

Creating short views of individuals

- [Overview](#)
 - [What does it do?](#)
 - [How is it created?](#)
- [Details](#)
 - [The class association](#)
 - [The customViewForIndividual definition](#)
 - [The SparqlQueryDataGetter definition](#)
 - [The Freemarker template](#)
 - [The default template can be modified in the theme](#)
- [Some examples](#)
 - [The scenario](#)
 - [SEARCH example](#)
 - [The default template](#)
 - [Specifying the custom short view](#)
 - [INDEX example](#)
 - [The default template](#)
 - [Specifying the custom short view](#)
 - [BROWSE example](#)
 - [The default template](#)
 - [Specifying the custom short view](#)
- [Troubleshooting](#)
 - [Errors in the template?](#)
 - [Errors in the Query?](#)
 - [Errors in the config file?](#)
- [Notes](#)
 - [Waiting for the Application and Display Ontology](#)
 - [Classes are not inferred](#)
 - [More than one applicable short view](#)
 - [Hard-coded BROWSE view for People](#)

Overview

What does it do?

Custom short views are used in three different contexts within VIVO, to give you more control over how an individual is displayed in that context.

You can configure VIVO to display different classes of individuals in different ways. As an example, you might choose to display a Faculty Member in a grey color if she has no current appointments.

Custom short views will frequently be different in the three different contexts in which they are available. For example, you might want to show the image of a Person on a search result, but you might not want to display that image in the Person index pages.

How is it created?

A short view is defined by two elements. First there will be a group of RDF statements in this file:

```
vitro/webapp/web/WEB-INF/resources/shortview_config.n3
```

In a VIVO release, this file is empty (except for a few comments), and the default short views are used in all cases. You will add RDF to this file as you define your custom short views. The RDF statements will name the class of Individual, the Freemarker template, and any SPARQL queries that are used to get the data you need to display.

The other thing you will need is the Freemarker template itself, to render your custom view.

An example of some custom short views can be found in this directory:

```
vivo/utilities/acceptance-tests/suites/ShortViews/
```

The directory contains a copy of shortview_config.n3, and some Freemarker templates. These files are essentially the same as the examples on this page.

Details

The class association

A statement associates a custom short view with a VIVO Class from the ontology. For example:

```
vivo:FacultyMember
  display:hasCustomView    mydomain:facultySearchView .
```

This means that the specified short view will be used for any Individual that has `vivo:FacultyMember` as a most specific class.

Only the most specific classes of an Individual are recognized by the short views. So if you want a custom short view to be used for all Person objects, you must define it on Person and on FacultyMember and on FacultyMemberEmeritus, etc.

The customViewForIndividual definition

An object is given a URI and declared to be a `customViewForIndividual`

It may apply to one or more of the contexts: `SEARCH`, `INDEX`, or `BROWSE`. It must have an associated template, and may have one or more associated DataGetters.

Here is an example:

```
mydomain:facultySearchView
  a                display:customViewForIndividual ;
  display:appliesToContext "SEARCH" ;
  display:hasTemplate  "view-search-faculty.ftl" ;
  display:hasDataGetter mydomain:facultyDepartmentDG .
```

This custom view applies in the `SEARCH` context. It specifies a DataGetter, which will be invoked to find data each time this short view is rendered. It also specifies the Freemarker template that will render the view.

The SparqlQueryDataGetter definition

The DataGetter must also be defined in the RDF, like this:

```
mydomain:facultyDepartmentDG
  a                datagettters:SparqlQueryDataGetter ;
  display:saveToVar "details" ;
  display:query     """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vivo: <http://vivoweb.org/ontology/core#>
    SELECT ?deptName
    WHERE {
      ?individualUri vivo:hasMemberRole      ?membership .
      ?membership    vivo:roleContributesTo ?deptUri .
      ?deptUri
        a                vivo:AcademicDepartment ;
        rdfs:label ?deptName .
    }
    LIMIT 20
    """ .
```

Besides the type and the URI, this object specifies a SPARQL query, and the name of a Freemarker variable where the results of the query will be stored.

When the SPARQL query is executed, the value of `?individualUri` will be bound to the actual URI of the Individual being displayed. The values returned from the query will be stored in an array of Freemarker Hash containers, with each one representing a row of the SPARQL query result. The Hash will contain the values returned by the query, keyed to the variable names used in the query.

When this array of Hash containers has been constructed, it is stored in the variable named by the DataGetter declaration; `details` in this case.

The DataGetter is optional in a custom short view. If no DataGetter is specified, then the Freemarker template will only have the standard set of data available to it.

Conversely, multiple DataGetters may be specified on a short view. If this is done, each DataGetter should assign to a different Freemarker variable, to avoid problems with overwriting data.

The Freemarker template

A default template exists for each of the short view contexts: `SEARCH`, `INDEX` and `BROWSE`.

If no custom short view is defined for an Individual, the default template will be used to render the Individual.

The custom template will likely be based on the default template for that context. For example, the default template for search results is called `view-search-default.ftl` and looks like this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<p class="snippet">${individual.snippet}</p>
```

Our modified template is this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<#if (details[0].deptName)?? >
    <span class="display-title">Member of ${details[0].deptName}</span>
</#if>

<p class="snippet">${individual.snippet}</p>
```

So, if a department name was found for this Faculty member, it will be displayed. If more than one was found, the remainder will be ignored, since the template only displays the first one.

The default template can be modified in the theme

Besides taking advantage of custom short views, the theme author may also choose to override the templates for the default short views. This would merely require creating a new template with the same name as the one being overridden, and putting this new template into the template directory of your theme.

Some examples

The scenario

When a `FacultyMember` appears in a short view, we would like to add the name of his/her department. This information isn't directly available, so we will need to obtain it from a SPARQL query.

This should work in all three contexts, `SEARCH`, `INDEX`, and `BROWSE`.

This will only apply to `FacultyMembers`. Other individuals will use the default short views.

If the `FacultyMember` is not a member of a department, the short view should just omit the name, without causing an error.

SEARCH example

The default template

The default short view for the `SEARCH` context looks like this:

And it produces results like this:

Search results for 'Faculty'

[Dog, Charlie](#) | Faculty Member

... Dog Charlie Chair 123 Midway Street Brooktondale New York Member Age

[Baker, Able](#) | Faculty Member

... Instructor Instructor Afghanistan Instructor Agent **Faculty** Member Person

Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

```
vivo:FacultyMember
  display:hasCustomView    mydomain:facultySearchView .

mydomain:facultySearchView
  a                        display:customViewForIndividual ;
  display:appliesToContext "SEARCH" ;
  display:hasTemplate      "view-search-faculty.ftl" ;
  display:hasDataGetter    mydomain:facultyDepartmentDG .

mydomain:facultyDepartmentDG
  a                        datagettters:SparqlQueryDataGetter ;
  display:saveToVar        "details" ;
  display:query            """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vivo: <http://vivoweb.org/ontology/core#>
    SELECT ?deptName
    WHERE {
      ?individualUri vivo:hasMemberRole      ?membership .
      ?membership    vivo:roleContributesTo  ?deptUri .
      ?deptUri
        a            vivo:AcademicDepartment ;
        rdfs:label   ?deptName .
    }
    LIMIT 20
    """ .
```

Create the template `view-search-faculty.ftl` to look like this:

```
<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<#if (details[0].deptName)?? >
  <span class="display-title">Member of ${details[0].deptName}</span>
</#if>

<p class="snippet">${individual.snippet}</p>
```

The new search results look like this:

Search results for 'faculty'

[Dog, Charlie](#) | Faculty Member | Member of Art Department
... Dog Charlie Chair 123 Midway Street Brooktondale New York Member Ag

[Baker, Able](#) | Faculty Member
... Instructor Instructor Afghanistan Instructor Agent **Faculty** Member Perso

INDEX example

The default template

The default short view for the INDEX context looks like this:

```
<#import "lib-properties.ftl" as p>
<a href="${individual.profileUrl}" title="name">${individual.name}</a>
<@p.mostSpecificTypes individual />
```

And it produces results like this:

Faculty Member | [RDF](#)

[Baker, Able](#)

[Dog, Charlie](#)

Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

```

vivo:FacultyMember
  display:hasCustomView    mydomain:facultyIndexView .

mydomain:facultyIndexView
  a                        display:customViewForIndividual ;
  display:appliesToContext "INDEX" ;
  display:hasTemplate      "view-index-faculty.ftl" ;
  display:hasDataGetter    mydomain:facultyDepartmentDG .

mydomain:facultyDepartmentDG
  a                        datagetters:SparqlQueryDataGetter ;
  display:saveToVar        "details" ;
  display:query            """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vivo: <http://vivoweb.org/ontology/core#>
    SELECT ?deptName
    WHERE {
      ?individualUri vivo:hasMemberRole      ?membership .
      ?membership    vivo:roleContributesTo ?deptUri .
      ?deptUri
        a            vivo:AcademicDepartment ;
        rdfs:label    ?deptName .
    }
    LIMIT 20
    """ .

```

Note that the DataGetter is the same as in the previous example. If two custom short views want to use the same DataGetter, there is no need to code it twice. If both of these examples are tried at the same time, the two custom short views would refer to the same DataGetter.

Create the template `view-index-faculty.ftl` to look like this:

```

<#import "lib-vivo-properties.ftl" as p>

<a href="${individual.profileUrl}" title="individual name">${individual.name}</a>

<@p.displayTitle individual />

<#if (details[0].deptName)?? >
  <span class="display-title">Member of ${details[0].deptName}</span>
</#if>

```

The new index looks like this:

Faculty Member | RDF

Baker, Able

Dog, Charlie | Member of Art Department

BROWSE example

The default template

The default short view for the BROWSE context looks like this:

```

<#import "lib-properties.ftl" as p>

<li class="individual" role="listitem" role="navigation">

<#if (individual.thumbUrl??>
  
  <h1 class="thumb">
    <a href="${individual.profileUrl}" title="${i18n().view_profile_page_for}
      ${individual.name}}">${individual.name}</a>
  </h1>
<#else>
  <h1>
    <a href="${individual.profileUrl}" title="${i18n().view_profile_page_for}
      ${individual.name}}">${individual.name}</a>
  </h1>
</#if>

<#assign cleanTypes =
  'edu.cornell.mannlib.vitro.webapp.web.TemplateUtils$DropFromSequence'?new()(individual.mostSpecificTypes,
vclass) />
<#if cleanTypes?size == 1>
  <span class="title">${cleanTypes[0]}</span>
<#elseif (cleanTypes?size > 1) >
  <span class="title">
    <ul>
      <#list cleanTypes as type>
        <li>${type}</li>
      </#list>
    </ul>
  </span>
</#if>
</li>

```

Notice that it contains some conditional logic, producing different results depending on whether there is an image file associated with the Individual, or whether the type being browsed is the most specific type for the individual.

The default template produces results like this:

Faculty Member

▶ All
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



[Baker, Able](#)

[Dog, Charlie](#)

Or this:

Person

▶ [All](#) A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



[Baker, Able](#)

Faculty Member

[Dog, Charlie](#)

Faculty Member

Specifying the custom short view

In the `shortview_config.n3` configuration file, create these structures:

```

vivo:FacultyMember
  display:hasCustomView    mydomain:facultyBrowseView .

mydomain:facultyBrowseView
  a                        display:customViewForIndividual ;
  display:appliesToContext "BROWSE" ;
  display:hasTemplate      "view-browse-faculty.ftl" ;
  display:hasDataGetter    mydomain:facultyDepartmentDG ;
  display:hasDataGetter    mydomain:facultyPreferredTitleDG .

mydomain:facultyDepartmentDG
  a                        datagettters:SparqlQueryDataGetter ;
  display:saveToVar        "details" ;
  display:query            """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vivo: <http://vivoweb.org/ontology/core#>
    SELECT ?deptName
    WHERE {
      ?individualUri vivo:hasMemberRole      ?membership .
      ?membership    vivo:roleContributesTo  ?deptUri .
      ?deptUri
        a            vivo:AcademicDepartment ;
        rdfs:label   ?deptName .
    }
    LIMIT 20
    """ .

mydomain:facultyPreferredTitleDG
  a                        datagettters:SparqlQueryDataGetter ;
  display:saveToVar        "extra" ;
  display:query            """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX vivo: <http://vivoweb.org/ontology/core#>
    SELECT ?pt
    WHERE {
      ?individualUri <http://vivoweb.org/ontology/core#preferredTitle> ?pt
    }
    LIMIT 1
    """ .

```

Note that the first DataGetter is the same as in the previous examples. If two custom short views want to use the same DataGetter, there is no need to code it twice. If two or more of these examples are tried at the same time, the short views would each refer to the same DataGetter.

Also note that this short view uses two DataGetters. One stores its results in "details" and the other stores its results in "extra", so the freemarker template will have access to both sets of results.

Create the template `view-browse-faculty.ftl` to look like this:

```

<#import "lib-properties.ftl" as p>

<li class="individual" role="listitem" role="navigation">

<#if (individual.thumbUrl)??>
  
  <h1 class="thumb">
    <a href="${individual.profileUrl}" title="View the profile page for
      ${individual.name}">${individual.name}</a>
  </h1>
<#else>
  <h1>
    <a href="${individual.profileUrl}" title="View the profile page for
      ${individual.name}">${individual.name}</a>
  </h1>
</#if>

<#if (extra[0].pt)?? >
  <span class="title">${extra[0].pt}</span>
<#else>
  <#assign cleanTypes =
    'edu.cornell.mannlib.vitro.webapp.web.TemplateUtils$DropFromSequence'?new()(individual.
mostSpecificTypes, vclass) />
  <#if cleanTypes?size == 1>
    <span class="title">${cleanTypes[0]}</span>
  <#elseif (cleanTypes?size > 1) >
    <span class="title">
      <ul>
        <#list cleanTypes as type>
          <li>${type}</li>
        </#list>
      </ul>
    </span>
  </#if>
</#if>

<#if (details[0].deptName)?? >
  <span class="title"><em>Member of</em> ${details[0].deptName}</span>
</#if>

</li>

```

The new browse results look like this:

Faculty Member

► [All](#) [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)



[Baker, Able](#)

[Dog, Charlie](#)

Member of Art Department

Or this:

Person

▶ [All](#) A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



[Baker, Able](#)

Faculty Member

[Dog, Charlie](#)

Faculty Member

Member of Art Department

Troubleshooting

Errors in the template?

If the freemarker template for a short view contains syntax errors, the request will not throw an exception, which will be written to the VIVO log (vivo.all.log). The page will still render, but with an error message in place of the intended short view.

For example, in search results:

Search results for 'faculty'

Can't process the custom short view for Dog, Charlie

Can't process the custom short view for Baker, Able

In an index page:

Faculty Member | [RDF](#)

Can't process the custom short view for Baker, Able

Can't process the custom short view for Dog, Charlie

In browse results:

Faculty Member

▶ **All** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Can't parse the short view template 'view-browse-faculty.ftl' for Baker, Able

Can't parse the short view template 'view-browse-faculty.ftl' for Dog, Charlie

Errors in the Query?

If the SPARQL query defined in the configuration file contains syntax errors, the log will contain a stack trace for the exception. The information in the exception may be cryptic, but will at least tell you where the error is located within the query.

For example:

```
2012-10-23 17:25:26,499 ERROR [FreemarkerHttpServlet] com.hp.hpl.jena.query.QueryParseException:
  Encountered " <VAR1> "?individualUri "" at line 6, column 1.
  Was expecting:
  "{" ...
```

The page will not render properly. Instead it will show a standard error screen:

[Home](#) | [People](#) | [Organizations](#) | [Research](#) | [Events](#)

There was an error in the system.

Return to the [home page](#)

Errors in the config file?

Other errors in the configuration file may give less obvious results. For example, if your customView object calls for a data getter that does not exist, the page will attempt to render without that data. If the data from that data getter is optional, you will see no error indicator except for a message in vivo.all.log

Notes

Waiting for the Application and Display Ontology

This implementation of short views is intended to be temporary, pending the implementation of the Application and Display Ontology (A&DO).

Much of the RDF that is entered in the configuration file (`shortview_config.n3`) should be replaced by triples in the A&DO. It's not clear where the SPARQL queries will be specified.

Classes are not inferred

Short views are applied based on the most-specific classes of the Individual. No inference is done when trying to find applicable views. So if an Individual has a type of `FacultyMember`, then a short view that applies to `Person` will not be used. Even though the Individual should also have a type of `Person`, it will not be among the most specific types for the Individual, and so does not apply. If you want a short view to apply to all sub-classes of `Person`, you must explicitly list each of these sub-classes in `shortview_config.n3`

This is expected to change when the Application and Display Ontology is used.

More than one applicable short view

In theory, it is possible that an Individual may qualify for two short views simultaneously. An Individual could have two most specific types (say `FacultyMember` and `ExemptEmployee`), and both of those types might have configured short views. Or, in the degenerate case, there might be multiple short views configured for a single type.

In these cases, one of the applicable short views will be arbitrarily selected.

Hard-coded BROWSE view for People

In the `BROWSE` context, if no short view is found for a given class URI, but that class is included in the `People` classgroup, a hard-coded short view is applied. This is to maintain compatibility with previous versions.

It is hoped that the Application and Display Ontology will be expressive enough to configure this behavior within the standard mechanism. Until then, it is coded into the class `edu.cornell.mannlib.vitro.webapp.services.shortview.FakeApplicationOntologyService`