XSLT Ingest Example: Gather

Gather

This step is performed by the **gather.xsl** transform which can be found in its entirety the **example/xslt** folder. Figure 5 shows the first portion of this XSLT. We will now comment on the highlighted sections. In what follows, we will use [FnHm] to denote Figure n, Highlight m.

```
<?xml version="1.0"?>                                                    0
<xsl:stylesheet version='2.0'
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:vfx='http://vivoweb.org/ext/functions'
        xmlns:xs='http://www.w3.org/2001/XMLSchema'
        exclude-result-prefixes='xs vfx xsl'
>

<xsl:param name='extOrgIn' required='yes'/>                              1
<xsl:param name='extPerIn' required='yes'/>

<xsl:output method='xml' indent='yes' normalization-form='NFC'/>         2
<xsl:strip-space elements="*"/>

<xsl:variable name='localNamespace'>                                     3
<xsl:text>http://vivo.cornell.edu/individual/</xsl:text>
</xsl:variable>

<xsl:variable name='MYNL'>                                               4
<xsl:text>
</xsl:text>
</xsl:variable>

<xsl:variable name='extantPersons'                                       5
        select="document($extPerIn)/ExtantPersons"/>

<xsl:variable name='extantOrgs'                                          6
        select="document($extOrgIn)/ExtantOrgs"/>
```

gather.xsl Fragment 1 - Figure 5

- [F5H0] This is the usual XSLT boiler plate where namespaces and short hand prefixes are declared. In this example, the **vfx** prefix will be used to identify functions that were created to support the ingest process.
- [F5H1] Two required parameters that are file system paths that point to the accumulator files.
- [F5H2] The usual output declaration including normalization of character sets.
- [F5H3] A variable is declared here to define the namespace we use for individual URIs at Cornell. This will be used as a prefix for new URI local names. You should change this variable in **gather.xsl** to reflect your VIVO instance.
- [F5H4] A variable used to append a newline character to the output where needed. This is mainly for adjusting output for readability.
- [F5H5-6] Two variable declarations that provide a means to access the pre-existing people and organization data from **Per0.xml** and **Org0.xml** .

Next we consider the highlights of the second half of the **gather.xsl** file. This part of the code handles filtering of the source, construction of the output xml and resolving as many URIs as possible by comparing name information against the elements in **Per0.xml** in the case of people and against **Org0.xml** in the case of organizations. Figure 6 shows more of **gather.xsl** .

```
<EduRecords>
<xsl:for-each select='ROW'>

<xsl:if test='DEGREE != "" and MAJOR != "" and YEAR != "" and                    0
              FNAME != "" and LNAME != "" and INSTITUTION != ""'>
<EduRecord>
<xsl:variable name='thisId' select='@id'/>
<xsl:variable name='school' select='normalize-space(INSTITUTION)'/>

<xsl:variable name='schoolUri'
    select='$extantOrgs/org[vfx:adjust(name) = upper-case($school)]/uri'/>      1

<edId><xsl:value-of select='$thisId'/></edId>
<edUri><xsl:value-of
    select='concat($localNamespace,"EX-",$thisId)'/></edUri>
<edDeg><xsl:value-of select='normalize-space(DEGREE)'/></edDeg>
<edMajor><xsl:value-of select='normalize-space(MAJOR)'/></edMajor>
<edYear><xsl:value-of select='normalize-space(YEAR)'/></edYear>             2

<edSchoolUri>
<xsl:value-of
       select='if(count($schoolUri)>0)then $schoolUri[1] else ""'/>
</edSchoolUri>                                                              3

<edSchool><xsl:value-of select='$school'/></edSchool>
<fn><xsl:value-of select='normalize-space(./FNAME)'/></fn>
<mn><xsl:value-of select='normalize-space(./MNAME)'/></mn>
<ln><xsl:value-of select='normalize-space(./LNAME)'/></ln>
<nid><xsl:value-of select='normalize-space(NETID)'/></nid>                  4

<!-- look for matching foaf:Person uri -->
<xsl:variable name='kuri'
       select='vfx:findMatchingPeople(FNAME, MNAME, LNAME, NETID,
                                      $extantPersons)'/>
<personUri><xsl:value-of select='$kuri'/></personUri>                       5

</EduRecord>
</xsl:if>
</xsl:for-each>
</EduRecords><xsl:value-of select='$MYNL'/>
</xsl:template>

<xsl:include href='auxfuncs.xsl'/>                                          6
</xsl:stylesheet>
```

gather.xsl Fragment 2 - Figure 6

- [F6H0] Establish which source rows will pass the rejection criteria filter and start an **EduRecord**.
- [F6H1] Find any and all matching school URIs among the known organizations. Notice that the variable **school** contains a space normalized copy of the **INSTITUTION** and is further shifted to uppercase before comparison with each adjusted organization **name**. The **vfx:adjust** functio n, shown in Figure 7, applies the standard XPATH functions **normalize-space** and **upper-case**. The variable **schoolUri** is refers to a sequence of 0 or more matching organization URIs.
- [F6H2] Collect and create the required XML elements, applying **normalize-space** to fix any white space issues in the source data. You may choose to add other normalizations at this point.
- [F6H3] If **schoolUri** contains a URI then use the first one; otherwise leave **edSchoolUri** empty. If **schoolUri** has more than one term then there are duplicate entries in **Org0.xml**. This is not the case for this example. However steps must be taken to prevent duplicates by properly maintaining organization triples in VIVO.
- [F6H4] Since we may not find a URI for the school or person, we include that as an empty element along with the name parts and netid of the person who received the degree in the output XML for downstream remediation.

- [F6H5] In this step we look for a matching person by calling the name matching function **vfx:findMatchingPeople** which will return a URI or the empty string. We will describe this function shortly. Several versions of this function are included in the source code so that the reader can experiment.
- [F6H6] This shows the end of the **gather.xsl** transform with the inclusion of a file of auxiliary functions that contains the definitions of **vfx:adjust** and **vfx:findMatchingPeople** and other functions. The function **vfx:findMatchingPeoplel**, not shown here, is much stricter in terms of what can match. It is included in the source code files so that the reader can compare it to the superior alternative **vfx:findMatchingPeople**.

```
<xsl:function name='vfx:adjust'>                                        0
<xsl:param name='s'/>
<xsl:value-of select='upper-case(normalize-space(string($s)))'/>
</xsl:function>

<xsl:function name='vfx:initial'>                                       1
<xsl:param name='s'/>
<xsl:value-of select='substring(normalize-space(string($s)),1,1)'/>
</xsl:function>

<xsl:function name='vfx:namesMatch' as='xs:boolean'>                    2
<xsl:param name='f1'/>
<xsl:param name='m1'/>
<xsl:param name='l1'/>
<xsl:param name='f2'/>
<xsl:param name='m2'/>
<xsl:param name='l2'/>
<xsl:value-of select='vfx:adjust($l1) = vfx:adjust($l2) and
                      vfx:adjust($f1) = vfx:adjust($f2) and
                      vfx:adjust($m1) = vfx:adjust($m2)'/>
</xsl:function>

<!-- Middle Name 'Initial Only' type of match -->                       3
<xsl:function name='vfx:namesMatchMNIO' as='xs:boolean'>
<xsl:param name='f1'/>
<xsl:param name='m1'/>
<xsl:param name='l1'/>
<xsl:param name='f2'/>
<xsl:param name='m2'/>
<xsl:param name='l2'/>
<xsl:value-of select='vfx:adjust($l1) = vfx:adjust($l2) and
                      vfx:adjust($f1) = vfx:adjust($f2) and
                      vfx:initial(vfx:adjust($m1)) =
                             vfx:initial(vfx:adjust($m2))'/>
</xsl:function>
```

auxfuncs.xsl Fragment 1 - Figure 7

The name comparison functions are built up from XPATH functions and are shown in the Figure 7. These functions can be found in **example/xslt/auxfuncs.xsl**.

- [F7H0] The **vfx:adjust** function is meant to take an argument convert it to a string, normalize white space and uppercase all characters. Note that the call to **string()** ensures that if the argument is a tag that does not appear in the source XML the result will be an empty string. Note also that white space normalization produces a string with no leading or trailing white space and all runs of white space characters in the argument will be reduced to a single space in the result. This function could be augmented to remove other extraneous characters that might appear in source data. For example, the period (.) character could be removed. The author has seen parentheses, colons, semicolons and numbers included in name parts. This is a good function to experiment with.
- [F7H1] The **vfx:initial** function is meant to return the first non-white space character in the argument.
- [F7H2] The **vfx:namesMatch** function compares two sets of name parts after **vfx:adjust** is applied. Comparing this way is an equivalence relation. It is reflexive, symmetric and transitive. This is the match function actually used in our example. The next function is included to show what can happen with heuristic matching methods.
- [F7H3] The **vfx:namesMatchMNIO** function is compares two sets of name parts but only requires that the middle names match on the first initial. This is riskier but also more likely to find a match. Consider comparing Jon J Smyth‘ with Jon James Smyth‘. Clearly there is a risk in declaring these to be equivalent. On the other hand, by treating this name pair as different we run the risk of creating two distinct **foaf:Person** s

where one would suffice. Use of this equivalence function is tempting. Although this relation is reflexive, symmetric and transitive, it is too coarse. For example Jon J Smyth‘ matches Jon James Smyth‘ and Jon Joseph Smyth‘ but Jon James Smyth‘ and Jon Joseph Smyth‘ are clearly not equivalent to a human reader but they are equivalent using this function.

The person name finding function illustrated in Figure 8 is in some ways just an elaborate version of the simpler organization match XPATH expression described above in Hightlight 1 of Figure 6. Notice that the name part matching function does not contain the heuristic middle initial weakness described above.

```
<xsl:function name='vfx:findMatchingPeople'>
<xsl:param name='fn'/>
<xsl:param name='mn'/>
<xsl:param name='ln'/>
<xsl:param name='nid'/>
<xsl:param name='ep'/>

<xsl:variable name='nidMatches'                                          0
   select='$ep/person[vfx:adjust(netid) = vfx:adjust($nid)]/uri'/>

<xsl:variable name='npMatches'                                           1
  select='$ep/person[vfx:namesMatch(fname,mname,lname,
                                    $fn,$mn,$ln) and string(netid) = ""]/uri'/>

<xsl:variable name='results' as='node()*'                                2
   select='if(vfx:adjust($nid) = "")
           then $npMatches
           else $nidMatches'/>

<xsl:value-of select='string($results[1])'/>                             3
</xsl:function>
```

auxfuncs.xsl Fragment 2 - Figure 8

- [F8H0] The variable **nidMatches** is the list of all matching person nodes from **Per0.xml** whose **netid**s match the nonempty netid (**nid**) of the person for whom we are searching. Note that there may be no matches. This is a normal XPATH predicate expression like that done for organizations.
- [F8H1] The variable **npMatches** is the list of all matching person nodes with no netid whose name parts match those of the person for whom we are searching. There may be no matches in this case as well.
- [F8H2] The variable **results** will contain the name part match list **npMatches** when the search subject has no netid. Otherwise it will contain the netid match list **nidMatches**. Use of the adjust function here deals with the case where a source **NETID** only contains whitespace.
- [F8H3] The function returns the first URI on the results list or the empty string. If there is more than one possible result then person triples have not been well maintained in VIVO or there are textually different but equivalent names with the same URI.

Start Previous Next