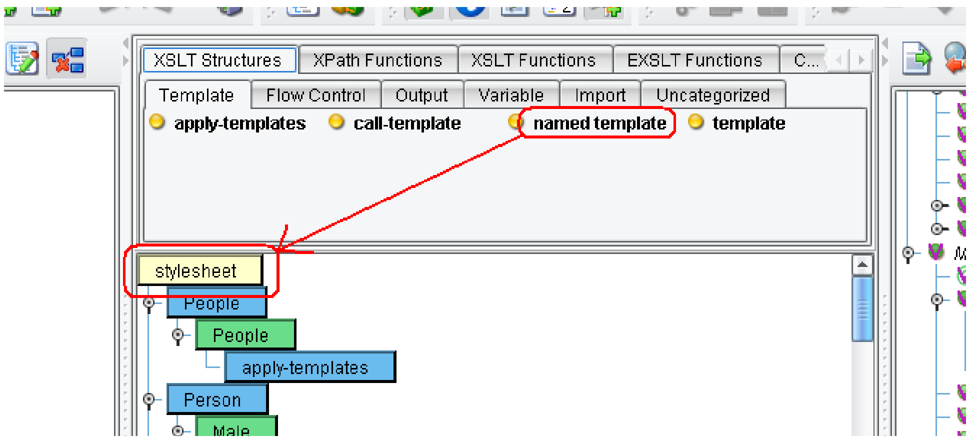


Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

This will create the necessary call for XSLT to evaluate the current context elements against existing templates. Incidentally, apply-templates can also be invoked with a “select” attribute to restrict which elements are evaluated against templates.

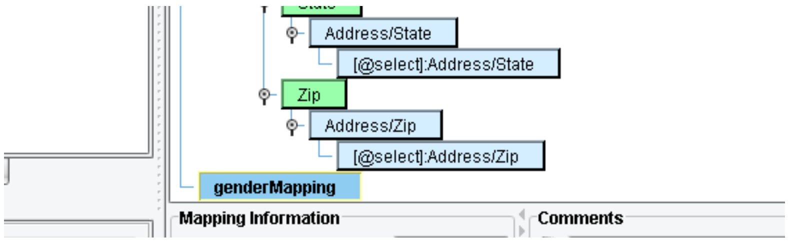
We've now recreated a previous mapping exercise using templates. However, templates can also be used another way: as named “functions.” Let's suppose that we want to arbitrarily map the Gender field from “Male” / “Female” to “M” / “F.” In previous tutorials, we handled this through conditions, tabular mappings, etc. For this case, we'll do it through conditions and also through a template.

Start by dragging XSLT Structures → Template → “named template” onto “stylesheet”:

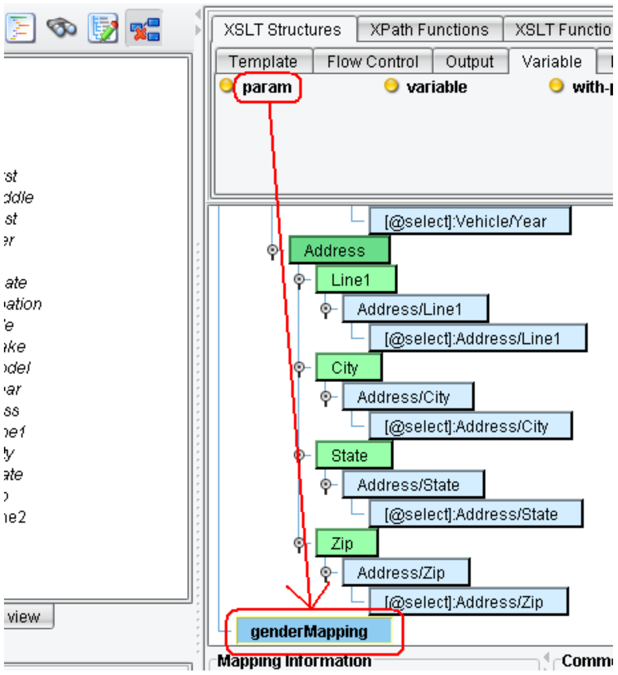


Double-click on the created template and name it. We'll call ours “genderMapping”:

Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

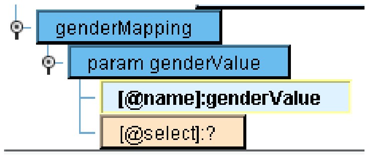


Next, we'll provide a parameter to our named template. Parameters are variables for which the values are provided when the template is called. Drag XSLT Structures → Variable → "param" onto our newly created template:

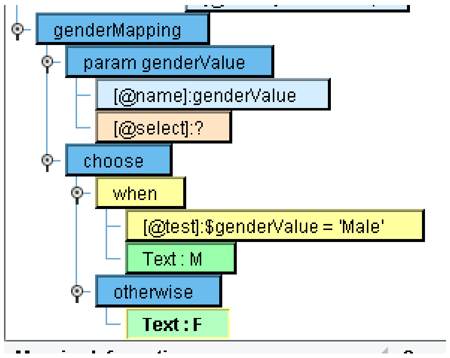


We'll need to provide a "name" value for our parameter. We can ignore the "select" attribute; this is the default value for this, if no other is provided. We'll name the parameter "genderValue":

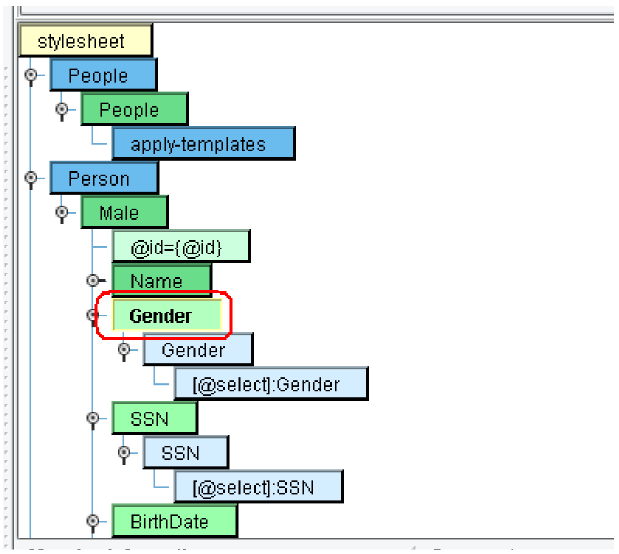
Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information



Next, we'll use a choose / when / otherwise block to test the value of "genderValue." We can use XSLT Structures → Output → "text" to output "M" or "F":

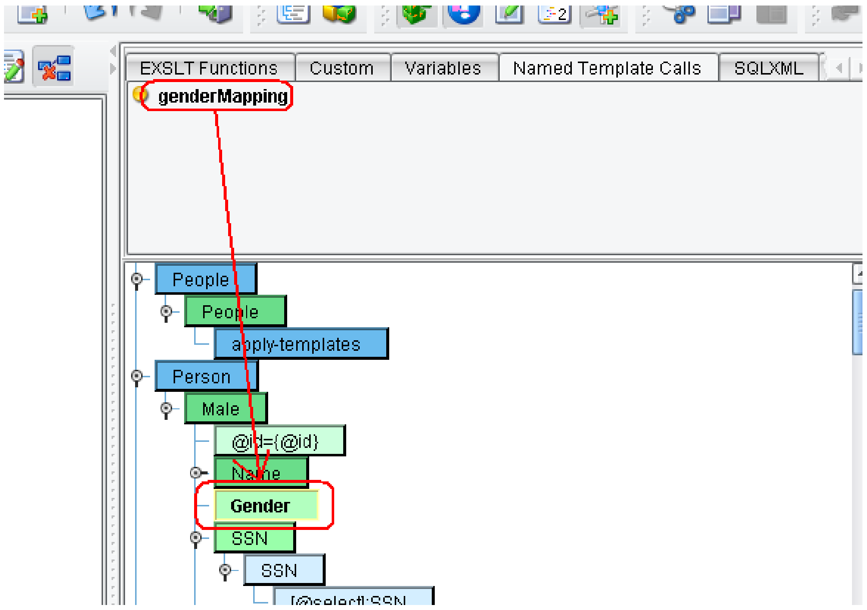


That completes our template definition. To invoke it, we can make use of a convenient tool in the Data Mapper: The "Named Template Calls" tab. Start by clicking one of the mapped Gender elements:

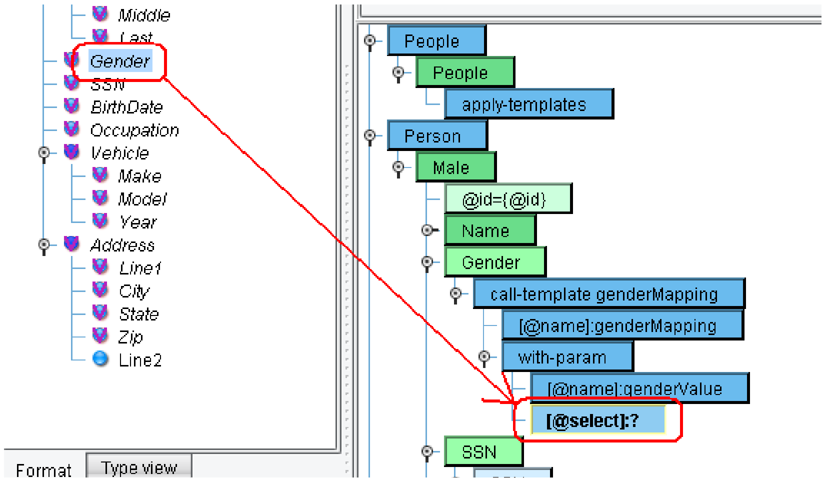


Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

Delete the mapped “Gender” value (the one in blue), then drag “genderMapping” from the “Named Template Calls” tab onto the Gender node:

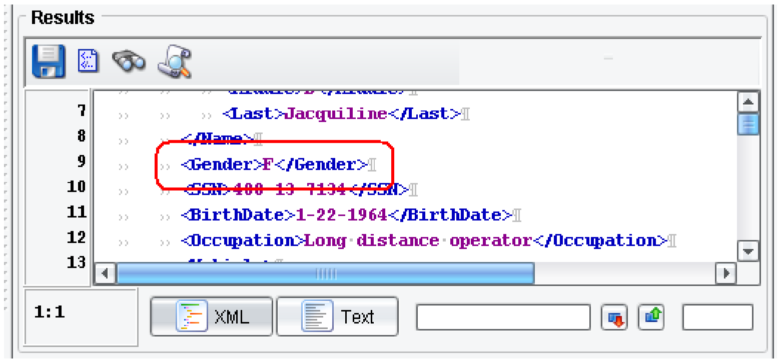


Next, drag “Gender” from the Source onto the newly mapped “with-param”'s “select” attribute:



Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information

Repeat the “Named Template Calls” part for the Female element, then test the transformation. You should see the “Male” and “Female” values mapped correctly:



```
<Last>Jacquiline</Last>
</Name>
<Gender>F</Gender>
<SSN>400-13-7134</SSN>
<BirthDate>1-22-1964</BirthDate>
<Occupation>Long distance operator</Occupation>
```

That's pretty much all there is to it. You can use templates both as ways to break up and modularize XSLT as well as to produce reusable mappings.

*** Bonus ***

One of the key principles of programming is DRY (Don’t Repeat Yourself). We have almost identical code in both our ‘Male’ and ‘Female’ templates; try moving that code into a separate ‘Named’ template.

Copyright © 2017 Duck Creek Technologies. All Rights Reserved.
No portion of this document may be copied without express written permission from Duck Creek Technologies.
Confidential and Proprietary Information