Enriching profile pages using SPARQL query DataGetters

- Introduction
- The Steps and an Example
  - Step 1. Define the Customization
  - Step 2. Write the SPARQL Query
  - Step 3. Produce the N3 for the Data Getter
  - Step 4. Create a Freemarker Template
  - Step 5. Incorporate the New Template into the Application
  - Step 6. Create the Drill-down Page Using Page Management (optional)

Write SPARQL queries to provide additional information to profile pages, depending on the class of the individual.

Introduction

The VIVO software, beginning with version 1.6, supports the development of SPARQL query data getters that can be associated with specific ontological classes. These data getters, in turn, can be accessed within Freemarker templates to provide richer content on VIVO profile pages. For example, the profile page for an academic department lists only the names of the faculty within that department and their titles, but with a SPARQL query data getter it is now possible to extend the faculty information to display all of the faculty members' research areas.

The Steps and an Example

There are five mandatory steps involved in developing and implementing a class-specific SPARQL query data getter. In this wiki page we'll walk through an example and provide details on each of these steps.

1. Define the customization

2. Write the SPARQL query

3. Produce the N3 for the data getter

4. Create a Freemarker template

5. Incorporate the new template into the application

Step 1. Define the Customization

This first step might seem obvious but it's helpful to define as specifically as possible the change being made to VIVO. For our example, we'll use the one mentioned in the introduction. On academic department pages, we'll provide a list of all the faculty members' research areas and we'll display these beneath the department overview near the top of the page. In addition, we want the listed research areas to be links that will take us to a detail page that shows all of the faculty who have selected a given research area. This last requirement, being able to drill down to a details page, requires both an additional template and data getter, and so we'll need an optional sixth step: Create the Drill-down Page Using Page Management.

Step 2. Write the SPARQL Query

Having defined our requirements, we now need to write a query that will return the data we want -- specifically, the rdfs labels of the research areas and, because we want to be able to drill-down on these labels, the URI of the research areas. An obvious place to write and test a query is the SPARQL Query page that you can access from the Site Admin page. Here's our test query:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
SELECT DISTINCT (str(?researchAreaLabel) AS ?raLabel) ?ra
WHERE {
       <http://localhost:8080/individual/n2936> vivo:organizationForPosition ?posn .
       ?posn  vivo:positionForPerson ?person .
       ?person vivo:hasResearchArea ?ra .
       ?ra rdfs:label ?researchAreaLabel
}
ORDER BY ?raLabel
```

There are two points to note here. In line 3 of the query we convert the label variable to a string to prevent any duplicate labels from appearing; and in line 5 we use the specific URI for an academic department. This URI allows us to test the query, but it will have to be replaced by a "generic" subject in our next step.

Step 3. Produce the N3 for the Data Getter

Once the SPARQL query has been tested, we define the data getter using triples stored in a .N3 file. This file is then placed in the WEB-INF directory in the VIVO source code, as follows: `rdf/display/everytime/deptResearchAreas.n3`.

The N3 for our data getter consists of two parts: (1) the triple that associates our data getter with the AcademicDepartment class and (2) the triples that define the data getter itself. Here is the former:

```
<http://vivoweb.org/ontology/core#AcademicDepartment> display:hasDataGetter display:getResearchAreaDataGetter .
```

And here are the triples that define the `getResearchAreaDataGetter` data getter:

```
display:getResearchAreaDataGetter
    a <java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter>;
    display:saveToVar "researchAreaResults";
    display:query """
      PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
      PREFIX vivo: <http://vivoweb.org/ontology/core#>
      PREFIX afn:  <http://jena.hpl.hp.com/ARQ/function#>
      PREFIX foaf: <http://xmlns.com/foaf/0.1/>
      SELECT DISTINCT (str(?researchAreaLabel) AS ?raLabel) ?ra
      WHERE {
         ?individualURI vivo:relatedBy ?posn .
         ?posn a vivo:Position .
         ?posn  vivo:relates ?person .
         ?person a foaf:Person .
         ?person vivo:hasResearchArea ?ra .
         ?ra rdfs:label ?researchAreaLabel
      }
      ORDER BY ?raLabel
      """ .
```

Note that we have exchanged our specific department URI with the variable `?individualURI`. `?individualURI` is a "built-in" variable; that is, when the data getter is executed the value of this variable is set to the URI of the individual whose page is being loaded. So in our example, because we have associated the data getter with the AcademicDepartment class, when the IndividualController loads an academic department, the URI of that department gets set as the value of the `?individualURI` variable in our query.

Also note line 3 of the data getter:

```
display:saveToVar "researchAreaResults".
```

The "save to" variable `researchAreaResults` is what we use to access the query results in our template.

Step 4. Create a Freemarker Template

Now that we've created our data getter, `getResearchAreaDataGetter`, and have a "save to" variable with which to access the query results, we create a Freemarker template -- individual-dept-research-areas.ftl -- and use the <#list> function to loop through and display the results. The following markup is all that's needed in this new template.

```
<#if researchAreaResults?has_content>
    <h2 id="facultyResearchAreas" class="mainPropGroup">
        Faculty Research Areas}
    </h2>
    <ul id="individual-hasResearchArea" role="list">
        <#list researchAreaResults as resultRow>
            <li class="raLink">
                <a class="raLink" href="${urls.base}/deptResearchAreas?deptURI=${individual.uri}
&raURI=${resultRow["ra"]}" title="research area">
                    ${resultRow["raLabel"]}
                </a>
            </li>
        </#list>
    </ul>
</#if>
```

In the very first line we check to ensure that the query actually produced results. If not, no markup of any kind gets rendered. Otherwise, we give the new template section a heading, define an unordered list (<ul>) to contain the research areas, and then loop through the results. Note that the research area labels are contained within an anchor tag (<a>) because we want to be able to use these as links to a list of the faculty members for each research area. The URL in the href attribute includes what looks like a servlet name, `/deptResearchAreas`, and two parameters: `deptURI` and `raURI`. The `deptURI` parameter is the URI of the department that has been loaded by the IndividualController, and this value is accessible through the template variable ${individual.uri}. The `raURI` parameter is the URI of the research area, the value of which is available in our query results. These parameters and the servlet name will be used to develop the drill-down page that lists the faculty members in a department that have an interest in a specific research area.

Step 5. Incorporate the New Template into the Application

Now that we have a template to display the list of research areas, we need to update the individual.ftl template to source in the new template. Since individual.ftl is used to render individuals of many different classes, we include an <#if> statement to ensure that the individual-dept-research-areas.ftl template only gets included when the individual being loaded is an AcademicDepartment:

```
<#if individual.mostSpecificTypes?seq_contains("Academic Department")>
    <#include "individual-dept-research-areas.ftl">
</#if>
```

Step 6. Create the Drill-down Page Using Page Management (optional)

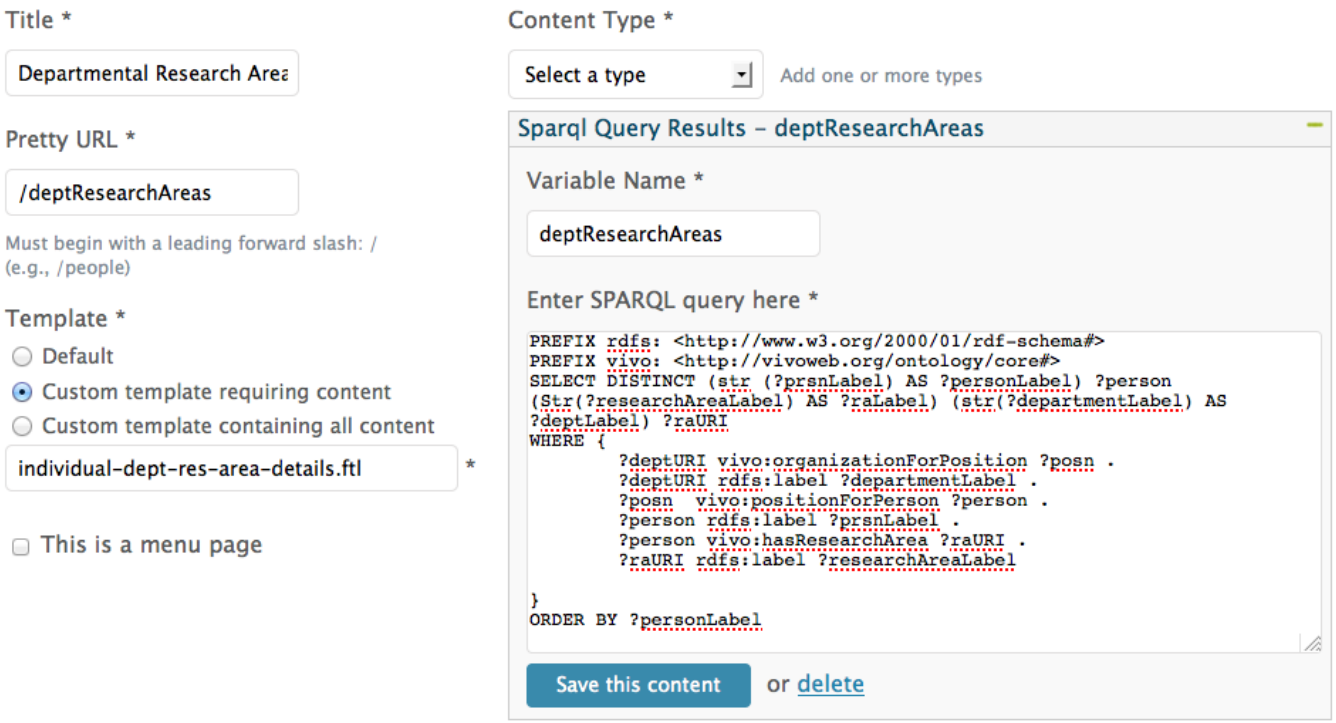
To this point, we have created a class-specific SPARQL query data getter, which retrieves the research areas of the faculty in a given academic department; developed a new template to render the results of our data getter; and updated the individual.ftl template to display the list of research areas. In Step 1, however, we defined requirements that include the ability to drill down from a selected research area to display a list of the faculty members in the department who have an interest in that research area. This is also done using a SPARQL query and new template. But in this case the query does not need to be associated with a specific class and defined in an .N3 file. Instead, we can create a SPARQL query page using the Page Management functionality.

As noted in Step 4, the anchor tags in the list of research areas include an `href` attribute that takes this format:

```
href="${urls.base}/deptResearchAreas?deptURI=${individual.uri}&raURI=${resultRow["ra"]}"
```

When creating the SPARQL query page in Page Management, as shown in the illustration below, we set the "Pretty URL" field to `/deptResearchAreas`. This portion of the `href` attribute, then, is not the name of an actual servlet but it effectively functions as one, and it is also associated with the template that we also define in Page Management: individual-dept-res-area-details.ftl. When a user clicks on one of the listed research areas, this is the template that the application will load.

Note the SPARQL query that is defined in the illustration below. It uses as variables the same parameters that are part of the `href` above: `deptURI` and `raURI`. Like the `?individualURI` discussed in Step 3, the values of these two parameters will become the values of the corresponding variables in the SPARQL query.



Now that the SPARQL query page has been created in Page Management, we still need to create the individual-dept-res-area-details.ftl template. Just as in Step 4, where we used the "save to" variable to access the query results in the individual-dept-research-areas.ftl template, we now use the variable defined in the "Variable Name" field (above) to access the results of that SPARQL query. Here is the content of the new template:

```
<#if deptResearchAreas?has_content>
    <section id="pageList">
        <#list deptResearchAreas as firstRow>
                <div class="tab">
                    <h2>${firstRow["raLabel"]}</h2>
                    <p>
                                        Here are the faculty members in the ${firstRow["deptLabel"]} department
with an interest in this research area.
                </p>
            </div>
            <#break>
        </#list>
    </section>
    <section id="deptResearchAreas">
        <ul role="list" class="deptDetailsList">
            <#list deptResearchAreas as resultRow>
                <li class="deptDetailsListItem">
                    <a href="${urls.base}/individual${resultRow["person"]?substring(resultRow["person"]?
last_index_of("/"))}"
                    title="faculty name">
                                                ${resultRow["personLabel"]}
                                        </a>
                </li>
            </#list>
        </ul>
    </section>
</#if>
```

Once again we use an <#if> statement to check for results. But this time we use the <#list> function twice: once to retrieve just the first row, which is used to provide a heading and some introductory text; and a second time to list all of the faculty members with an interest in the selected research area.