XSLT Ingest Example: Appendix E

Appendix E

In this section we consider the process used to create **Per0.xml** from the **foaf:Person** triples in VIVO. We begin with the **Sparql** query shown in the next figure. Note that the **aka** attribute has no role in the example.

- [F25H0] This is the section where prefixes are declared.
- [F25H1] In the **construct** clause the potential triple set is specified.
- [F25H2] This section of the where clause declares the mandatory triples:
  - **personURI** must be a foaf:Person and
  - have both a first and last name
- [F25H3] The optional triples are listed here: netid, middle name, label and any **aka:nameParts** triples. These express the list of aliases for the **personURI.**

```
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>      0
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX hr: <http://vivo.cornell.edu/ns/hr/0.9/hr.owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX vivoc: <http://vivo.library.cornell.edu/ns/0.1#>
PREFIX aka: <http://vivoweb.org/ontology/aka#>

construct {
       ?personURI rdfs:label ?personLabel .                      1
       ?personURI hr:netId ?netId .
       ?personURI foaf:firstName ?firstName .
       ?personURI vivo:middleName ?middleName .
       ?personURI foaf:lastName ?lastName .
       ?personURI aka:nameParts ?akaParts .

} where {

       ?personURI a foaf:Person .                                2
       ?personURI foaf:lastName ?lastName .
       ?personURI foaf:firstName ?firstName  .

       OPTIONAL {                                                3
              ?personURI hr:netId ?netId .
       }
        OPTIONAL {
              ?personURI vivo:middleName ?middleName  .
       }
       OPTIONAL {
              ?personURI rdfs:label ?personLabel .
       }
       OPTIONAL {
              ?personURI aka:nameParts ?akaParts .
       }
}
```

The Sparql Query for Per0.xml - Figure 25

This query is invoked with a selection of RDF/XML as the result set format. This XML is transformed into the form used in **Per0.xml** by the transform shown in the next two figures.

- [F26H0] Set up to collect a node list of person information in the **allpeeps** variable.
- [F26H1] Extract the URI from the **rdfsyn:about** attribute and

- [F26H1a] Create a person element from the **foaf**, **core** and **rdfs** elements and set the **aka** attribute to '**N**' indicating that this data is not from **aka:nameParts** triples;
- [F26H1b] For each aka:nameParts triple with this URI as a subject create a person element with the **aka** attribute set to '**Y**'.

```
<xsl:template match='/rdfsyn:RDF'>

<xsl:variable name='allpeeps'  as='node()*'>                                    0

<xsl:for-each select='./rdfsyn:Description'>
<xsl:sort select='lower-case(normalize-space(./foaf:lastName[1]))'/>

<xsl:variable name='uri' select='./@rdfsyn:about'/>                             1

<person aka='N'>                                                                a
<uri><xsl:value-of select='$uri'/></uri>
<lname><xsl:value-of select='vfx:stripSuffix(foaf:lastName[1])'/></lname>
<fname><xsl:value-of select='normalize-space(foaf:firstName[1])'/></fname>
<mname><xsl:value-of select='normalize-space(core:middleName[1])'/></mname>
<netid><xsl:value-of
       select='lower-case(normalize-space(hr:netId[1]))'/></netid>
<label><xsl:value-of select='normalize-space(rdfs:label[1])'/></label>
</person>

<xsl:for-each select='aka:nameParts'>                                           b
<person aka='Y'>
<uri><xsl:value-of select='$uri'/></uri>
<xsl:variable name='parts' select='tokenize(.,"\|")' as='xs:string*'/>
<lname><xsl:value-of select='vfx:stripSuffix($parts[1])'/></lname>
<fname><xsl:value-of select='normalize-space($parts[2])'/></fname>
<mname><xsl:value-of select='normalize-space($parts[3])'/></mname>
<netid><xsl:value-of select='normalize-space($parts[4])'/></netid>
<label><xsl:value-of select='normalize-space($parts[5])'/></label>
</person>
</xsl:for-each>

</xsl:for-each>
</xsl:variable>
```

Per0 XSLT Part 1 Figure 26

The next figure shows the portion of the transform that outputs the XML **Per0.xml** and the **vfx:stripSuffix** function used to remove such data from the last name of each person.

- [F27H0] This outputs the person elements that have last and first names.
- [F27H1] This function removes common suffixes from the end of a string. These often cause mismatches because they are employed inconsistently.

```
<ExtantPersons>
<xsl:for-each-group select='$allpeeps' group-by='uri'>
<xsl:for-each select='current-group()'>

<xsl:if test='string(lname) != "." and string(lname) != ""
              and string(fname) != ""'>
<xsl:copy-of select='.'/>
</xsl:if>

</xsl:for-each>
</xsl:for-each-group>
</ExtantPersons><xsl:text>
</xsl:text>
```

```
<xsl:function name='vfx:stripSuffix'>
<xsl:param name='ln'/>
<xsl:variable name='res'
       select='replace($ln,
                  "\s*(Jr|Jr.|Sr|Sr.|III|II|IV)\s*$","")'/>
<xsl:value-of
       select='normalize-space(replace($res,
                                       "\s*,\s*$",
                                       "")
                                )'/>
</xsl:function>
```

Per0 XSLT Part 2 Figure 27

Start Previous Next