Custom List View Configurations

Introduction

Custom list views provide a way to expand the data that is displayed for object and data properties. For example, with the default list view the hasPresenterRole object property would only display the rdfs:label of the role individual; but with a custom list view, the "presentations" view includes not only the role but also the title of the presentation, the name of the conference where the presentation was given and the date the presentation was given. This wiki page provides guidelines for developing custom list views as well as an example of a custom list view.

List View Configuration Guidelines

Registering the List View

A custom list view is associated with an object property in the RDF files in the directory `/vivo/rdf/display/everytime`. To register a list view, create a new `.rdf` or `.n3` file in that directory. The file must be well-formed RDF/XML or N3.

Here is an example of registering a new association in a file named `newListViews.n3`:

```
<http://vivoweb.org/ontology/core#authorInAuthorship>
<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#listViewConfigFile>
"listViewConfig-authorInAuthorship.xml" .
```

With this triple the `authorInAuthorship` object property is associated with a list view configuration that is defined in a file named `listViewConfig-authorInAuthorship.xml`.

Place the N3 file containing this triple (or the well-formed RDF/XML file) in the `/vivo/rdf/display/everytime` directory, redeploy VIVO and restart tomcat to put the new custom list view in effect.

Note: Faux property custom list views are not registered in the same way. The list view is specified in the configuration of the faux property itself, using the faux property editing form. See details in Create and edit faux properties.

Required Elements

The list view configuration file requires three elements:

1. list-view-config: this is the root element that contains the other elements
2. query-select: this defines the SPARQL query used to retrieve data
3. template: this holds the name of the Freemarker template file used to display a single property statement

Optional Elements

The list-view-config root element can also contain two optional elements:

1. query-construct: one or more construct queries used to construct a model that the select query is run against
2. postprocessor: a Java class that postprocesses the data retrieved from the query before sending it to the template. If no post-processor is specified, the default post-processor will be invoked.

Construct Queries

Because SPARQL queries with multiple OPTIONAL clauses are converted to highly inefficient SQL by the Jena API, one or more construct queries should be included to improve query performance. They are used to construct a model significantly smaller than the entire dataset that the select query can be run against with reasonable performance.

The construct queries themselves should not contain multiple OPTIONAL clauses, to prevent the same type of inefficiency. Instead, use multiple construct queries to construct a model that includes all the necessary data.

In the absence of any construct queries, the select query is run against the entire dataset. If your select query does not involve a lot of OPTIONAL clauses, you do not need to include construct queries.

The construct queries must be designed to collect all the data that the select query will request. They can be flexibly constructed to contain more data than is currently selected, to allow for possible future expansion of the SELECT and to simplify the WHERE clause. For example, one of the construct queries for core:hasRole includes:

```
CONSTRUCT {
    ?role ?roleProperty ?roleValue .
    ...
} WHERE {
    ?role ?roleProperty ?roleValue .
    ...
}
```

That is, it includes all the properties of the role, rather than just those currently selected by the select query.

The ordering of the construct queries is not significant.

The Select Query

General select query requirements

Use a SELECT DISTINCT clause rather than a simple SELECT. There can still be cases where the same individual is retrieved more than once, if there are multiple solutions to the other assertions, but DISTINCT provides a start at uniqueness.

The WHERE clause must contain a statement ?subject ?property ?object, with the variables ?subject and ?property named as such. For a default list view, the ?object variable must also be named as such. For a custom list view, the object can be given any name, but it must be included in the SELECT terms retrieved by the query. This is the statement that will be edited from the edit links.

Data which is required in public view, optional when editing

Incomplete data can result in a missing linked individual or other critical data (such as a URL or anchor text on a link object). When the user has editing privileges on the page, these statements are displayed so that the user can edit them and provide the missing data. They should be hidden from non-editors. Follow these steps in the select query to ensure this behavior:

- Enclose the clause for the linked individual in an OPTIONAL block.
- Select the object's localname using the ARQ localname function, so that the template can display the local name in the absence of the linked individual. Alternatively, this can be retrieved in the template using the localname(uri) method.
- Require the optional information in the public view by adding a filter clause which ensures that the variable has been bound, inside tag <critical-data-required>. For example:

```
OPTIONAL { ?authorship core:linkedInformationResource ?infoResource }
```

- This statement is optional because when editing we want to display an authorship that is missing a link to an information resource. Then add:

```
<critical-data-required>
    FILTER ( bound(?infoResource) )
</critical-data-required>
```

- The Java code will preprocess the query to remove the <critical-data-required> tag, either retaining its text content (in public view) or removing the content (when editing), so that the appropriate query is executed.

Collated vs. uncollated queries

The query should contain <collated> elements, which are used when the property is collated. For uncollated queries, the fragments are removed by a query preprocessor. Since any ontology property can be collated in the Ontology Editor, all queries should contain the following <collated> elements:

- A ?subclass variable, named as such, in the SELECT clause. If the ?subclass variable is missing, the property will be displayed without collation.

```
SELECT DISTINCT <collated> ?subclass </collated> ...
```

- ?subclass must be the first term in the ORDER BY clause.

```
ORDER BY <collated> ?subclass </collated> ...
```

- Include the following in the WHERE clause, substituting in the relevant variables for ?infoResource and core:InformationResource:

```
<collated>
    OPTIONAL { ?infoResource a ?subclass
               ?subclass rdfs:subClassOf core:InformationResource .
    }
</collated>
```

Postprocessing removes all but the most specific subclass value from the query result set.

Alternatively (and preferably):

```
<collated>
    OPTIONAL { ?infoResource vitro:mostSpecificType ?subclass
               ?subclass rdfs:subClassOf core:InformationResource .
    }
</collated>
```

Automatic postprocessing to filter out all but the most specific subclass will be removed in favor of this implementation in the future.

Both collated and uncollated versions of the query should be tested, since the collation value is user-configurable via the ontology editor.

Datetimes in the query

To retrieve a datetime interval, use the following fragment, substituting the appropriate variable for ?edTraining:

```
OPTIONAL {
    ?edTraining core:dateTimeInterval ?dateTimeInterval
    OPTIONAL { ?dateTimeInterval core:start ?dateTimeStartValue .
               ?dateTimeStartValue core:dateTime ?dateTimeStart
    }
    OPTIONAL { ?dateTimeInterval core:end ?dateTimeEndValue .
               ?dateTimeEndValue core:dateTime ?dateTimeEnd
    }
}
```

The variables ?dateTimeStart and ?dateTimeEnd are included in the SELECT clause.

Many properties that retrieve dates order by end datetime descending (most recent first). In this case, a post-processor must apply to sort null values at the top rather than the bottom of the list, which is the ordering returned by the SPARQL ORDER BY clause in the case of nulls in a descending order. In that case, the variable names must be exactly as shown to allow the post-processor to do its work.

The Template

To ensure that values set in the template on one iteration do not bleed into the next statement:

- The template should consist of a macro that controls the display, and a single line that invokes the macro.
- Variables defined inside the macro should be defined with <#local> rather than <#assign>.

To allow for a missing linked individual, the template should include code such as:

```
<#local linkedIndividual>
    <#if statement.org??>
        <a href="${url(statement.org)}">${statement.orgName}</a>
    <#else>
        <#-- This shouldn't happen, but we must provide for it -->
        <a href="${url(statement.edTraining)}">${statement.edTrainingName}</a> (no linked organization)
    </#if>
</#local>
```

The query must have been constructed to return orgName (see above under "General select query requirements"), or alternatively the template can use the localname function: ${localname(org)}.

If a variable is in an OPTIONAL clause in the query, the display of the value in the template should include the default value operator ! to prevent an error on null values.

List View Example

This example will walk through the custom list view for the core:researchAreaOf object property. This property is displayed on the profile page for research area individuals.

Associate the property with a list view

In this example we're using RDF/XML to associate the researchAreaOf object property (line 1) with a specific list view configuration (line 2):

```
<rdf:Description rdf:about="http://vivoweb.org/ontology/core#researchAreaOf">
    <display:listViewConfigFile rdf:datatype="http://www.w3.org/2001/XMLSchema#string">listViewConfig-
researchAreaOf.xml</display:listViewConfigFile>
</rdf:Description>
```

The list view configuration

The root <list-view-config> element in our listViewConfig-researchAreaOf.xml file contains the required <query-select> and <template> elements as well as two optional <query-construct> sections and an optional <postprocessor> element.

This is the <query-select> element:

```
    <query-select>
        PREFIX afn:  &lt;http://jena.hpl.hp.com/ARQ/function#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX vitro: &lt;http://vitro.mannlib.cornell.edu/ns/vitro/0.7#&gt;
        PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt;

        SELECT DISTINCT
                       ?person
                       ?personName
                       ?posnLabel
                       ?orgLabel
                       ?type
                       ?personType
                       ?title
        WHERE {
                 ?subject ?property ?person .
                 ?person core:personInPosition ?position .
                 OPTIONAL { ?person rdfs:label ?personName }
                 OPTIONAL { ?person core:preferredTitle ?title }
                 OPTIONAL { ?person vitro:mostSpecificType ?personType .
                            ?personType rdfs:subClassOf foaf:Person
                 }
                 OPTIONAL { ?position rdfs:label ?posnLabel }
                 OPTIONAL { ?position core:positionInOrganization ?org .
                            ?org rdfs:label ?orgLabel
                 }
                 OPTIONAL { ?position core:hrJobTitle ?hrJobTitle }
                 OPTIONAL { ?position core:rank ?rank }
        }
        ORDER BY ?personName ?type
    </query-select>
```

Here is the first <query-construct> element:

```
    <query-construct>
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;

        CONSTRUCT {
            ?subject ?property ?person .
            ?person core:personInPosition ?position .
            ?position rdfs:label ?positionLabel .
            ?position core:positionInOrganization ?org .
            ?org rdfs:label ?orgName .
            ?position core:hrJobTitle ?hrJobTitle
        } WHERE {
            {
                ?subject ?property ?person
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position rdfs:label ?positionLabel
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:positionInOrganization ?org
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:positionInOrganization ?org .
                ?org rdfs:label ?orgName
            } UNION {
                ?subject ?property ?person .
                ?person core:personInPosition ?position .
                ?position core:hrJobTitle ?hrJobTitle
            }
        }
    </query-construct>
```

The second <query-construct> element:

```
    <query-construct>
        PREFIX rdfs: &lt;http://www.w3.org/2000/01/rdf-schema#&gt;
        PREFIX core: &lt;http://vivoweb.org/ontology/core#&gt;
        PREFIX foaf: &lt;http://xmlns.com/foaf/0.1/&gt;
        PREFIX vitro: &lt;http://vitro.mannlib.cornell.edu/ns/vitro/0.7#&gt;

        CONSTRUCT {
            ?subject ?property ?person .
            ?person rdfs:label ?label .
            ?person core:preferredTitle ?title .
            ?person vitro:mostSpecificType ?personType .
            ?personType rdfs:subClassOf foaf:Person
        } WHERE {
            {
                ?subject ?property ?person
            } UNION {
                ?subject ?property ?person .
                ?person rdfs:label ?label
            } UNION {
                ?subject ?property ?person .
                ?person core:preferredTitle ?title
            }  UNION {
                ?subject ?property ?person .
                ?person vitro:mostSpecificType ?personType .
                ?personType rdfs:subClassOf foaf:Person
            }
        }
    </query-construct>
```

Next is the required <template> element:

```
<template>propStatement-researchAreaOf.ftl</template>
```

And here is the <postprocessor> element:

```
<postprocessor>edu.cornell.mannlib.vitro.webapp.web.templatemodels.individual.ResearchAreaOfPostProcessor<
/postprocessor>
```

Note: the <postprocessor> is included here only to show the syntax. The actual listViewConfig-researchAreaOf.xml file in the VIVO code base does not use a custom post-processor.

The Freemarker Template

Finally, here are the contents of our Freemarker template, propStatement-researchAreaOf.ftl.

```
<#import "lib-sequence.ftl" as s>
<@showResearchers statement />
<#-- Use a macro to keep variable assignments local; otherwise the values carry over to the
     next statement -->
<#macro showResearchers statement>
    <#local linkedIndividual>
        <a href="${profileUrl(statement.uri("person"))}" title="${i18n().person_name}">${statement.personName}<
/a>
    </#local>
    <#if statement.title?has_content >
        <#local posnTitle = statement.title>
    <#else>
        <#local posnTitle = statement.posnLabel!statement.personType>
    </#if>
    <@s.join [ linkedIndividual, posnTitle, statement.orgLabel!"" ] /> ${statement.type!}
</#macro>
```